Authentication and Authorization Protocol Security Property Analysis with Trace Inclusion Transformation and Online Minimization

Yating Hsu and David Lee
Department of Computer Science and Engineering, The Ohio State University
Columbus, OH 43210, USA
{hsuya,lee}@cse.ohio-state.edu

***Abstract* – A major hurdle of formal analysis of protocol security properties is the well-known state explosion – a protocol system usually contains infinitely many or a formidable number of states. As a result, most of the analysis resorts to heuristics, such as state space pruning. Given the temporal property of authentication and authorization protocols, we introduce trace inclusion transformation of protocol specification to reduce significantly the state space. We further cut down the number of states by online minimization for obtaining a model of a manageable size for a formal and rigorous analysis. However, the two state space reduction procedures may result in false negative and false positives. We show that our trace inclusion transformation and online minimization do not introduce any false negative. On the other hand, we design an efficient algorithm for ruling out all the possible false positives. Therefore, our analysis is sound and complete.**

For a case study, we analyze OAuth, a standardization of API authentication protocols. Our automated analysis identifies a number of attacks in the original specification, including the one that has been detected. We also analyze the second version of OAuth and prove it is secure if the API interface is secure.

***Index terms* – authentication and authorization, communicating extended finite state machine, trace inclusion transformation, online minimization, and OAuth**

I. INTRODUCTION

With the advancement of Internet technologies and business opportunities web applications have led to unprecedented integrated services. A web application may create user specific services by acquiring user data from an external data server. For the web application to access user data at the data server, it requires the web application to sign in at the data server with user's login credential, usually username and password. However, it is undesirable for the user to reveal his username and password. From this and other applications' needs, API (Application Programming Interface) authentication and authorization protocols are developed for web application to obtain user data without user credential.

Various vendors offer their own API authentication and authorization protocols, such as AOL OpenAuth [29], Yahoo! BBAuth [35], and Flickr Authentication API [30]. OAuth [32] is an effort for the standardization of the prevalent API authentication protocols. There are three communicating entities in OAuth: Service Provider, Consumer and User. Service Provider hosts user data, and consumer is the web application that needs access to user data. The basic idea of API authentication and authorization protocols is to use a data structure, called *token,* that decouples the access right from user login credential [32, 36]. Consumer obtains a Request Token from Service Provider and redirects User to Service Provider to authorize the Request Token. Service Provider first authenticates User and then asks User to authorize the Consumer to access his data. After successful User authorization, Consumer exchanges the Authorized Request Token for an Access Token with Service Provider and then accesses User data with the Access Token. It is an authentication and authorization process; Service Provider authenticates the identity of User and Consumer, and User authorizes Consumer with the access right.

After the publication of first version of OAuth v1.0, a *Session Fixation* attack was discovered [32, 36] and then fixed in the second version of OAuth v1.0A. It motivates our work to study security properties of both OAuth versions.

An authentication or authorization property can often be formulated as a temporal property: one type of action e (authentication and authorization) should precede another action d (private data access) [26, 27]. Naturally, such property analysis can be reduced to a reachability analysis in a transition system. For instance, OAuth authorization can be stated as: action e of User authorization must precede action d of Consumer data access. Equivalently, if transition e is removed from the system, then transition d is no longer reachable from the initial system state. However, this problem is in general undecidable, and the reachability graph can be infinite or of a formidable size due to the well-known state explosion problem. Therefore, we need to effectively reduce the state space first.

To cope with state space explosion we first introduce trace inclusion transformation and then apply online minimization to construct a minimal reachable graph of manageable size while still preserving all the executable traces. Specifically, given a transition system M with a set of states and transitions among the states, we want to verify if M satisfies certain temporal property. We first conduct a trace inclusion transform on M by simplifying certain system parameters and variables and obtain $\widehat{M}$ that preserves all the execution traces of M. However, the state space of $\widehat{M}$ may still be infinite or too large to construct. We further apply online minimization to reduce the state space to a manageable size by merging states, which are bisimulation equivalent, and hence the execution traces of the original transition system are still preserved. Nevertheless, the two state space reduction processes may introduce false negative and false positives. We show that our

978-1-4244-8645-8/10/$26.00 ©2010 IEEE

procedures do not cause any false negative and we present an efficient algorithm to filter out all the false positives.

As a case study and experiment, we formally model OAuth by communicating extended finite state machine, and then apply our procedures for an automated analysis. The original state space is infinite since there could be infinitely many variable and parameter values. Our procedures effectively reduce the number of states to *67* for OAuth v1.0 and *44* states for v1.0A. We first analyze both versions with an assumption that all the API interfaces are secure (HTTPS). We discover various traces that lead to Session Fixation attacks in v1.0, including the one that was detected [32, 36]. We show that v1.0A is indeed secure. However, if the API interfaces are insecure (HTTP), there are various attacks on both versions. To the best of our knowledge there is no published work on rigorous formal analysis of OAuth yet.

Our approach for formal analysis of protocol security properties is rather general and is applicable to a range of properties, including authorization, so long as they can be formulated as a temporal property.

After a brief review of related works in Section II, we describe a formal model and the desired security properties in Section III. We present our trace inclusion transformation and online minimization procedures in Section IV and V. We show that the two state space reduction procedures do not result in any false negative and provide an efficient algorithm for filtering out all the false positives in Section VI. Finally, in Section VII we report the experimental results of an automated analysis of OAuth.

II. Related Work

It is commonly acknowledged that the design and analysis of security protocols are difficult and prone to errors, which are extremely hard to detect [20]. A lot of research efforts have been devoted to formally analyzing security protocols. Various logic [4, 24] and state space based [15] approaches were developed to facilitate analysis of security protocols. Works in [5,6] use language MSR to formally specify Kerberos as first order atomic formulas and verify various properties. However, logic based approaches are often with daunting and tedious syntax and rules. Lowe [20] models Needham-Schroeder protocol with CSP and FDR along with a model checker for finding an attack in the protocol. Murφ [22, 23] is a protocol verification tool that checks a specification against a desired property by explicit state enumeration and is hampered by the state space explosion [22, 23]. Interrogator [15] and NRL Protocol Analyzer [15] use backward search to avoid explicitly constructing the whole state space.

State minimization is a natural approach to deal with state space explosion. However, one cannot afford to construct the whole state space of a transition system and then minimize it. Online minimization [18, 28, 2, 3, 7] is to perform reachability analysis and minimization at the same time without constructing the whole state space.

Protocol model simplification and transformation have been studied. Work in [13] defines the notion of safe transformation on CSP protocol models that has the property to preserve insecurity after transformation. [21] provides sufficient conditions under which checking simplified systems is enough. Their transformation is applicable to CSP models whereas our transformation and minimization are on general transition systems with protocol data portions. Data independence techniques are explored [25] to find "threshold" for variables in a protocol model to simplify the analysis.

For API authentication protocols, Mashup [10,17] has a similar design goal as OAuth. However, its applications require that user reveals his credentials. Various frameworks, such as MashupOS [12], SMash [16] and OMash [8], have been proposed for its security. OpenID [33, 34] allows users to sign in at different websites using only one digital identity. InfoCard [1] allows users to manage its digital identities with guaranteed security.

III. Formal Model And Security Properties

We model a communication protocol system as a collection of communicating extended finite state machines (CEFSM) [19]. A protocol system with n principals can be modeled as a CEFSM $M=(M_1, M_2, \ldots, M_n, M_A)$ with $n+1$ component machines where each component machine M_i, $1 \le i \le n$, is a communicating principal and M_A is an attacker. Each component machine is an Extended Finite State Machine (EFSM) that is a quintuple

$$M = (I, O, S, \vec{x}, T)$$

where I, O, S, $\vec{x}$, and T are the finite sets of input symbols, output symbols, states, variables, and transitions, respectively. Each transition t in T is a six-tuple

$$t = (s_t, q_t, a_t, o_t, P_t, A_t)$$

where s_t, q_t, a_t and o_t are the current state, next state, input and output, respectively. $P_t(\vec{x})$ is a predicate on the current variable values, and $A_t(\vec{x})$ defines an action on variable values. Each combination of a state and variable values is called a configuration. To specify the input and output actions among component machines, we use the notations from CSP [11]. A message msg sent from M_i to M_j is an output operation $M_j!msg$ in M_i and a matching input operation $M_i?msg$ in M_j. The matching operations are synchronized between M_i and M_j.

For the attacker model M_A, we follow Dolev-Yao [9] and use a full knowledge [23] attacker model. The attacker could be a legitimate participant in a protocol [22]. In each run of a protocol, an attacker may take actions of other participants. For example, in Needham-Schroeder authentication protocol, an attacker can take the actions of the initiator or the responder. Similarly, in OAuth, an attacker can take the role of: (1) both User and Consumer; (2) User but not Consumer; (3) Consumer but not User; (4) Neither User nor Consumer.

Given a formal model M of a protocol, we want to verify if M satisfies certain authentication or authorization property. Usually, such property can be formulated as a temporal property [26]: an action d occurs only after an action e. For instance, in OAuth, Consumer accesses User data (action d)

only after User authorizes the access (action *e*). To verify such property, we want to check that for every execution trace in *M*, action (transition) *d* is always preceded by action (transition) *e*. It can be reduced to a reachability analysis of *M*: Is the transition *d* reachable from the initial state without going through transition *e*? Or, equivalently, if we remove transition *e*, is transition *d* still reachable from the initial state? Given the predicates and actions on the transitions, to check the reachability in a CEFSM is rather complex [19]. A common approach is to construct a reachability graph from the CEFSM for an analysis. A reachability graph consists of all the reachable configurations of the CEFSM, each of which consists of a global state and a set of variable values. Initially, the protocol is in an initial global state where each component is in its initial state, along with a set of initial variable values. This is called the initial configuration of a protocol system. We want to construct a reachability graph from *M* that consists of all the configurations, which are reachable from the initial configuration. Note that a transition in a reachability graph does not contain any predicate or action on variable values anymore; the information is contained in the current and next configuration of a transition. Given the reachability graph, we can verify a temporal property as follows. We remove all the transitions, which are derived from (authorization) action *e,* and check whether any transition from (data access) action *d* is still reachable from the initial configuration. This is the case if and only if the temporal property is violated. It can be easily checked by a graph search.

However, a protocol system can have several sessions (runs), which are simultaneously executed. For instance, in OAuth there could be a normal session by User, Consumer and Service Provider. Meanwhile an attacker could also initiate a session as a User that is executed with Consumer and Service Provider. Such simultaneous and possibly interleaved executions of several sessions appropriately model complicated yet real protocol system operations with attacker involved. This can be modeled by a Cartesian product of protocol machines and that further explodes the state space.

In this paper we propose two approaches to reduce the size of the reachability graph to tackle the state explosion problem: (1) Trace Inclusion Transformation. It simplifies the input, output, predicate and action on transitions of CEFSM while all the executable traces in the original CEFSM are preserved in the transformed machine. (2) Online Minimization. It merges configurations together according to bisimulation equivalence yet without first constructing a reachability graph. After the state space reduction, we analyze the reduced protocol system and prove security properties.

IV. Assumptions and Trace Inclusion Transformation

There are two key aspects of API protocol systems: secure interfaces and stateful processing. For instance, HTTPS is secure and HTTP is not in general. If each communication party keeps track of messages exchanged for verification in processing, then it is stateful, otherwise, it is stateless. Both versions of OAuth do not require secure API interfaces or stateful processing. Relaxation of any one of the two assumptions may lead to attacks with the current OAuth specifications. However, the majority of the implementations is stateful and uses HTTPS, which experienced system engineers consider as default choices. We assume that the API protocols are stateful and all the interfaces are secure. Later in Subsection VII.F, we discuss the case when the two assumptions are relaxed.

Informally, an attack of a protocol machine *M* is an execution trace from the initial state to a certain state with a security property violation.

Definition 1. A trace $v_0,t_{0,1},v_1,t_{1,2},\ldots t_{n-1,n},v_n$ consists of a sequence of states $v_0, v_1, \ldots, v_n$ and transitions $t_{0,1}, t_{1,2},\ldots t_{n-1,n}$ in *M* where transition $t_{i,i+1}$ moves from state v_i to v_{i+1}. A trace from the initial configuration with initial state v_0 and initial variable value $\overrightarrow{x_0}$ is executable if and only if for all *i*: $0 \le i \le n-1$, the predicate of transition $t_{i,i+1}$ on the current variable values $\overrightarrow{x_i}$ is true where the action of the transition $t_{i,i+1}$ updates the variable values from $\overrightarrow{x_i}$ to $\overrightarrow{x_{i+1}}$.

Definition 2. An attack trace in *M* is an executable trace from the initial configuration to a state with a security property violation.

For instance, in OAuth if there is an executable trace from the initial state to a state of User's data access by Consumer without going through User authorization transition, then it is a security property (authorization) violation and hence is an attack trace.

Given the complexity of real application protocols, we want to first simplify it to reduce the state space to make it feasible to identify attack traces. For such simplification, we have to preserve all the executable traces, which include the attack traces, which are to be detected later. Such desirable simplification is called trace inclusion transformation. We first describe a special yet important case of trace inclusion transformation.

Message content adds complexity to formal protocol analysis. For instance, a message may contain session related fields, such as source and destination IP address, nonce and timestamp, whose values may not affect the result of security property analysis. We want to abstract out some system parameters, message fields and variables without losing any executable traces – a special case of trace inclusion transformation.

We determine abstraction functions α: $I \rightarrow \alpha(I)$ and β: $O \rightarrow \beta(\mathrm{O})$ that remove irrelevant message fields from the input and output of transitions. We also define a function ω: $\vec{x} \rightarrow \omega(\vec{x})$ that is a transformation of variables. By transformations of α, β and ω, we transform a CEFSM $M=(M_1, M_2, \ldots, M_n, M_A)$ to CEFSM $\widehat{M}=(\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n}, M_A)$, where EFSM $M_i=(I, \mathrm{O}, S_i, \overrightarrow{x_i}, T_i)$, $1 \le i \le n$, is transformed to

$$\widehat{M_i}=(\alpha(I), \beta(\mathrm{O}), S_i, \omega(\overrightarrow{x_i}), \widehat{T_i}).$$

Note that M_i and $\widehat{M_i}$ have the same states and transitions, but the input and output on transition are simplified according to α and β, and the predicate and action are changed by ω. A

transition $t=(s_t, q_t, a_t, o_t, P_t, A_t) \in T_i$ is transformed to $\hat{t} \in \hat{T}_i$:

$$\hat{t}=(s_t, q_t, \alpha(a_t), \beta(o_t), \hat{P}_t, \widehat{A_t}),$$

and γ: $P_t \rightarrow \hat{P}_t$ and δ: $A_t \rightarrow \widehat{A_t}$ are the derived transformations from ω.

Given an EFSM, its transition diagram is a graph where the nodes are the states and the edges are the transitions. A transition diagram models the control portion of an EFSM. Obviously, with the above transformations on input, output and variables, the transition diagram of EFSM $M_i=(I, O, S_i, \overrightarrow{x_i}, T_i)$ and $\widehat{M_i}=(\alpha(I), \beta(O), S_i, \omega(\overrightarrow{x_i}), \hat{T}_i)$ are isomorphic.

Definition 3. Two EFSMs E and $\hat{E}$ are isomorphic in control portion if their transition diagrams are isomorphic.

It can be easily shown:

Proposition 1. Given CEFSMs $M=(M_1, M_2, \ldots, M_n)$ and transformations of input, output, variable, predicate and action on its component EFSMs, the corresponding transformed CEFSM $\widehat{M}=(\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n})$ is isomorphic in control portion to M if each component EFSM $\widehat{M_i}$ is isomorphic to M_i in control portion, $1 \le i \le n$.

It is crucial that after the transformations the executable traces of the simplified CEFSM $\widehat{M}$ include all of the executable traces of the original machine M so that the transformations preserve all the attack traces to be analyzed. Recall that if EFSM $\hat{E}$ and E are isomorphic in control portion after transformations, they have isomorphic transition diagram. Since a trace is specified by a sequence of interleaved states and transitions, each trace in E has a unique corresponding trace in $\hat{E}$.

Definition 4. An EFSM $\hat{E}$ is a trace inclusion transformation from EFSM E if: (1) $\hat{E}$ and E are isomorphic in control portion with the transformations on input, output, variables, predicate and action; and (2) For each executable trace of E, the corresponding transformed trace in $\hat{E}$ is also executable.

After a trace inclusion transformation from EFSM E to $\hat{E}$, each executable trace in E is also an executable trace in $\hat{E}$ and hence the name: trace inclusion transformation. Since a transition in a CEFSM is executable if all the associated transitions in the component machines are executable, we have:

Proposition 2. With trace inclusion transformations on each component machine of CEFSM $M=(M_1, M_2, \ldots, M_n, M_A)$ from M_i to $\widehat{M_i}$, $1 \le i \le n$, the resulting CEFSM $\widehat{M}=(\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n}, M_A)$ is also a trace inclusion transformation from M.

There are various transformations which are trace inclusive. We discuss a special and practically important case where the transformation $\omega^*(\vec{x})$ on variables is a projection to a subset of variables of $\vec{x}$ and part of the predicate $P_t(\vec{x})$ in the original protocol model is independent of the variables other than those in $\omega^*(\vec{x})$ where $P_t(\vec{x})$ can be decomposed as:

$$P_t(\vec{x}) = \hat{P}_t(\omega^*(\vec{x})) \wedge P'_t(\vec{x})$$

where $\wedge$ is the logical "and" operator. A commonly used transformation is to disregard the clause $P'_t(\vec{x})$ and keep action $A_t(\vec{x})$ unchanged – an identity mapping. The derived transformations are:

$$\gamma^*: P_t(\vec{x}) \rightarrow \hat{P}_t(\omega^*(\vec{x}))$$
$$\delta^*: A_t(\vec{x}) \rightarrow A_t(\vec{x}) \quad (1)$$

Proposition 3. The mapping functions of input, output, variables, predicates and actions in (1) provide a trace inclusion transformation of CEFSM $M=(M_1, M_2, \ldots, M_n, M_A)$ to $\widehat{M}=(\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n}, , , M_A)$.

Proof. We want to show that for any executable trace c in M the corresponding trace $\hat{c}$ in $\widehat{M}$, with the transformed input, output, variables, predicate and action, is also executable. We prove by induction on the length l of trace c.

Let l=1. Suppose the initial variable values of $\vec{x}$ and $\omega^*(\vec{x})$ are $\overrightarrow{x_0}$ and $\overrightarrow{x'_0}$, respectively. Since the length of c is 1, c starts at the initial state v_0, follows a transition $t=(v_0, v_1, a_t, o_t, \hat{P}_t(\overrightarrow{x'_0}) \wedge P'_t, A_t(\overrightarrow{x_0}))$ and moves to v_1. Therefore, $\hat{P}_t(\overrightarrow{x'_0})$ and P'_t must be both true. Since the corresponding predicate on $\hat{c}$ is $\hat{P}_t(\overrightarrow{x'_0})$ and hence is also true, and $\hat{c}$ is executable. Since δ^* is an identity mapping, the variable values in the next state v_1 after transition c and $\hat{c}$ are identical.

Inductively, for all executable trace c of length l in M that goes through the states $v_0, v_1, \ldots, v_l$, there is an executable trace $\hat{c}$ of length l in $\widehat{M}$ that also goes through the states $v_0, v_1, \ldots, v_l$ in the same order. Also, the variable values of $\omega^*(\vec{x})$ at v_l in M are the same at that at v_l in $\widehat{M}$. We show that for any trace of length l+1, the statement still holds.

For an executable trace c of length l+1 in M that goes through the states $v_0, v_1, \ldots, v_l, v_{l+1}$, the subtrace c' of length l going through $v_0, v_1, \ldots, v_l$ is also an executable trace in M. By the inductive hypothesis, there is an executable trace $\widehat{c'}$ in $\widehat{M}$ of length l and going through states $v_0, v_1, \ldots, v_l$, and the end variable values $\vec{x}$ are the same as that in M after c'. When the trace c' extends from v_l to v_{l+1} through transition t, the predicate $P_t(\vec{x}) = \hat{P}_t(\vec{x'}) \wedge P'_t(\vec{x})$ is true, and hence $\hat{P}_t(\vec{x'})$ must be also true. Therefore, the corresponding trace $\hat{c}$ in $\widehat{M}$ of length l+1 is also executable on the last transition t with the same end variable values, since the action on transition t is an identity mapping. ■

How to determine the transformations α, β, ω, γ and δ? The general rule is: all the executable traces are preserved after the transformations. Further, we want to simplify a transition system as much as we can. For instance, we can abstract out all the parameters and variables so long as they are irrelevant to the security properties under consideration.

Given the simplified machine $\widehat{M}$, however, the reachability graph of $\widehat{M}$ may still be too large or even infinite. In the next section, we apply online minimization to build a minimal reachable graph G^* that is bisimulation equivalent to the reachability graph of $\widehat{M}$ and preserves the executable and hence attack traces for an analysis.

V. Online Minimization and Minimal Reachable Graph

Given a protocol system modeled as a CEFSM $M=(M_1, M_2, \ldots, M_n, M_A)$, we first make a trace inclusion transformation on M and obtain a simplified machine $\widehat{M}$. However, its reachability graph $\hat{G}$ may still be infinite or too large to analyze and we want to minimize it. However, there is no way to construct the reachability graph and then merge equivalent states for minimization. We apply an online minimization procedure [18], that is, we construct a minimized reachability graph G^*, called minimal reachable graph, directly from the original transition system without first constructing the reachability graph $\hat{G}$. A CEFSM is a transition system and we can apply the online minimization procedure in [18]. Here the state equivalence is bisimulation equivalence [14, 27] so that trace information is preserved. Therefore, an analysis on the minimal reachable graph G^* is equivalent to that on the reachability graph $\hat{G}$. It is crucial since our security property analysis is based on protocol and attack traces.

A theory and general procedure on online minimization were published in [18] and algorithms for real-time transition systems were further developed in [28]. For our application, we develop online minimization algorithms of CEFSM for our applications.

Given a CEFSM $\widehat{M}$, instead of exploring all possible configurations, we group the configurations into blocks so that configurations in a same block undergo a same sequence of state transitions in $\widehat{M}$. Initially, we start with a set of blocks where each (global control) state of $\widehat{M}$ and all possible variable values forming a block of configurations. Then we split blocks according to unstable transitions, that is, separate configurations to different blocks, which are bisimulation inequivalent. Specifically, a transition t from a block B to C is stable if every configuration in B is mapped by transition t into C. Otherwise, it is unstable, and we split the domain block B for stabilization. We repeat this process on all blocks that are reachable from the initial configuration until all transitions are stable. All the reachable blocks along with the stable transitions among them form a minimal reachable graph G^* that is bisimulation equivalent to the reachability graph $\hat{G}$ of $\widehat{M}$.

Algorithm 1 (Minimal Reachable Graph for CEFSM)

Input: CEFSM $\widehat{M}=(\widehat{M_1}, \widehat{M_2}, \ldots, \widehat{M_n}, M_A)$, s_0=initial control states of $\widehat{M}$, v_0=initial variable values, B_0=all possible variable values.
Output: a minimal reachable graph G^*.
begin

```
1.  S_G={<s, B_0> | s=control state of M̂};
2.  mark <s_0, B_0>; mark s_0 and v_0 in B_0;
3.  stack={<s_0, B_0>};      /* reachable blocks */
4.  queue=∅;                 /* unstable blocks to be split */
5.  while (stack≠ ∅)
6.     while (stack≠ ∅)
7.        <s, B> = stack.pop();
8.        v = marked variable values in B;
9.        for each outgoing transition tr: <s, B>→<s', B'> from <s, B>
10.          if (tr is not stable and <s, B>∉ queue)
11.             queue.enqueue(<s, B>);
12.          if (<s', B'> is not marked)
13.             v'= variable values after tr from v;
14.             mark s' and v' in B';
15.             stack.push(<s', B'>);
16.    while (queue ≠ ∅)
17.       <s, B> = queue.dequeue();
18.       for each outgoing tr from <s, B>
19.          split <s, B> into <s, B'> and <s, B''> so that
             <s, B>→ <s, B'> is stable;
20.          <s, B> =<s, B'>;
21.          add <s, B''> to S_G;
22.          for each incoming tr: <u, C> →<s, B> from
             marked u
23.             if (tr is not stable and <u, C> ∉ queue)
24.                queue.enqueue(<u, C>)
25.             if (<s, B> is not reachable from marked
                variable value of C by tr)
26.                delete tr;
27.             if (<s, B''> is reachable from marked
                variable value of C from tr)
28.                add transition <u, C>→<s, B''>;
29.                if (<s, B''> not marked)
30.                   v''= variable values reachable from
                      marked variable values of C
31.                   mark <s, B''>; mark v'' in B'';
32.                   stack.push(<s, B''>);
33.       if (stack≠ ∅)
34.          break;
35.  remove unmarked blocks in S_G;
36.  G*=S_G and transitions among marked configurations
End
```

In Algorithm 1, the first while loop (line 6-15) searches for reachable configurations starting from the initial configuration (initial control global states and initial variable values) and marks them. The second while loop splits domain variable value blocks for each outgoing unstable transition (line 18) and incoming unstable transition (line 22). We repeat this process until there is no unstable state.

The online minimization by Algorithm 1 merges bisimulation equivalent configurations and hence any two equivalent configurations in G^* can be reached from the initial configuration through a same transition sequence in $\widehat{M}$. On the other hand, CEFSM $\widehat{M}$ and its reachability graph $\hat{G}$ have the same executable traces. Consequently, the constructed minimal reachable graph G^* contains the same executable traces as $\hat{G}$. We have:

Proposition 4. A reachable minimal graph G^* from the online minimization Algorithm 1 of a CEFSM $\widehat{M}$ contains the same executable traces as $\widehat{M}$.

Recall that the essence of authentication and authorization property analysis of a protocol system is on its executable traces. From Proposition 2 and 4, we can reduce the analysis of $\widehat{M}$ and hence the original machine M to that on the reachable minimal graph G^*.

VI. Security Property Analysis - False Negatives and False Positives

Given an original CEFSM M, we apply trace inclusion transformation and then online minimization and obtain a minimal reachable graph G^* that is of a manageable size and contains all the executable traces of M. With G^* we can verify the desired authentication and authorization property, such as: action e (authentication and authorization) must always precede action d (User data access). Equivalently, none of the transitions (states) of action d is reachable from the initial state of M after removing all the transitions (states) of action e. This can be done by a simple graph search of G^*.

However, we may have introduced false negative and false positives due to the two state space reduction procedures. We show that trace inclusion transformation and online minimization do not introduce any false negative. We also present an efficient algorithm to filter out all the false positives.

A. False Negative

A protocol system has a desired security property if its CEFSM M does not contain any attack traces on that property. We have shown that all the traces, including the attack traces, of M are included in the traces of the minimal reachable graph G^* after trace inclusion transformation and online minimization. Therefore, if G^* does not contain any attack traces, then M does not have any attack traces either and hence is secure. That is, a security property analysis of G^* does not introduce any false negative:

Proposition 5. For temporal security properties, including authentication and authorization, an analysis of a minimal reachable graph of a CEFSM, after trace inclusion transformation and online minimization, does not introduce any false negative.

Sketch of Proof. We prove by contradiction. Suppose that there is no attack trace in the minimal reachable graph G^* but there is an attack trace τ in the original CEFSM M that is an executable trace from the initial state to an attack state d (e.g. accessing to private data) without going through any authentication and authorization transitions e. Then it is also an executable trace after a trace inclusion transformation to $\widehat{M}$, following the same states and transitions in both machines. Since $\widehat{M}$ and its reachability graph $\hat{G}$ have the same executable traces and G^* is bisimulation equivalent to $\hat{G}$, τ must be an executable attack trace in G^*- a contradiction.

From the proof, we conclude:

Corollary 1. Every attack trace of the original CEFSM M is also an attack trace in G^*.

B. False Positive

From Corollary 1, if the minimal reachable graph of a CEFSM does not contain any attack traces, then the protocol system is secure. However, the converse may not be true. That is, an attack trace in G^* may not be an attack trace in M. If this happens then there is a false positive from an analysis of G^*. From Corollary 1, we only have to examine all the attack traces from G^* and rule out all the possible false positives. This can be checked directly on the original CEFSM M:

Algorithm 2 (Checking False Positives)

Input: Original CEFSM M and an attack trace $c=v_0,t_{0,1},v_1,t_{1,2},\ldots t_{n-1,n},v_n$ from G^*
Output: whether c is an attack trace on M.
begin
1. $\overrightarrow{x_0}$ = initial variable value of $\vec{x}$;
2. $i = 0$;
3. **if** (*isExecutable*(M, i, $v_0,t_{0,1},v_1,t_{1,2},\ldots t_{n-1,n},v_n$, $\overrightarrow{x_0}$))
4. **return** c is an attack trace in M
5. **else**
6. **return** c is a false attack trace

end

subroutine *isExecutable*(M, i, $v_i,t_{i,i+1},v_{i+1},\ldots t_{n-1,n},v_n$, $\overrightarrow{x_t}$)
7. **if** (i==n)
8. **return true**;
9. **if** ($P_t(\overrightarrow{x_t})$==**false**)
10. **return false**;
11. **else if for** *any input on* $t_{i,i+1}$ (*isExecutable*(M,$i+1$,$v_{i+1},t_{i+1,i+2},v_{i+2},\ldots t_{n-1,n},v_n$, $\overrightarrow{x_{t+1}}$))
12. **return true**;
13. **return false**

In Algorithm 2, given an attack trace c from G^*, we restore the input, output, predicate and action on the transitions along the trace and check if c is executable in M (line 3). If c is not executable in M, then it is a false positive (line 6). We use a subroutine *isExecutable* to recursively check if the trace from state v_i to v_n is executable in M, given the variable values $\overrightarrow{x_t}$ at v_i in M. If the predicate P_t of transition $t_{i,i+1}$ on $\overrightarrow{x_t}$ is false (line 9), then the transition $t_{i,i+1}$, hence the trace, from state v_i to v_n is not executable. While restoring the original input and output for each transition (line 11), there can be a range of possible values (e.g. parameters). If any one of the values makes the transition executable, then we move on – the trace is executable so far. However, if for all the possible input values the transition is not executable, then the trace is not executable – blocked at the current transition.

C. Simultaneous Execution Traces

On a protocol system there can be a number of simultaneous sessions – protocol runs, and their execution traces can be interleaved. For instance, an attacker can initiate a session while a normal session is in process. The two execution sequences may reach a same configuration (same system state and variable/parameter values), and the attacker can follow the execution of the normal protocol run and launch an attack.

To model the interleaved behaviors of m simultaneous sessions we can take the Cartesian product of m CEFSMs, each of which is a protocol run. Or, equivalently, we can examine the Cartesian product of their corresponding minimal

reachable graphs.

D. Summary of Our Approach

We now summarize our approach. Given a protocol system, we model it by a CEFSM M. We first reduce its state space by trace inclusion transformation and then by online minimization (Algorithm 1), obtaining a reachable minimal graph G^*.

We then determine simultaneous runs of the protocol system. The general rule is: we make as few runs as possible yet covering all possible attacks. It depends on the application environment and practical constraints. For instance, it can be proved that one attacker is equivalent to two or more attackers in the general Dolev-Yao model [9]. For OAuth, four simultaneous sessions cover all possible attack scenarios (see Section III). Suppose that m runs are needed. We take the Cartesian product of m minimal reachable graph G^* for an analysis.

We examine the temporal properties of the security protocol under investigation, such as authentication and authorization. We identify type e transitions (states) which correspond to the actions that must be taken (e.g. authentication and authorization) before any type d transitions (states) which are from the actions to be granted after satisfying the security requirements (e.g. private data access).

Finally, we remove all the type e transitions and check whether any of the type d transitions is still reachable from the initial system state. If none of them are reachable, the security property is verified. Otherwise, for every executable trace from the initial configuration to a type d transition, check whether it is executable using Algorithm 2; if it is executable then an attack trace is identified, otherwise, it is a false positive.

VII. A Case Study: OAuth

In this section, we apply our general methods for an analysis of OAuth.

A. A Formal Model of OAuth

OAuth is an API (Application Programming Interface) authentication protocol for a standardization effort. The goal of OAuth is to let User delegate access right at a Service Provider to a web or desktop application without revealing User's login credential at the Service Provider. The workflow of OAuth is [32]:

1. Consumer, upon receiving the request from the User, sends a request to Service Provider's Request Token URL to ask for a Request Token.
2. Service Provider grants Consumer an Unauthorized Request Token.
3. Consumer redirects User to Service Provider's User Authorization URL.
4. Service Provider authenticates User and obtains consent (authorization) from User.
5. After User authorizes Consumer's Request Token, Service Provider redirects User back to Consumer's callback URL.
6. Consumer requests Service Provider for an exchange of the Authorized Request Token for an Access Token.
7. Service Provider grants Consumer an Access Token.
8. Consumer uses the Access Token to access data on behalf of User.

The fist version of OAuth (OAuth 1.0) was published in December 2007. A security flaw was found in January 2009. A revised version OAuth 1.0A is posted in June 2009. The basic workflow of the protocol remains the same. The main differences are as follows. In OAuth 1.0, after User authorization, Service Provider redirects User back to Consumer. In OAuth 1.0A, when Service Provider redirects User back to Consumer, Service Provider issues a verifier to User. When User is redirected back to Consumer, Consumer obtains the verifier from User. Then Consumer must use the Authorized Request Token and also the verifier to exchange for an Access Token.

We now apply our general procedures to formally analyze both versions of OAuth. We first construct EFSMs M_{SP}, M_C and M_U for Service Provider, Consumer and User from the OAuth specification [32]. There are 4, 5 and 4 states in the M_{SP}, M_C and M_U, respectively, as shown in Figure 1. For readability, only the input and output of the transitions are shown. Service Provider has variables to keep track of Consumer and User information, Unauthorized Request Token, Authorized Request Token, Access Token, callback URL of Consumer, timestamps and nonce. Consumer has variables for Unauthorized Request Token, Authorized Request Token, Access Token, timestamps and nonce. User has variables for Unauthorized Request Token and Authorized Request Token. In OAuth 1.0A, all three parties have the same variables as in version 1.0, and there is an additional verifier.

For the attacker model M_A, the initial knowledge includes the Service Provider's Request Token URL, User Authorization URL, Access Token URL, and the necessary identity information so that he can make request to Service Provider as a Consumer. There are two types of API interfaces and attacker's capabilities:

(1) All communication channels (API) are secure and all communicating parties are mutually authenticated. The attacker can be a legitimate User or Consumer, but it cannot impersonate Service Provider or eavesdrop the messages exchanged between other communicating principals. It is a weak form of attacker model. Most of the OAuth Service Providers require HTTPS for API interfaces that falls into this category.

(2) A strong attacker model complies with the well-known Dolev-Yao model. We assume that communication channels (API) are not secure and communicating parties are not mutually authenticated. The attacker can have full control of the network; it can intercept, delete, insert or modify any messages in transit. There are some OAuth Service Providers that accept insecure HTTP for API interfaces [31], and the HTTPS constraint can be easily bypassed [37].

Given $M=(M_{SP}, M_C, M_U, M_A)$, the attacker M_A can be from a

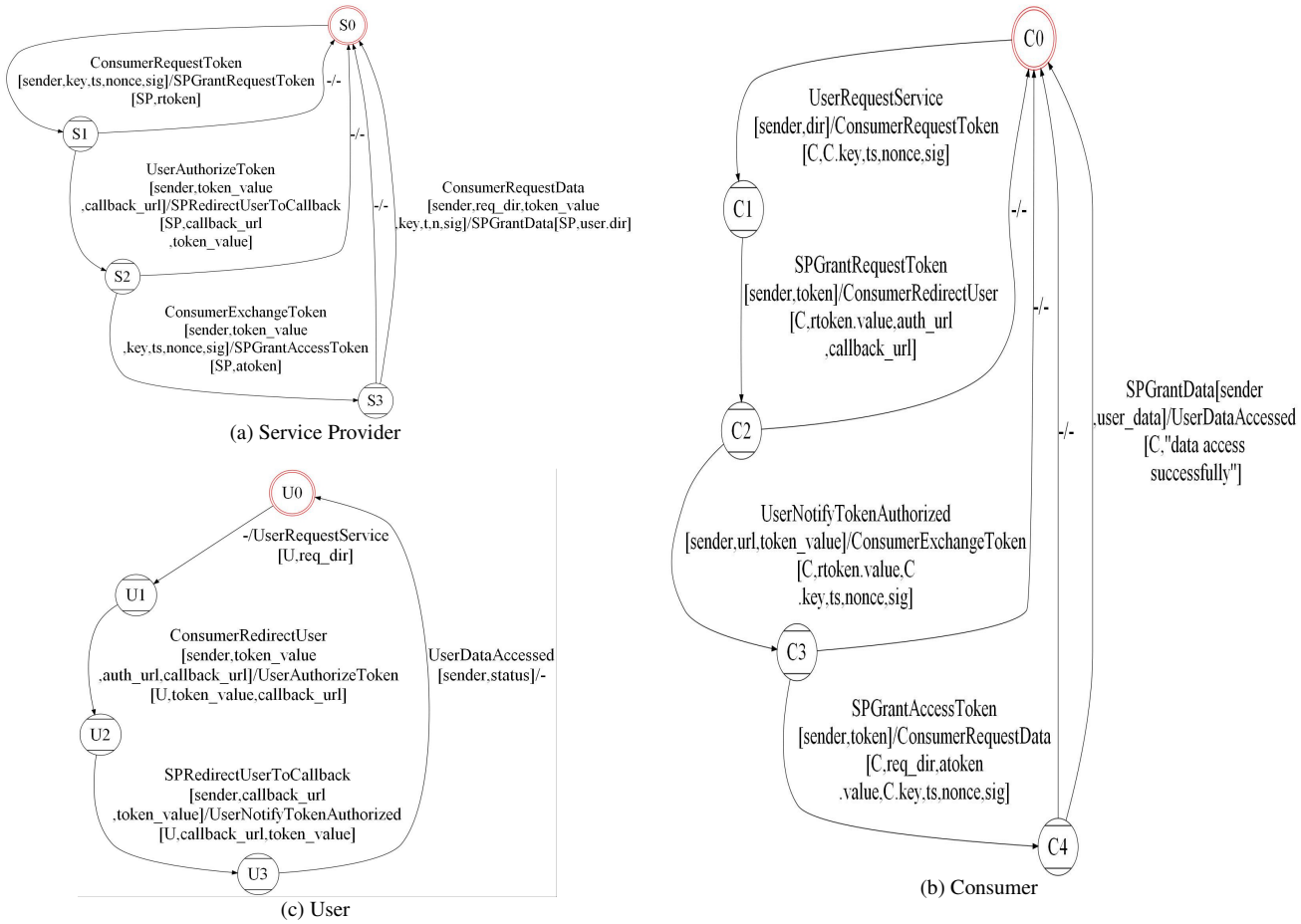

Fig. 1. EFSMs of OAuth (a) Service Provider, (b) Consumer, and (c) User

weak model M_{WA} or a strong one M_{SA}. We first discuss in detail the first case of weak model M_{WA}, that is, all the API interfaces are secure with HTTPS; most of the OAuth implementations and the majority of API application follow this model. We have implemented all our algorithms and apply them to a formal CEFSM model of OAuth for an automated analysis of its security properties as follow.

B. Security Properties of OAuth

We verify the following five security properties of OAuth. The first three properties are on the authorization process and the last two are on User service security.

Property P_1: If a Consumer accesses User's data with an Access Token, then Service Provider must have issued the Access Token to this Consumer.

To check P_1, we first remove the transitions in the constructed minimal reachable graph G^* of Service Provider issuing Access Token. We then check whether the transitions that represent Consumer accessing data are still reachable from the initial system state.

Property P_2: If a Consumer obtains an Access Token by exchanging a Request Token, then the Request Token must have been authorized by User.

To check P_2, we remove the transitions of User authorization and check if the transitions for Consumer to exchange for Access Token are still reachable from the initial system state.

Property P_3: If a Consumer obtains an Access Token by exchanging a Request Token, then the Service Provider must have issued the Request Token to this Consumer.

To check P_3, we remove the transitions of Service Provider issuing Request Token and check if the transitions for Consumer to obtain Access Token are still reachable from the initial system state.

Property P_4: If a Consumer accesses data that belong to User, then that User must have authorized the access.

To check P_4, we remove the transitions of User authorization and check if the transitions for Consumer accessing User data are still reachable from the initial system state.

Property P_5: If a Consumer accesses data that belong to a User, then only this User can get the service from Consumer with the accessed data – not any other User.

After the Request Token is authorized, the User who has authorized the access and is redirected back to Consumer is the User who gets the service provided with the accessed data.

Thus to check P_5, we remove the transitions where User authorizes the access or is redirected back to Consumer. Then we check if the transitions of Consumer accessing data are still reachable from the initial system state.

C. Trace Inclusion Transformation

Given a CEFSM model M of OAuth, we first define the abstraction function that simplifies M and produces a trace inclusion transformation $\widehat{M}$. In OAuth, Consumer has to generate a nonce and a timestamp for each request made to Service Provider, and append a digital signature to the request. Since all the API interfaces are secure with HTTPS, all the tokens are securely exchanged among Service Provider, Consumer and User. Therefore, we define α and β as the functions that remove timestamps, nonce and signatures from the messages exchanged. We define function $\omega(\vec{x})$ as a projection to a subset of variables of $\vec{x}$ with all the other variables and parameters removed, if they are related to timestamp, nonce and signature. From the OAuth specification, the predicate on $\omega(\vec{x})$ in M are independent of timestamp, nonce, signature and Token content, that is, independent of the variables other than those in $\omega(\vec{x})$. From Proposition 3, $\widehat{M}$ is a trace inclusion transformation from M.

D. Minimal Reachable Graph

We apply Algorithm 1 to $\widehat{M}$ to construct a minimal reachable graph G^* for both versions of OAuth. The minimal reachable graph of OAuth 1.0 consists of 67 states and OAuth 1.0A has 44 states. Note that the reachability graph $\hat{G}$ of $\widehat{M}$ is infinite. From Section III, four simultaneous runs, and hence a Cartesian product of 4 minimal reachable graphs G^* is constructed for an analysis. The number of reachable states in the two Cartesian products is 167,648 and 124,800 for OAuth version 1.0 and 1.0A, respectively.

E. Results

As summarized in Subsection VI.D, if any attack trace is found, we use Algorithm 2 to check if it is a false positive. If no attack trace is found, from Proposition 5, we are sure that there is no false negative and hence the protocol is secure with respect to that security property.

We now summarize the results from out automated security property verification.

Our experiment shows that both OAuth 1.0 and 1.0A satisfy Property P_1, P_2 and P_3. Consumer can access User's data only after Consumer has obtained an Access Token. Meanwhile, to obtain Access Token, Consumer must first obtain a Request Token and has the Request Token authorized by the User.

Our study also shows that both versions of OAuth satisfy Property P_4. Consumer accesses a User's data only after the User has authorized the corresponding Request Token. However, our investigation indicates that Property P_5 is violated in OAuth 1.0. That is, after User authorization of Consumer's access right, the User may not obtain the service provide from the accessed data; some other User may. It is exactly how a Session Fixation attack succeeded as reported in [32]. We identify a total of 25 different traces leading to this attack, including the one reported in [32]. For OAuth 1.0A, we also identify 1 attack trace from the Cartesian product of the minimal reachable graphs. Algorithm 2 rules it out as a false positive. Therefore, we conclude that OAuth 1.0A is secure in terms of the five security properties we investigate.

F. Insecure Interfaces

We now discuss briefly the case that the API interfaces are insecure with HTTP, for instance. There are API authentication and authorization protocols in the public domain which still use insecure API [31]. We comment with respect to the five security properties in Subsection *B*.

Our results show that neither OAuth 1.0 nor OAuth 1.0A satisfy security property P_4 anymore. Our attack traces show that when an attacker obtains information of a Request Token by eavesdropping or message interception he can authorize it on behalf of the User. Consequently, Consumer can access User data without the authorization by the User. Our attack traces also show that Property P_5 is violated, that is, Session Fixation attack is still possible for both OAuth versions. Specifically, if an attacker seizes the verifier and brings it to Consumer then he can still obtain the service authorized by the User. We also find attack traces on property P_2, but we rule them out as false positives.

Similarly, if an implementation of OAuth is stateless, then there are security attacks for both versions.

The current OAuth specification uses nonce, timestamps and signatures to guard against possible attacks. If the API interfaces are secure, they are not needed. On the other hand, if the API interfaces are insecure, they are not sufficient to guarantee the desired security properties. This observation might suggest a redesign of the protocol. It may also involve rather complicated analysis with insecure API interfaces.

VIII. Conclusion

In this paper, we present an approach for a formal and automated analysis of protocol security properties, which can be formulated as temporal properties. Our approach is rather general and could also be used for analyzing other protocol security properties.

Trace inclusion transformation and online minimization are powerful tools for state space reduction. For OAuth, they reduce the state space from infinity to a few dozens. These techniques could be useful for other applications, which require significant state space reduction.

One has to consider all possible interleaved protocol runs to model attacker behaviors so that the analysis is complete – no attack traces are missed. However, we want to keep it as few as possible since it contributes multiplicatively to the state

space expansion. For OAuth, four simultaneous executions are enough. In general, it is intriguing to find a minimal number of simultaneous runs yet with a guarantee of the completeness of the security property analysis.

ACKNOWLEDGMENT

We are deeply indebted to Eric Grosse for introducing the API authentication problem, particularly OAuth, to us and for the insightful and stimulating comments. We thank Breno de Medeiros for keeping us informed of the evolvement of OAuth and for the helpful discussions.

We appreciate very much the constructive and valuable comments by ICNP 2010 reviewers.

This work is partially supported by a Research Gift from Google and by a Research Grant from DoD FA9550-09-1-0280.

REFERENCES

[1] K. Bhargavan, C. Fournet, A.D. Gordon and N. Swamy, "Verified Implementations of the Information Card Federated Identity-Management Protocol," in *Proc. ASIAN ACM Symposium on Information, Computer and Communications Security (ASIACCS'08),* pp. 123-135, Mar. 2008.

[2] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in *IEEE Transaction on Computers*, vol. 35, no 8, pp. 677-691, 1986.

[3] J. R. Burch, E. M. Clarke, K. L. Mcmillan, D. L. Dill and L. J. Hwang, "Symbolic Model Checking: 10^{20} states and Beyond," in *Proc. 5th IEEE LICS*, pp. 428-439, 1990.

[4] M. Burrows, M. Abadi and R.M. Needham, "A Logic of Authentication," *ACM Trans. Computer Systems*, vol. 8, no. 1, Feb. 1990, pp. 18-36.

[5] F. Butler, I. Cervesato, A.D. Jaggard, A. Scedrov and C. Walstad, "Formal Analysis of Kerberos 5," *Theoretical Computer Science,* vol. 367, no. 1, 2006.

[6] I. Cervesato, A.D. Jaggard, A. Scedrov, J.-K. Tsay and C. Walstad, "Breaking and Fixing Public-Key Kerberos," in *Proc.6th Int'l Workshop on Issues in the Theory of Security (WITS '06),* 2006.

[7] O. Coudert, C. Berthet and J.C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution," in *Proc. Int'l Workshop on Automatic Verification Methods for Finite State Systems*, pp. 365-373, 1990.

[8] S. Crites, F. Hsu and H. Chen, "OMash: Enabling Secure Web Mashups via Object Abstractions," in *Proc. 15th ACM Conf. on Computer and Communications Security* (CCS 2008), Oct., 2008.

[9] D. Dolev and A.C. Yao, "On the Security of Public Key Protocols," *IEEE Trans. Information Theory*, vol. IT-30, no. 2, pp. 198-208, Mar. 1983.

[10] R. Hasan, M. Winslett, R. Conlan, B. Slesinsky and N. Ramani, "Please Permit Me: Stateless Delegated Authorization in Mashups," in *Proc. 24th Computer Security Applications Conference* (ACSAC 08), Dec. 2008 pp. 173-182.

[11] C. A. R. Hoare, *Communicating Sequential Process,* Eaglewood Cliffs, NJ: Prentice-Hall, 1985.

[12] J. Howell, C. Jackson, H. Wang and X. Fan, "MashupOS: Operating System Abstractions for Client Mashups," in *Proc. 11th USENIX Workshop on Hot Topics in Operating Systems*, 2007.

[13] M. L. Hui and G. Lowe, "Safe Simplification transformation for Security Protocols or Not Just Needham Schroeder Public Key Protocol," in *Proc 12th Computer Security Foundations Workshop*, pp. 32-43, 1999.

[14] P.C. Kanellakis and S.A. Smolka, "CCS Expressions, Finite State Processes, and Three Problems of Equivalence," in *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, pp. 228-240, 1983.

[15] R. Kemmerer, C. Meadows and J. Millen, "Three Systems for Cryptographic Protocol Analysis," in *Journal of Cryptology*, vol. 7, no. 2, 1994.

[16] F.D. Keukelaere, S. Bhola, M. Steiner, S. Chari and S. Yoshihama, "SMash: Secure Component Model for Cross-Domain Mashups on Unmodified Browsers," in *Proc. 17th Int'l Conf. on World Wide Web* (WWW 08), pp. 534-544, Apr. 2008.

[17] N. Kulathuramaiyer, "Mashups: Emerging application development paradigm for a digital journal," *Journal of Universal Computer Science*, vol. 13, no. 4, 2007.

[18] D. Lee and M. Yannakakis, "Online Minimization of Transition Systems," in *Proc. 24th Annu. ACM Symp. on Theory of Computing,* 1992, pp. 264-274.

[19] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines – A Survey," in *Proc. The IEEE*, vol. 84, no. 8, Aug. 1996.

[20] G. Lowe, "Breaking and Fixing the Needham-Schroeder Public Key Protocol using FDR," In *Proc. 2nd Int'l Workshop on Tools and Algorithms for Construction and Analysis of Systems*, LNCS 1055, Springer, pp. 147-166, 1996.

[21] G. Lowe, "Toward a Completeness Result for Model Checking of Security Protocols," in *Journal of Computer Science*, vol. 7, no. 2-3, pp. 89-146, 1999.

[22] J.C. Mitchell, M. Mitchell and U. Stern, "Automated Analysis of Cryptographic Protocols Using Murφ," in *Proc. IEEE Symposium on Security and Privacy,* 1997.

[23] J. C. Mitchell, V. Shmatikov and U. Stern, "Finite-State Analysis of SSL 3.0 and Related Protocols," in *Proc. 7th USENIX Security Symposium*, 1998.

[24] B.C. Neuman and S.G. Stubblebine, "A Note on the Use of Timestamps as Nonce," *ACM SIGOPS Operating Systems Review*, vol. 27, no. 2, pp. 10-14, 1993.

[25] A. W. Roscoe, "Proving Security Protocols with Model Checkers by Data Independence Techniques," in *Proc. 11th Computer Security Foundations Workshop*, pp. 84-95, 1998.

[26] P. Ryan and S. Schneider, *Modelling and Analysis of Security Protocols,* Addison-Wesley, 2001.

[27] J. Sifakis, "A Unified Approach for Studying the Properties of Transition Systems", *Theoretical Computer Science*, vol. 18, 1982.

[28] M. Yannakakis and D. Lee, "An Efficient Algorithm for Minimizing Real-Time Transition Systems," in *Proc. Formal Methods in System Design,* vol. 11, no. 2, pp. 113-136, 1997.

[29] AOL OpenAuth, http://dev.aol.com/api/openauth

[30] Flickr Authentication API, http://www.flickr.com/services/api/auth.spec.html

[31] Meetup, http://www.meetup.com/meetup_api/

[32] OAuth, http://oauth.net/

[33] OpenID, http://openid.net/

[34] OpenID OAuth Extension, http://step2.googlecode.com/svn/spec/openid_oauth_extension/latest/openid_oauth_extension.html

[35] Yahoo! BBAuth, http://developer.yahoo.com/auth/

[36] http://hueniverse.com/oauth/

[37] http://stub.bz/sslrebinding/