Detecting the Vulnerability of Multi-Party Authorization Protocols to Name Matching Attacks

Wenjie Lin (Contact Author)*, Guoxing Chen*, Ten H. Lai*, David Lee†
*Ohio State University, 2015 Neil Ave, Columbus, OH
†HP Labs, 1501 Page Mill Road, Palo Alto, CA
{linw, chenguo, lai}@cse.ohio-state.edu {david.lee10}@hp.com

***Abstract*—Software as a Service (SaaS) clouds cooperate to provide services, which often provoke multi-party authorization. The multi-party authorization suffers the so-called name matching attacks where involved parties misinterpret the other parties in the authorization, thus leading to undesired or even fatal consequences (*e.g.,* an adversary can shop for free or can log into a victim's Facebook account).**

In this paper, we propose a scheme to detect the vulnerability of multi-party authorization protocols that are susceptible to name matching attacks. We implement the detecting scheme and apply it to real world multi-party authorization protocols including Alipay PeerPay, Amazon FPS Marketplace, and PayPal Express Checkout. New name matching attacks are found, and fixes are proposed accordingly.

Keywords: *multi-party authorization, name matching attacks, protocol analysis.*

Tracks: *Security Applications, Information Assurance*

I. Introduction

There are a large number of Software as a Service (SaaS) clouds in the market, such as Google Docs (document management), Dropbox (storage and sharing), HP ePrint (printing), Salesforce (customer relation management), Cloud9 (revenue forecast), 1010Data (Big Data analysis), and Alipay (third-party payment). When these clouds cooperate with one another, more advanced services can be provided, and they often require multi-party authorization. For example, a user Alice may authorize HP ePrint to retrieve and print her bank statements stored in her Google Docs—this service would involve three-party authorization. There are services requiring four-party authorization. For example, Alice may order some products from Newegg and ask her husband Bob to pay the bill through Alipay (which is a popular PayPal-like service provider in Asia). Here, Alice and Bob together authorize Newegg to withdraw some money from Bob's Alipay account.

Cloud services requiring five- or more-party authorization are not unthinkable. Envision a new cloud service called SyncData that allows multiple users to synchronize their data stored at various clouds. For example, Alice and Bob may want SyncData to update Alice's dataset in 1010Data according to Bob's sales data in Saleforce. This requires five-party authorization. As another example, suppose it takes several managers' approvals to allow Cloud9 to retrieve a company's sales data in Salesforce, thus triggering an n-party authorization.

Cisco recently announced plans to build the world's largest global Intercloud—a network of clouds—in the next two years. As data communications and job migrations among clouds become more efficient and secure, we believe that new services involving multiple clouds and thus requiring multi-party authorizations will be emerging.

In this paper, we investigate multi-party authorization protocols in cloud services, focusing on detecting the vulnerability to the so-called *name matching attacks*, which we believe are main threats to the security of multi-party authorization. Our study is motivated by the following reasons.

First of all, name matching attacks are new threats that typically exist in cloud-based authorization but not in traditional authorization such as RBAC [22], PBAC [20] and ABAC [28]. In cloud applications, a user may use different usernames at various clouds. It is not trivial (if not impossible) to figure out whether the username say `aaa` at one cloud and the username say `xyz` at another cloud refer to the same user (especially when privacy is a concern). Thus, when an initiator issues an authorization request with n parties' names in it, it is important, but nontrivial, for each named party to know who exactly the other parties are — which is the root cause of the name matching attacks.

Secondly, name matching attacks lead to undesired or even fatal consequences to multi-party authorization. In the aforementioned example of HP ePrint, Alice (with username `xyz` at Google) grants the authorization at Google Docs. If an adversary was able to launch an attack, such that HP ePrint failed to match Alice's name—thought it was `bbb` (Bob's HP username) who granted the authorization—HP ePrint would print Alice's bank statements on Bob's printer (Figure 1). Alice's personal data would be released. We summarize name matching attacks against three-party authorization in the literature in Table I.

Finally, to the best of our knowledge, although name matching attacks have been vaguely recognized in the literature as violations of "a series bindings" [24] and "association" [26], detecting the vulnerability of multi-party authorization protocols to the name matching attacks has never been singled out as a problem. The existing research focused on three-party scenarios and was in case-by-case fashion. For example, researchers identified attacks [3], [4] against the famous three-party authorization protocol OAuth [12], [13] and its applications to Single Sign On [25], [23], [7], [6]. Various attacks were also identified in PayPal and in Amazon payment services [24]. As far as we know, more-than-three party authorization and its security have not been studied.

This paper has two contributions. First, we propose a

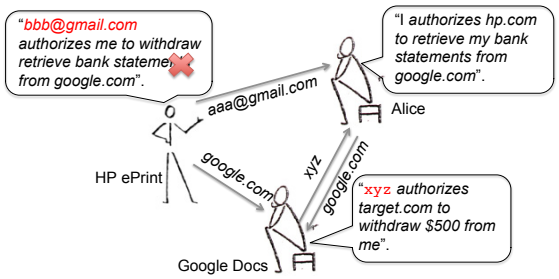


Fig. 1: An example of name matching attack

TABLE I: Name matching attacks in literature

Pub date	Affected protocols	Reference
April, 2009	OAuth 1.0	Session fixation attack [3]
Aug, 2011	OAuth v2-16	Auth code swap attack [4]
May, 2011	NopCommerce integrating Amazon Simple Pay	[24]
May, 2011	Amazon Payment SDK	[24]
May, 2012	Google ID & Smartsheet SSO	[25]
Oct, 2012	One third of studied RPs using OAuth in [23]	Session swapping attack [23]

scheme to detect the vulnerability of multi-party authorization protocols to name matching attacks. Our detecting scheme is inspired by the observation that all current multi-party authorization protocols are composed of five types of three-party primitives. Therefore, by checking the security of each primitive and their composition with Protocol Composition Logic (PCL) [11], [21], we are able to detect the vulnerability. Comparing to other formal method approaches (*e.g.,* [14], [19]), our detecting scheme takes more protocol insights into consideration, and thus can pinpoint the vulnerability in a protocol and suggest a remedy immediately.

Moreover, we implement the detecting scheme and apply it to high-profile three-party and four-party authorization protocols, including Alipay PeerPay, Amazon FPS Marketplace, and PayPal Express Checkout. New vulnerabilities to name matching attacks are found and the remedies are proposed.

The rest of the paper is organized as follows. In Section II, we introduce the system model and the adversary model. We propose the detecting scheme in Section III. The scheme is applied to case study in Section IV.

II. System and Adversary Models

Entities and names: There is a universal set $\mathcal{E}$ of *entities*, which in practice consists of all cloud service providers (*e.g.,* Google, Amazon), end users (*e.g.,* Alice, Bob), enterprise users, traditional service providers (*e.g.,* banks, Visa) and so on. An entity knows another entity by a single *name* such as a user's username or a cloud's URL. (Each entity typically knows only a subset of entities in $\mathcal{E}$.) For simplicity, we assume that a user may have at most one account/username at a cloud. (If Alice has multiple accounts/usernames at Google, we will have to treat the "person" Alice as multiple entities, each corresponding to one of her Google username.) Denote the set of all entities' names as $\mathcal{N}$. The acquaintances between entities (*i.e.,* who knows who by what name) are modeled as a function Ent, as described below.

Definition 1: **[Function Ent]** $\texttt{Ent} : \mathcal{E} \times \mathcal{N} \rightarrow \mathcal{E} \cup \{\perp\}$. For each entity $e \in \mathcal{E}$ and name $m \in \mathcal{N}$, let $\texttt{Ent}(e, m)$ denote the entity that is known to e by the name m. If e doesn't know anybody by the name m, then $\texttt{Ent}(e, m) = \perp$.

Example: If Alice's username at Google is `xyz`, then $\texttt{Ent}(\text{Google}, \texttt{xyz}) = \text{Alice}$.

Multi-party authorization: In an n-party authorization, an initiator sends out an authorization request (typically to one of the entities involved in the request), thereby triggering n entities to exchange messages and reach an authorization decision respectively (e.g., the decisions made by HP ePrint, Google Docs and Alice in the aforementioned example in Figure 1).

The request specifies names of n entities—authorizers, authorizees, and enforcers—in the authorization. An authorizer (*e.g.,* Alice) is an entity who grants the authorization; An authorizee (*e.g.,* HP ePrint) is an entity who is granted the authorization; An enforcer (*e.g.,* Google Docs) is an entity (often a server) who manages resources and enforces the authorization.

Reliable and secure channels: The channel is reliable without message loss (*e.g.,* a TCP channel), and is secure against eavesdroppers and interception (*e.g.,* an SSL channel). However, we do not assume all messages can be authenticated, because it is unrealistic to assume all ordinary users (*e.g.*, Alice and Bob) have a public key known by the rest parties in the authorization.

Bounded message delay and computation time: We assume that there are bounds on message delay and computation time. A party aborts the protocol if it does not receive a message or obtain a computation result in time.

Concurrent self composition: As there are concurrent multi-party authorizations, an authorization protocol should be secure under concurrent self composition, that is, the protocol remains secure even when it is executed concurrently multiple times [17].

The adversary model: We consider the Byzantine adversary model [15] where an adversary can exhibit arbitrarily malicious behaviors. Moreover, an adversary can launch web attacks described in [23]: It can send malicious links via spam or by posting an Ad on a malicious website. If a victim clicks on the malicious link, the (browser of the) victim will send HTTP requests (*i.e.*, GET and POST methods) with the messages crafted by the adversary.

III. Detecting the Vulnerabilities to Name Matching Attacks

In this section, we introduce a scheme that can detect vulnerabilities to name matching attacks in a class of multi-party authorization protocols, the protocols that are composed by three-party primitives (described next).

A. Overview of our detecting scheme

To the best of our knowledge, the multi-party authorization protocols we are aware of—which are provided by Google,

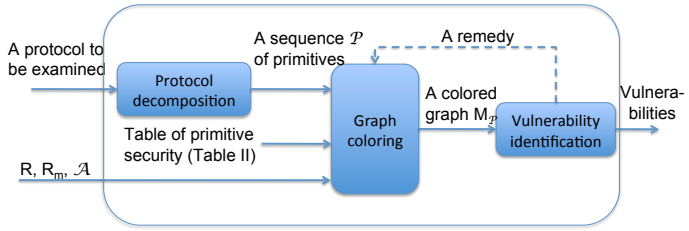


Fig. 2: A scheme to detect vulnerabilities to name matching attacks

Amazon, Alipay, Facebook, and other high-profile vendors—are all composed of three-party primitives (described in Section III-B1). We thus focus on detecting vulnerabilities in this kind of multi-party authorization protocols. The detecting scheme includes three modules (Figure 2): protocol decomposition, graph coloring, and vulnerability identification.

In the protocol decomposition module, a given n-party authorization protocol is re-written as a sequence of three-party primitives. We describe the primitives and their security in Section III-B1 and Section III-B2. In this step, manual assistance is needed.

In the graph coloring module, the sequence of primitives together with other inputs are fed into Algorithm 1, which thereby outputs a colored graph. If the colored graph has a red or grey edge, a name matching vulnerability is likely to exist.

In the vulnerability identification module, one cab identify the vulnerability by investigating how the graph is colored step by step in the graph coloring module. This investigation reveals the root cause of the vulnerability, so that one can construct a name matching attack accordingly. With each vulnerability, one can propose a remedy and verify it by repeating the last two modules. Here manual assistance is needed.

B. Protocol decomposition

All multi-party authorization protocols that we are aware of can be composed by the following five types of three-party primitives.

1) **Five types of three-party primitives:** A primitive (e, c, d) is a protocol involving three (not necessarily distinct) entities e, c, d. The input to the protocol is an entity e and two names m_c and m_d by which e knows entities c and d, respectively. (Note that c and d are not part of the input; only their names known by e are.) At the end of the protocol, entity c outputs a name m'_d by which it knows entity d. That is, c outputs a name m'_d such that $\texttt{Ent}(e, m_d) = \texttt{Ent}(c, m'_d)$. (We could have written the primitive as (e, m_c, m_d), but chose to write it as (e, c, d) for ease of understanding.)

In the following, e, c, d indicate distinct entities. A session ID is used to distinguish between different sessions or executions of the same primitive.

1) $P_1 = (e, c, c)$: Suppose e knows c by a name m_c. The protocol is trivial; just let e send c a message containing the name m_c and session ID sid. After the primitive (protocol), c outputs the name it calls itself in session sid:
$e \to c : \{m_c, sid\}$;
c outputs its own name in session sid.

2) $P_2 = (e, c, e)$: Suppose c calls e by a name m_e. The protocol is straightforward — e sends c a message containing the name m_e and session ID sid. After the primitive, c outputs m_e in session sid:
$e \to c : \{m_e, sid\}$;
c outputs m_e in session sid.

3) $P_3 = (e, c, d)$ where e, c know d by the same name m_d. After the primitive, c outputs m_d in session sid:
$e \to c : \{m_d, sid\}$;
c outputs m_d in session sid.

4) $P_4 = (e, c, d)$ where e, d know c by the same name m_c. The protocol consists of two steps. In the first step, e sends a message to d (who is known to e by the name m_d): "Hi m_d, please ask m_c to output your name." In the second step, d sends a message to c: "Hi m_c, my name is m'_d." After the primitive, c outputs m'_d in session sid (m'_d is the name by which c knows d):
$e \to d : \{m_c, m_d, sid\}$;
$d \to c : \{m'_d, sid\}$;
c outputs m'_d in session sid.

5) $P_5 = (e, c, d)$ where e, d know c by the same name m_c, and c, d knows e by the same name m_e. The protocol includes four steps. The first step is similar to the one in P_4, except that e generates a secret sec and sends it to d. In the second step, d sends a message to c: "Hi m_c, my name is m'_d. The secret is sec. Please confirm it with m_e." In the third and last steps, c sends sec back to e, who verifies if it has sent the sec in the first step. After the primitive, c outputs m'_d in session sid (m'_d is the name by which c knows d):
$e \to d : \{m_c, m_d, sec, sid\}$;
$d \to c : \{m_e, m'_d, sec, sid\}$;
$c \to e : \{sec, sid\}$;
$e \to c : \{\text{correct}, sid\}$;
c outputs m'_d in session sid.

2) Security of individual primitives: We examine if each primitive satisfies the security property that all honest entities output correct names, that is, if c is honest and outputs a name m'_d, there must be an entity e who inputs two names m_c, m_d such that $\texttt{Ent}(e, m_c) = c$ and $\texttt{Ent}(e, m_d) = \texttt{Ent}(c, m'_d)$.

The security of individual primitives is summarized in Table II. In the table, 0 indicates that a name matching attack can be found and 1 indicates that the security property holds (if certain condition is satisfied). Here $\mathcal{A}$ is the authentication attributes. For example, $\mathcal{A}(c, e) = 1$ means that c can authenticate the messages sent by e. The proof is based on Protocol Composition Logic (PCL) [11], [21]. Due to space limit, please refer to our full paper [2] for details.

C. Graph coloring

The GRAPH_COLORING algorithm (Algorithm 1) takes five inputs: (1) a sequence of primitives $\mathcal{P}$; (2) the set of entities $R = \{au_1, \ldots, au_i, az_1, \ldots, az_j, ef_1, \ldots, ef_k\}$ in an authorization, where each au_i is an authorizer, each az_j is an authorizee, and each ef_k is an enforcer; (3) the set of entities R_m ($R_m \subset R$) who may be malicious; (4) the authentication

TABLE II: Security of individual primitives

Primitive	**All are honest**	e **may be malicious**	d **may be malicious**
$P_1 = (e, c, c)$	1 *if* $\mathcal{A}(c, e) = 1$	0	–
$P_2 = (e, c, e)$	1 *if* $\mathcal{A}(c, e) = 1$	0	–
$P_3 = (e, c, d)$, where e, c know d by the same name.	1 *if* $\mathcal{A}(c, e) = 1$	0	1 *if* $\mathcal{A}(c, e) = 1$
$P_4 = (e, c, d)$, where e, d know c by the same name.	1 *if* $\mathcal{A}(d, e) = \mathcal{A}(c, d) = 1$	0	0
$P_5 = (e, c, d)$, where e, d know c by the same name, c, d know e by the same name.	1 *if* $\mathcal{A}(d, e) = \mathcal{A}(c, d) = 1$ and $\mathcal{A}(e, c) = \mathcal{A}(c, e) = 1$	0	1 *if* $\mathcal{A}(d, e) = \mathcal{A}(c, d) = 1$ and $\mathcal{A}(e, c) = \mathcal{A}(c, e) = 1$

attributes $\mathcal{A}$, where $\mathcal{A}(c, e) = 1$ iff entity c can authenticate the messages sent by entity e; and (5) Table II (T) that determines the security of individual primitive.

The GRAPH_COLORING algorithm outputs a three-color (green, red, gray) graph. The nodes are the entities in the authorization. A green edge (c, d) where $c, d \in R$ indicates that c can correctly figure out the name of d; A red edge (c, d) implies that c may *incorrectly* match the name of d—an alarm of name matching attacks; A gray edge (c, d) means that c may not be able to find out the name of d.

Here we briefly illustrate how the algorithm work: GRAPH_COLORING algorithm checks the sequence of primitives one by one (Line 8–16). If a primitive is simply in one of the five types (e, c, d), the color of the edge (c, d) is determined by the function CHECK_GREEN (Line 9–13). If c continues executing the protocol when the name outputted in the primitive is the same with the name c has known (denoted as $\overline{(e, c, d)}$), the edge (e, d) is colored according to Line 14–16.

The correctness of the algorithm is justified by PCL too. Due to space limit, please refer to our full paper [2] for details.

Algorithm 1 GRAPH_COLORING

Input:
1: Set of entities: $R = \{au_1, \ldots, au_i, az_1, \ldots, az_j, ef_1, \ldots, ef_k\}$;
2: Set of entities that may be malicious $R_m \subset R$;
3: Sequence of primitives: $\mathcal{P} = p_1 \ldots p_t$ where each $p_i = (e, c, d)$ has $e, c, d \in R$;
4: Authentication attributes $\mathcal{A}$
5: Security of primitive T: $\mathcal{P} \times \mathcal{A} \times R_m \rightarrow \{0, 1\}$ (Table II)
Output: Colored graph $M_{\mathcal{P}}$
6: Color all edges gray in the complete graph $M_{\mathcal{P}}$
7: Color all edges directed from the initiator to other nodes green
8: **for** $i = 1$ to t **do**
9: **if** $p_i = (e, c, d)$ **then**
10: **if** CHECK_GREEN(p_i, $\mathcal{A}$, R_m) **then**
11: Color the edge (c, d) green
12: **else**
13: Color the edge (c, d) red
14: **else** ▷ $p_i = \overline{(e, c, d)}$ is for verification
15: **if** (e, d) is red && (c, d) is green && $T(p_i, \mathcal{A}, R_m)$ && $c \in R - R_m$ **then**
16: Color the edge (e, d) green
17: **return** $M_{\mathcal{P}}$;

1: **function** CHECK_GREEN((e, c, d), $\mathcal{A}$, R_m)
2: **if** $T((e, c, d), \mathcal{A}, R_m) = 0$ **then**
3: **return** false
4: **if** $d = e$ && (e, c) is green **then return** true
5: **else if** $d \neq e$&& (e, d), (e, c) are green **then return** true
6: **else return** false
7: **end function**

D. Vulnerability identification

A red or grey edge in the colored graph rings an alarm of name matching attacks. To detect the vulnerability and construct possible attacks, one should examine the step-by-step coloring in Algorithm 1.

We summarize common vulnerabilities as follows.

1) **Lack of authentication:** For example, in the Single Sign On service provided by Gigya, 13% of studied websites do not authenticate the messages sent by Gigya [23]. Due to the lack of authentication, the edge from Gigya to the authorizer is colored red. Name matching attacks identified in [23] can thus be launched.
2) **Misused primitives:** For example, PayPal uses primitive P_4 instead of P_5. However, $P_4 = (e, c, d)$ is insecure when entity d is malicious. This causes a name matching attack (Section IV-C).
3) **Missing primitives:** For example, in Alipay protocol a primitive P_5 is missing, which produces two red edges and thus a name matching attack (Section IV-A).

The detecting scheme may have false positives. It is possible a vulnerability is found, but an adversary cannot exploit it due to application constrains. In this case, manual investigation is needed to rule out false alarms.

IV. CASE STUDY

We apply the detecting scheme to real world multi-party applications, including Alipay PeerPay, Amazon FPS Marketplace and PayPal Express Checkout. New vulnerabilities and name matching attacks are found. Remedies are proposed accordingly.

Due to space limit, we will introduce the case study on Alipay PeerPay in details and briefly report our findings in Amazon FPS Marketplace and PayPal Express Checkout.

A. Alipay PeerPay (four parties)

In this section, we apply the detecting scheme to Alipay PeerPay and find a vulnerability susceptible to a name matching attack.

Alipay is a popular PayPal-like payment service provider in Asia, which has 300 millions of registered users. The PeerPay service provided by Alipay enables one to shop online and let someone else to pay the bill.

For example, with PeerPay service Alice can order a $500 iPad on Yihaodian (an online shopping website owned by Walmart with URL `yihaodian.com`) and let her husband Bob pay for her. In this example, a four-party authorization is needed: Alice and Bob together authorize Yihaodian to withdraw $500 from Bob's Alipay account. Here Alice and Bob are two authorizers; Yihaodian is the authorizee, and Alipay is the enforcer.

1) **Normal workflow of Alipay PeerPay:** In the above example, Alice first places an order of iPad at `yihaodian.com` and selects the payment method as "alipay". Once Alice clicks on "checkout", she is redirected to `alipay.com` where Alice logs in her account (say with username `aaa@gmail.com`). Upon a successful login, Alipay asks Alice to verify the request: "paying `yihaodian.com` $500 for an iPad." If the request is correct, Alice specifies her husband Bob to pay the bill by providing his Alipay username say `bbb@gmail.com` to Alipay. Alipay will thereby send an email to `bbb@gmail.com` (Bob) as a notification. Once Bob logs in `alipay.com`, he will review the request "`aaa@gmail.com` is requesting you to pay $500 to `yihaodian.com`. Do you agree?" If Bob confirms, Alipay will notify Yihaodian of the successful authorization.

2) **The detected name matching attack:** We applied the detecting scheme to the Alipay PeerPay protocol and identified a new name matching attack, with which an adversary can shop online for free. Note that this attack is so easy to launch, that the adversary does not need to have a priori knowledge on computer science. The attack scenario is as follows.

Step 1: An adversary (say Eve) posts an advertisement: Eve posts a malicious advertisement on her Weibo page (a Twitter-like social network), claiming that she can order an iPad for buyers at `yihaodian.com` with %5 cash back. Suppose Eve's Yihaodian username is `eee@gmail.com`.

Step 2: Alice places an order: If Alice is attracted by the advertisement and places an order for an iPad at Eve's Weibo page, she will be redirected to `alipay.com`, where Alice logs in with her Alipay username say `aaa@gmail.com`.

Step 3: Alice verifies the request and specifies Bob to pay the bill: At `alipay.com`, Alice is asked by Alipay to verify the request "paying $500 to `yihaodian.com` for an iPad"—this is exactly what Alice wants to do. Note that in the request, the money is paid to Yihaodian instead of Eve. Alice then specifies Bob to pay the bill by providing Bob's Alipay username say `bbb@gmail.com`.

Step 4: Bob verifies the request and grants the authorization: Alipay sends an email to `bbb@gmail.com`, indicating that a request is waiting for approval. Bob then logs in to his Alipay account where he verifies the request "`aaa@gmail.com` is requesting you to pay $500 to `yihaodian.com`. Do you agree?" As this is what Bob wants to do and he trusts Alipay, Bob will grant the authorization.

Step 5: Eve shops for free: Once Bob grants the authorization, Alipay allows Yihaodian to withdraw $500. However Alipay (as well as Alice and Bob) and Yihaodian do not match the names of the first authorizer correctly—Alipay thinks that the first authorizer is `aaa@gmail.com` (Alice), while Yihaodian thinks that the first authorizer is `eee@gmail.com` (Eve). As a result, Alipay will charge the bill to Bob who is specified by Alice, while Yihaodian will ship the iPad to Eve.

Remark 1: This attack is different from the traditional phishing attack, because Eve does not masquerade as `yihaodian.com`.

Remark 2: The reader may argue that this is not an attack, because Alice and Bob can ask for a refund from Alipay. However, because the refund is charged on Yihaodian, Yihaodian now becomes the victim—it follows the Alipay PeerPay protocol exactly, but suffers the financial loss.

Remark 2: The reader may argue that, as Alice places an order at Eve, she deserves the attack. It is not true in this scenario, because Alice does not authorize the adversary Eve to withdraw the money from Alipay. Instead, she authorizes Yihaodian to withdraw the money. However, Yihaodian retrieves the money from her husband Bob, but ships the iPad to Eve.

3) **Techniques to launch the attack:** Eve can easily take four steps to launch the name matching attack without a priori knowledge on computer science. First of all, Eve logs in to her Yihaodian account using her web browser and places an order for an iPad. In the second step, Eve turns on her firewall (*e.g.,* Little Snitch) and blocks any request that is sent to `alipay.com`. In the third step, Eve chooses "alipay" as the payment method at `yihaodian.com` and clicks on "checkout". Because of the firewall, her browser cannot send any request to `alipay.com`. In the final step, Eve records the blocked request (an URL) to `alipay` (Figure 3a) and posts the URL in her advertisement page at Weibo. When a victim (Alice) clicks on "checkout" at Eve's Weibo page, Alice's browser follows the URL and sends the recorded request to `alipay.com`.

4) **Applying the detecting scheme:** Our detecting scheme identifies the above name matching attack as follows.

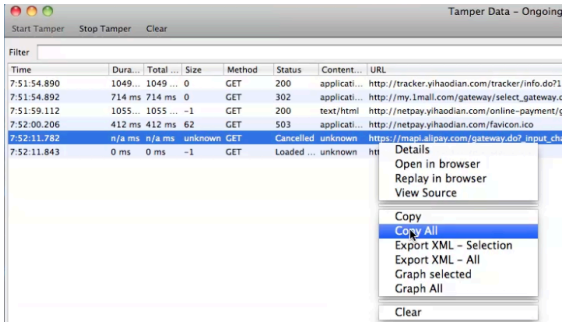
Protocol decomposition: According Alipay official document [1], there are four entities in the authorization: the first authorizer au_1 (*e.g.,* Alice), the second authorizer au_2 (*e.g.,* Bob), the authorizee (*e.g.,* Yihaodian), and the enforcer (Alipay). Here au_1 and au_2 may be malicious. Note that $\mathcal{A}(au_1, az) = 0$ because the first authorizer does not authenticate the messages sent by the authorizee.

The Alipay PeerPay protocol can thus be decomposed to the sequence of primitives: $p_1 = (az, ef, au_1)$, $p_2 = (az, ef, ef)$, $p_3 = (az, ef, az)$, $p_4 = (ef, au_1, au_1)$, $p_5 = (ef, au_1, ef)$, $p_6 = (ef, au_1, az)$, $p_7 = (au_1, ef, au_2)$, $p_8 = (ef, au_2, au_1)$, $p_9 = (ef, \underline{au_2, au_2})$, $p_{10} = \underline{(ef, au_2, ef)}$, $p_{11} = (ef, au_2, az)$, $p_{12} = \overline{(ef, az, ef)}$, $p_{13} = \overline{(ef, az, az)}$.

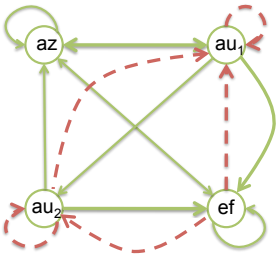
Graph coloring: Figure 3b shows the output of Algorithm 1. The red edges indicate that authorizers and the enforcer may match the names of the first authorizer incorrectly. A vulnerability to name matching attacks may exist.

Vulnerability identification: According to the process of Algorithm 1, the red edges are caused by $\mathcal{A}(au_1, az) = 0$ and malicious au_1. The edges do not turn green when GRAPH_COLORING algorithm terminates. Therefore, a primitive that can verify the authorizers' names and turn the edges green is missing.

5) **A remedy:** To fix the vulnerability, intuitively, Alipay should verify with Yihaodian that Alice is really the first authorizer, before Alipay shows the request to her. More specifically, the protocol should add the type P_5 primitive (ef, az, au_1) right after p_3, which will turn all red edges to green. We have reported the attack as well as the remedy to Alipay.



(a) Eve recording the blocked request to `alipay.com`



(b) The colored graph

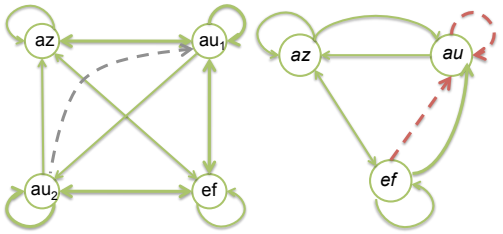
Fig. 3: Alipay PeerPay

B. Amazon FPS Marketplace (four parties)

Amazon FPS Marketplace service enables one to setup his own marketplace (say `payloadz.com`) where buyers and sellers can trade products (*e.g.*, files, eBooks, music) and pay through Amazon. This service requires a four-party authorization: a seller (the first authorizer au_1) and a buyer (the second authorizer au_2) together authorize Payloadz (authorizee az) to transfer a certain amount of money from the buyer's Amazon account to the seller's Amazon account. (Here Amazon is the enforcer ef.) Both authorizers may be malicious.

The detected name matching attack (minor): We found that an adversary can pretend to be a well-reputed seller at *payloadz.com* and sell fair quality products. Suppose Alice is attracted by the adversary's advertisement and places an order for a $50 file from the adversary who claims he is the well-reputed seller say `bbb` at `payloadz.com`. When Alice is directed to Amazon and logs in there, Amazon will show Alice a request, saying "pay `bbb` via `payloadz.com`" with "total amount: $50". However, `bbb` at `amazon.com` and the well-reputed seller `bbb` at `payloadz.com` may be two different entities. As a result, Alice actually pays the money to the adversary and buys the file with poor quality from him.

This name matching attack exploits the vulnerability that the edge from the second authorizer (*e.g.,* Alice) to the first authorizer (*e.g.,* the seller) is grey (Figure 4a), which indicates that Alice cannot know which seller at `payloadz.com` she is paying the money to. This can be easily fixed by the first authorizer verifying the name of the second authorizer at the authorizee.



(a) Amazon FPS Market-place (b) PayPal Express Checkout

Fig. 4: Colored graphs

C. PayPal Express Checkout (three Parties)

The detecting scheme can also be used in three-party authorization protocols, such as PayPal Express Checkout.

1) The detected name matching attack: We are able to identify a new name matching attack, which enables an adversary to shop on `target.com` for free. The attack is similar to the one against Alipay PeerPay.

The adversary first creates an advertisement at Facebook or Twitter, promoting a special offer to get cash back if one orders Target's merchandise through him. Suppose Alice is attracted by the advertisement and places an order for a $50 Target e-gift card through the adversary. When she chooses to pay by PayPal, Alice thinks that she is authorizing `target.com` to withdraw $50 dollars from her PayPal account.

Once Alice logs in her PayPal account, she will be asked to verify the request: paying Target (TARGETCORPO) $50 dollars for a "Seasons Greetings Snowglobal Gift Card"—exactly what Alice wants. Because the page looks exactly the same as if she places the order by herself, and more importantly because Alice trusts PayPal to protect the order, she is very likely to click on "continue". Afterwards, the adversary can construct a valid request to `target.com` and completes the payment.

As a result, PayPal will charge the bill to Alice while Target will ship the gift card to the adversary. This name matching attack exploits a vulnerability in PayPal Express Checkout protocol, indicated by the red edges in Figure 4b.

2) **A remedy:** The vulnerability is caused by a misuse of three-party primitive. To fix it, the P_4 type of primitive should be replaced by the P_5 type of primitive. For more details, please refer to our full paper [2]. The attack and the remedy have been reported to PayPal.

V. Related work

Authorization has been studied for decades. The most traditional ones include discretionary access control (DAC) [16], mandatory access control (MAC) [8], role based access control (RBAC) [22], which are widely used in homogeneous environment. In Internet environment (which is no longer homogeneous), authorization is closely related to Single-Sign-On (SSO) authentication. When a user requests services from a website, he is first authenticated by a third-party identity provider, where the user receives tickets (Kerberos [18]), cookies (Microsoft .Net Passport [10]) or encoded URLs (Liberty Alliance Project [5]). The user sends them to the website, who then provides services accordingly. In these authorizations, name matching is typically not an issue.

Security of three-party authorization in cloud environment has drawn great attention of researchers. Researchers found a session fixation attack [3] and an authorization code swap attack [4] against OAuth [13], whose security was further analyzed formally with Communicating Extended Finite State Machines [14], pi-calculus [7], Alloy [19], and Universal composability Security Framework [9]. Since 2011, researchers has inspected OAuth's application on Single Sign On (SSO) and revealed attacks [25], [23], [7]. Besides OAuth, online payment, another three party application, was also studied [24]. Learned from the field study of three-party authorization, researchers built systems to automatically extracting specifications from implementations (AUTHSCAN [6]), to offers authorizees the security protection to vulnerable web API integrations (InteGuard [27]), and to uncovering implicit assumptions of SDK provided by enforcers [26]. To the best of our knowledge, security of four-or-more party authorization has not been systematically studied yet.

VI. Conclusion

In this paper, we proposed a scheme to detect the vulnerability of multi-party authorization protocols to name matching attacks. By applying the scheme, we found new name matching attacks in high-profile three- and four-party authorization protocols such as Alipay PeerPay, Amazon FPS Marketplace, and PayPal Express Checkout.

In our future work, we will investigate name matching attacks when multiple colluded adversaries are present. We are also seeking a secure protocol that can solve the multi-party authorization problem.

References

[1] Alipay Peerpay. http://home.alipay.com/bank/paymentPayOther.htm.

[2] Full paper. https://www.dropbox.com/s/hf49zxs3gzknkcy/full.pdf.

[3] OAuth Security Advisory: 2009.1. http://oauth.net/advisories/2009-1/.

[4] [OAUTH-WG] Auth Code Swap Attack. http://www.ietf.org/mail-archive/web/oauth/current/msg07233.html.

[5] L. Alliance. Liberty alliance project. *Web page at http://www. projectliberty. org*, 2002.

[6] G. Bai, J. Lei, et al. AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations. *Proceedings of 20th Annual Network & Distributed System Security Symposium*, 2013.

[7] C. Bansal et al. Discovering Concrete Attacks on Website Authorization by Formal Analysis. In *Computer Security Foundations Symposium (CSF)*, pages 247–262. IEEE, 2012.

[8] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

[9] S. Chari, C. Jutla, and A. Roy. Universally Composable Security Analysis of OAuth v2.0. Technical report, 0. Cryptology ePrint Archive, Report 2011/526. http://eprint.iacr.org, 2011.

[10] M. Corporations. Microsoft. net passport review guide. Technical report, Technical report, Available at www. microsoft. com, 2003.

[11] A. Datta et al. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, 2007.

[12] E. Hammer-Lahav. RFC 5849: The OAuth 1.0 protocol. *Internet Engineering Task Force (IETF)*, 2010.

[13] D. Hardt. Rfc 6749: The oauth 2.0 authorization framework. *Internet Engineering Task Force (IETF)*, 2012.

[14] Y. Hsu and D. Lee. Authentication and Authorization Protocol Security Property Analysis with Trace Inclusion Transformation and Online Minimization. In *IEEE International Conference on Network Protocols (ICNP)*, pages 164–173. IEEE, 2010.

[15] L. Lamport et al. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[16] B. W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[17] Y. Lindell. Lower Bounds for Concurrent Self Composition. In *Theory of Cryptography*, pages 203–222. Springer, 2004.

[18] J. Lopez et al. Authentication and authorization infrastructures (AAIs): a comparative survey. *Computers & Security*, 23(7):578–590, 2004.

[19] S. Pai, Y. Sharma, S. Kumar, R. Pai, and S. Singh. Formal verification of oauth 2.0 using alloy framework. In *Proc. of Communication Systems and Network Technologies (CSNT)*, pages 655–659. IEEE, 2011.

[20] L. Pearlman et al. A Community Authorization Service for Group Collaboration. In *Proc. of Policies for Distributed Systems and Networks*, pages 50–59. IEEE, 2002.

[21] A. Roy et al. Secrecy Analysis in Protocol Composition Logic. In *Advances in Computer Science-ASIAN 2006. Secure Software and Related Issues*, pages 197–213. Springer, 2007.

[22] C. E. Sandhu, R.S. et al. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.

[23] S.-T. Sun and K. Beznosov. The Devil is in the (Implementation) Details: an Empirical Analysis of Oauth SSO Systems. In *Proceedings of ACM conference on Computer and Communications Security (CCS)*, pages 378–390. ACM, 2012.

[24] R. Wang et al. How to Shop for Free Online—Security Analysis of Cashier-as-a-Service based Web Stores. In *Security and Privacy (SP), IEEE Symposium on*, pages 465–480, 2011.

[25] R. Wang et al. Signing Me onto Your Accounts through Facebook and Google: a Traffic-Guided Security Study of Commercially Deployed Single-Sign-on Web Services. In *Security and Privacy (SP), IEEE Symposium on*, pages 365–379, 2012.

[26] R. Wang, Y. Zhou, et al. Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In *Proceedings of the USENIX Security Symposium*. USENIX, 2013.

[27] L. Xing, Y. Chen, et al. InteGuard: Toward Automatic Protection of Third-Party Web Service Integrations. In *Proceedings of 20th Annual Network & Distributed System Security Symposium*, 2013.

[28] E. Yuan and J. Tong. Attributed based Access Control (ABAC) for Web Services. In *Web Services, Proceedings. IEEE International Conference on*. IEEE, 2005.