Security issues in OAuth 2.0 SSO implementations

Wanpeng Li and Chris J Mitchell

Information Security Group, Royal Holloway, University of London
Wanpeng.Li.2013@live.rhul.ac.uk, C.Mitchell@rhul.ac.uk

Abstract. Many Chinese websites (relying parties) use OAuth 2.0 as the basis of a single sign-on service to ease password management for users. Many sites support five or more different OAuth 2.0 identity providers, giving users choice in their trust point. However, although OAuth 2.0 has been widely implemented (particularly in China), little attention has been paid to security in practice. In this paper we report on a detailed study of OAuth 2.0 implementation security for ten major identity providers and 60 relying parties, all based in China. This study reveals two critical vulnerabilities present in many implementations, both allowing an attacker to control a victim user's accounts at a relying party without knowing the user's account name or password. We provide simple, practical recommendations for identity providers and relying parties to enable them to mitigate these vulnerabilities. The vulnerabilities have been reported to the parties concerned.

1 Introduction

Since OAuth 2.0 was published in 2012 [1], it has been used by many websites worldwide to provide single sign-on (SSO) services. By using OAuth 2.0, websites can ease password management for their users, as well as saving them the inconvenience of re-typing attributes that are instead stored by identity providers and provided to relying parties as required.

OAuth 2.0 is very widely used on Chinese websites, and there is a correspondingly rich infrastructure of identity providers (IdPs) providing identity services using OAuth 2.0. For example, some relying parties (RPs), such as the travel site Ctrip, support as many as eight different IdPs. At least ten major IdPs offer OAuth 2.0-based identity management services. RPs wishing to offer users identity management services from multiple IdPs must support the peculiarities of a range of different IdP implementations of OAuth 2.0.

Use of OAuth 2.0 by Facebook, Google and Microsoft has previously been studied, and issues have been identified [2–5]. However, despite the wide use of OAuth 2.0 for SSO in China, the authors are not aware of any published research on the properties of Chinese implementations. The very large and essentially self-contained OAuth 2.0 infrastructure in China is an important area for study, motivating the work described here. Also, as an early adopter of OAuth 2.0, lessons learnt from studying the Chinese infrastructure may apply globally.

OAuth 2.0 is used to protect access to hundreds of millions of user accounts in China alone, and so its security in practice is very important. Assessing practical security is non-trivial, especially as system operation relies on closed code and proprietary specifications and implementation guidance. In the absence of detailed specifications, security assessments require exhaustive experimental evaluation and analysis. In this paper we report on such investigations, including a detailed discussion of serious vulnerabilities found. We also provide recommendations for system improvements that address the identified vulnerabilities.

The paper is structured as follows. §2 introduces OAuth 2.0 and describes related work. In §3 we give two general classes of vulnerability in OAuth 2.0 SSO systems, both of which have been observed in practice. §4 covers our study of real-world OAuth 2.0 systems in China, including details of instances of the classes of vulnerability described in §3. Possible reasons for these vulnerabilities are considered in §5, together with proposed mitigatations.

2 Background and Related Work

OAuth 2.0 OAuth 2.0 [1] allows an application to access resources protected by a resource server on behalf of the resource owner, by consuming an access token issued by the authorisation server. OAuth 2.0 involves four roles. The *Resource Owner* is a host acting on behalf of an end user who can grant access to protected resources. The *Resource Server* is a server which stores the protected resources and consumes access tokens provided by an authorisation server. The *Client* is an application running on a server, which makes requests on behalf of the resource owner (the *Client* is the RP when OAuth 2.0 is used for SSO). The *Authorisation Server* generates access tokens for the client, after authenticating the resource owner and obtaining its authorisation (the *Resource Server* and *Authorisation Server* together constitute the IdP when OAuth 2.0 is used for SSO).

In order to use OAuth 2.0 for SSO, the resource server and authorisation server together play the IdP role, the client plays the role of the RP, and the resource owner corresponds to the user. OAuth 2.0 SSO systems build on user agent (UA) redirections, where a user (U) wishes to access services protected by the relying party (RP) which consumes the access token generated by the identity provider (IdP). The UA is typically a web browser. The IdP provides ways to authenticate the user, asks the user to allow the RP to access the user's attributes, and generates an access token. The RP uses the access token to access the user's attributes using an API provided by the IdP.

OAuth 2.0 supports four ways for RPs to obtain access tokens, namely Authorisation Code Grant, Implicit Grant, Resource Owner Password, and Client Credentials Grant. In this paper we are only concerned with the Authorisation Code Grant procedure, outlined below.

1. U $\rightarrow$ RP: The user clicks a login button on the RP website, as displayed by the UA, causing the UA to send a HTTP request to the RP.

2. RP → UA: The RP produces an OAuth 2.0 authorisation request and sends it back to the UA. The authorisation request includes *client_id*, the identifier for the client, which the RP registered with the IdP previously; *response_type=code*, indicating the Authorisation Code Grant method; *redirect_uri*, the URI to which the IdP redirects the UA after access is granted; *state*, an opaque value used by the RP to maintain state between request and callback (step 6 below); and the *scope* of the requested permission.
3. UA → IdP: The UA redirects the request received in step 2 to the IdP.
4. IdP → UA: If the user has already been authenticated by the IdP, then steps 4/5 are skipped. If not, the IdP returns a login form used to collect user authentication data.
5. U → UA → IdP: The user completes the login form and grants permission for the RP to access the attributes stored by the IdP.
6. IdP → UA: After using the login form data to authenticate the user, the IdP generates an authorisation response and sends it to the UA. This contains *code*, the IdP-generated authorisation code, and *state*, sent in step 2.
7. UA → RP: The UA redirects the response received in step 6 to the RP.
8. RP → IdP: The RP produces an access token request and sends it to the IdP token endpoint directly (i.e. not via the UA). The request includes the *client_id*, the *code* generated in step 6, the *redirect_uri* and also a *client_secret* shared between the IdP and the RP.
9. IdP → RP: The IdP checks the *client_id*, *client_secret*, *code* and *redirect_uri* and responds to the RP with *access_token*, an access token.
10. RP → IdP: The RP passes *access_token* to the IdP via a defined API to request the user attributes.
11. IdP → RP: The IdP checks *access_token* and, if satisfied, sends the requested user attributes to the RP.

Identity Federation for OAuth 2.0 Like OpenID [6], OAuth 2.0 does not support identity federation as defined in Shibboleth [7] or SAML [8]. A commonly used means of achieving identity federation involves the RP locally binding the user's RP-managed account with the user's IdP-managed account, using the unique identifier for the user generated by the IdP. After binding, a user can log in to the RP-managed account using his or her IdP-managed account.

Such a federation scheme operates as follows. After receiving the access token, the RP retrieves the user's IdP-managed account identifier and binds the user's RP-managed account identifier to the IdP-managed account identifier. When the user next tries to use his or her IdP-managed account to log in to the RP, the RP looks in its account database for a mapping between the supplied IdP-managed identifier and an RP-issued identifier. If such a mapping exists, then the RP simply logs the user in to the corresponding RP-managed user account.

In real-world OAuth 2.0 SSO systems supporting federation, RPs typically use one of two ways to perform the binding. Firstly, suppose a user chooses to log using SSO. After finishing the authorisation process with the IdP, the user is asked either to bind the IdP-managed account to his or her RP-managed

account or to log in to the RP directly. The user will need to provide his/her RP-managed account information (e.g. account name and password) to complete the binding. Alternatively, after a user has already logged into an RP, he or she can initiate a binding operation. After being authenticated by the IdP and granting permission to the RP, the user can bind his or her RP-managed account to the IdP-managed account. After binding, many RPs allow users to log in to their websites using an IdP-managed account.

Related Work The OAuth 2.0 specification [1] and threat model [9] describe possible threats and countermeasures. Pai et al. [10] confirm a security issue described in the OAuth 2.0 Threat Model ([9] §4.1.1) using the Alloy framework [11]. Chari et al. [12] analyse OAuth 2.0 in the Universal Composability Security framework [13], and show that OAuth 2.0 is secure if all communications links are SSL-protected. Frostig and Slack [14] discovered a cross site request forgery attack in the Implicit Grant flow of OAuth 2.0, using the Murphi framework [15]. However, all this work is based on abstract models of OAuth 2.0, and so delicate implementation details are ignored.

To understand the real-world security of OAuth 2.0, Wang et al. [5] examined a number of deployed SSO systems, focussing on a logic flaw present in many such systems, including OpenID. In parallel, Sun & Beznosov [4] also studied deployed systems. Both these studies restricted their attention to systems using English. Indeed, very little research has been conducted on the security of OAuth 2.0 systems using other languages, some of which, like those in Chinese, have very large numbers of users. In this paper, we redress this imbalance by reporting on an analysis of Chinese-language OAuth 2.0 systems.

Like Sun & Beznosov [4], this paper considers the security of deployed OAuth 2.0 systems; however, there are two major differences in approach. First, we do not exploit specific web browser and application vulnerabilities. Second, Sun & Beznosov focus on attacks involving stealing the user's access token from an RP; specific web browser and application vulnerabilities are used to allow the attacks. By contrast, this paper focuses on the security of OAuth 2.0 SSO systems supporting identity federation, and the security flaws identified do not exploit browser or application vulnerabilities.

3 Threats to OAuth 2.0 Identity Federation

OAuth 2.0 is intended to let an RP gain limited access to a service either on behalf of the user or for the RP's own purposes. Hence identity federation, as in Shibboleth [7] or SAML [8], is not supported. As discussed in §2, in order to provide identity federation for OAuth 2.0, RPs typically employ ad hoc means to bind an RP-managed account to an IdP-managed account.

Cross Site Request Forgery Attacks A cross site request forgery (CSRF) attack [16–22] operates in the context of an ongoing interaction between a target

web browser (running on behalf of a target user) and a target website. In the attack, a malicious website somehow causes the browser to initiate a request of the attacker's choice to the target site. This can cause the target site to execute actions without the involvement of the user. In particular, if the target user is currently logged in to the target site, the browser will send cookies containing the target user's authentication tokens, along with the attacker-supplied request, to the target site. The target site will process the malicious request as if it was initiated by the target user. The target browser could be made to send the spurious request in various ways; e.g., a malicious site visited by the browser could use the HTML <img> tag's src attribute to specify the URL of a malicious request, causing the browser to silently use a GET method to send the request.

The OAuth 2.0 specification ([1], §10.12) describes a possible CSRF attack in which the target website corresponds to *redirect_uri*, i.e. the URI to which the target browser is directed by OAuth 2.0. The attack involves an attacker causing the target browser to send the target site a request containing the attacker's own authorisation code or access token. As a result, the target site might associate the attacker's protected resources with the target user's current session; possible undesirable effects could include saving user credit card details or other sensitive user data to an attacker-controlled location.

In this paper we show that a CSRF attack could also be used to attack the federation process of an OAuth 2.0 SSO system, with potentially very serious effects. Suppose a target UA is logged in to a target RP. The UA visits the malicious site, perhaps by following a link on the target RP's site. The malicious site now forces the UA (unbeknownst to the user) to send a request to the target site containing a binding request for the attacker's IdP account. If not appropriately secured, the target website might now bind the attacker's IdP-managed account to the target user's RP-managed account. The attacker can now log in to the target user's RP-managed account at will. If the vulnerability is present, this simple attack could be launched on a very large scale to take control of multiple RP-managed accounts. Note that the attacker would need to use a distinct IdP-managed account for each instance of the attack, although this should not be an issue in practice.

The OAuth 2.0 specification recommends inclusion of a *state* parameter in the authorisation request to protect against CSRF attacks. This allows the RP to verify the source of a request by matching the *state* value to the user-agent's authenticated state (as recorded in a session cookie). However, for this to work the *state* value must not be guessable; otherwise the attacker could include the guessed value in its fraudulent request. However, despite this advice, we have found that many real-world RPs either omit the *state* parameter from the authorisation request or fail to use *state* correctly (e.g., some RPs allocate a fixed value to *state*). We have also observed that some RPs do not check the correctness of the *state* value even if it is non-guessable. As a result, many RPs supporting identity federation are vulnerable to a CSRF attack against the RP's redirect URL, allowing an attacker to gain full access to the victim's RP-managed account without knowing the user's account name or password.

Logic Flaws To achieve identity federation, the RP must support a way to bind the user's RP-managed and IdP-managed accounts. The binding operation is clearly security-critical since, after binding, the owner of the IdP-managed account has full control over the RP-managed account. Design flaws in binding could allow an attacker to bind the victim user's RP-managed account to the attacker's IdP-managed account, without the knowledge of the user.

Binding security largely depends on the RP, since binding is done by the RP and the IdP simply provides an access token. The RP chooses how binding works, and decides whether or not to perform it. Since there is no standard for binding, different RPs use different ways of completing it. As a result the security of binding largely depends on the security awareness of the implementers. This is clearly dangerous, and the almost inevitable result is that some RP implementations of OAuth 2.0 SSO contain serious logic flaws, potentially enabling an attacker to bind its IdP- managed account to any RP-managed account. The consequences of such an attack could be very serious indeed.

Adversary Model We assume all RPs and IdPs are benign, i.e. we only consider attacks involving third parties. However, we suppose an attacker can share malicious links and/or post comments which could contain malicious content on a benign RP website, and send malicious links to the victim, e.g. via email. The malicious content constructed by the attacker could cause the browser to initiate an HTTP request to either the RP or the IdP (or both).

4 Case Studies

We report on an investigation of the security of real-world implementations of SSO systems using OAuth 2.0, including both RPs and IdPs. In particular we looked for vulnerabilities of the types described in §3 above. We focussed our study on RPs using OAuth 2.0 for identity federation, especially those supporting the second method of binding specified in §2. This is because the first method requires a user to provide account information to complete binding, which seems to make using a CSRF attack to achieve a false binding much more difficult.

Conducting a security analysis of commercially deployed OAuth 2.0 SSO systems requires a number of challenges to be addressed. These include lack of access to detailed specifications for the SSO systems, undocumented RP and IdP source code, and the complexity of APIs and/or SDK libraries in deployed SSO systems. The methodology we used is similar to that employed by Wang et al. [5] and Sun & Beznosov [4], i.e. we analysed the browser relayed messages (BRMs). We treated the RPs and IdPs as black boxes, and analysed the BRMs produced during binding to look for possible exploit points.

We used Fiddler (`http://www.telerik.com/fiddler`) to capture the BRMs sent between RPs and IdPs; we also developed a Java program to parse the BRMs to simplify analysis and to avoid mistakes resulting from manual inspection. After confirming an exploit point, we used widely deployed browsers, including IE, Safari, Firefox, and Chrome, to replay or relay the browser request. At no

time during our experiments did we access any user accounts without the explicit permission of the user concerned.

Renren Network Renren Network (`http://www.renren.com`) is a Chinese social networking service which has been described as the 'Facebook of China'. It claims to have about 320 million active users. Renren Network supports several SSO IdPs, including Baidu [23] and China Mobile [24]. A user can thus sign in to Renren Network using a Baidu or China Mobile account.

A Renren-Baidu account binding attack. In order to use an IdP-managed account to log in to Renren via OAuth 2.0, a user's Renren-managed account must first be bound to an IdP-managed account. Suppose a user already logged in to Renren wants to bind his or her Renren-managed and Baidu (IdP) accounts (step 1 in §2.2). Renren generates an OAuth 2.0 authorisation request (step 2) and redirects the user browser to Baidu (step 3). The authorisation request generated by Renren does not contain a *state value*. After authenticating the user (steps 4 and 5), Baidu generates the authorisation response (step 6), which only contains the *redirect_uri* and *code*. The user agent will send the authorisation response to Renren (step 7) with cookies containing the user's session identifier. Renren uses the *code* to exchange an access token with Baidu (steps 8 and 9). Renren then uses the access token to retrieve the user's Baidu account's identifier (steps 10 and 11), and employs the user's session identifier to retrieve the user's Renren account identifier. Finally, Renren binds the user's Renren-managed and Baidu-managed accounts, based on the identifiers it received earlier.

The RP needs to know the identifiers of the user's RP-managed and IdP-managed accounts in order to complete binding. Renren does not implement any measures to protect against a CSRF attack on the *redirect_uri*. Thus if an attacker can replace the *code* in the authorisation response with its own IdP-generated *code*, then the identifier that the RP retrieves from the IdP will correspond to the attacker's IdP-managed account. This will cause the victim user's RP-managed account to be bound to the attacker's IdP-managed account.

We tested the viability of such an attack by initiating the Renren-Baidu authorisation process. We used a Baidu account to perform authentication to Baidu (acting as the IdP). Baidu then generated and sent a response (as in step 6 of §2.2) containing a *redirect_uri* and *code*. We intercepted this response and posted it as a link on a web forum. If a victim user who has previously logged in to Renren clicks on the link, the victim's browser will submit the request with the cookie containing the victim's session identifier to the *redirect_uri* of Renren. When we tested this, Renren successfully bound the victim's account to our IdP-account. We could thus access the victim's account via our IdP-managed account, without knowing the victim user's account name or password.

A Renren-China Mobile account binding attack. We analysed the data flow for Oauth 2.0 SSO performed between Renren Network (RP) and China Mobile (IdP). Unlike Renren-Baidu, both the authorisation request (step 1) and the authorisation response (step 6) contain a *clientState* value, which we assume is used by Renren to try to prevent CSRF attacks.

However, we observed that the *clientState* value is the same for multiple requests and responses (in fact *clientState*=9 in all requests and responses we observed). That is, the *clientState* is guessable. Thus, and as we observed in practical tests, Renren-China Mobile federation is also susceptible to a CSRF attack that enables an attacker to bind his or her own China Mobile-managed account to a victim user's Renren-Managed account.

For both the above scenarios, the response generated in step 6 begins with the Renren host name. Thus, if posted on a website it will resemble a benign sharing link, so a victim user will have no reason not to click on it.

Ctrip Ctrip (`www.ctrip.com`) is a China-focused travel agency with around 60 million members and 2.5 million user reviews. Its services cover around 9,000 flight routes and 200,000 hotels across the world. In order to access Ctrip services, a user must have a membership with either Ctrip itself or with one of the SSO systems it supports. Ctrip supports eight OAuth 2.0 SSO IdPs, including Renren [25], Wangyi [26], Taobao [27], MSN [28] and Sina [29].

A logic flaw in Ctrip. To study the security of the Ctrip-supported SSO systems, we analysed BRMs exchanged between Ctrip (the RP) and Renren (the IdP) while the user is binding his or her Ctrip-managed and Renren-managed accounts using the second method described in §2. As for the Renren-Baidu binding, the OAuth authorisation request in step 2 and the authorisation response in step 6 do not contain the *state* value. This immediately suggested that Ctrip-Renren binding might be vulnerable to a CSRF attack. To test this, we relayed an intercepted IdP-generated authorisation response to a victim user agent which had already logged in to Ctrip. The user agent sent the authorisation response to Ctrip, along with the cookies containing the victim user's session identifier. However, instead of binding the attacker's Renren account to the victim user's Ctrip account, Ctrip just responded with a web page asking the user to input his or her account name and password. We also tried to perform the attack on other IdPs supported by Ctrip. In each case, Ctrip responded with a web page requesting the user to input his or her account name and password. Hence Ctrip, by some means, resists the attack described above.

However, we observed that the request generated in step 1 contains a *Uid*, the Ctrip-generated user identifier. Observing that Ctrip account identifiers are guessable, we conjectured that if we could replace the *Uid* value in the request generated in step 1 with the *Uid* corresponding to the victim user, then it might be possible to force Ctrip to bind the attacker's IdP-managed account to the victim user's Ctrip-managed account. We therefore tested this approach. In order not to cause damage to a real user of the Ctrip website, we modified the *Uid* value to correspond to an account created for the purposes of the experiment. We relayed the request to Ctrip and completed the authorisation procedure with the IdP. Ctrip responded with a blank web page with the URL *http://RP@Recp=0*, indicating that Ctrip had successfully bound the IdP-managed and Ctrip-managed accounts.

```
http://accounts.ctrip.com/member/RenrenLogin/Authorize.aspx?Action=B
&Uid=E60444782&BackUrl=http://my.ctrip.com/Home/Third/ThirdTransfer.aspx
```

Fig. 1. The request generated in step 1

To understand why Ctrip is vulnerable to this attack, we analysed all BRMs exchanged in both a normal binding operation (where a logged-in user initiates a binding operation) and an attack binding operation (where an attacker initiates the request in Fig. 1 without logging in to the *Uid* account). We observed that, in a normal binding operation the browser sent Ctrip the request in step 1 with cookies containing the user's session identifier. However, in the attack binding operation, as no cookies had previously been set for the *Uid* account, the user agent just sent the request (step 1). Ctrip generated the authorisation request and set a session identifier cookie for the *Uid* account (step 2). After receiving the IdP-generated authorisation response (step 6), the browser sent both the authorisation response and the cookie containing the session identifier to Ctrip. Ctrip treated the combination of session identifier and authorisation response as a legal binding operation, and so it bound the IdP-managed account to the victim user's Ctrip account. From this we deduced that Ctrip fails to verify the validity of the request in step 1 before generating the authorisation request, i.e. Ctrip does not check the request is initiated by the real owner of *Uid*. An attacker can thus successfully forge a request to bind his or her IdP-managed account to the *Uid* account, i.e. an attacker can circumvent Ctrip's user authentication.

A generic Ctrip binding attack. We used our observations regarding the operation of the Ctrip website to devise the following attack on federation. When a user initiates a binding operation to a different IdP, only the *IdPLogin* value (the *RenrenLogin/Authorize.aspx* in Fig. 3) changes in the request. An attacker can use this to control the binding between RP and IdP. That is, an attacker can bind any RP-managed account to any IdP just by replacing the *IdPLogin* value and the *Uid* value in the request sent in step 1. We further observed that Ctrip provides a user forum to share information and initiate events. An attacker can readily find user *Uid* values by examining the forum, since Ctrip does not effectively conceal them. Using a simple guessing attack, many *Uid* values can be recovered from the poorly-protected forum entries.

We reported these flaws to the Ctrip Security Response Centre and helped Ctrip fix them. Ctrip has listed this report on its acknowledgement page.

5 Discussion and recommendations

Scope of study We studied a total of 60 Chinese RPs supporting SSO via identity federation to an IdP using OAuth 2.0. Of these, 14 only support the first method of binding described in §2.3, and so are not vulnerable to the CSRF attack in §4. Of the remaining 46, a total of 21, i.e. almost half, are vulnerable to the CSRF attack. Many millions of users were potentially affected by this vulnerability, since Renren alone has around 320 million active users.

We further analysed the BRMs to find out why the 21 RPs are vulnerable. Since these RPs support an average of at least three IdPs, we had to analyse 68 distinct sets of RP-IdP browser relayed messages. Of these 68 OAuth 2.0 authorisation processes, 48 do not involve the use of any countermeasures to a CSRF attack. However even in the 20 cases where countermeasures were employed, poor implementation means that the attack remains possible.

One possible reason why some implementers use a constant value for *state* is that the IdP-provided documentation [23, 25, 29, 27, 26, 30] does not describe how to generate it. In the absence of guidance on the use of *state*, implementers may reasonably, but falsely, believe they have implemented effective protection against CSRF attacks by using a constant value. Secondly, some RPs which use the same *redirect_uri* for multiple IdPs use the *state* value to distinguish between IdPs, i.e. so they can determine to which IdP the RP-managed account should be bound. That is, they do not appear to understand the intended purpose of the *state* variable, and the need for such values to be non-guessable; as a result they may use guessable *state* values, which again represents a possible vulnerability. Thirdly, even if the *state* value is 'opaque' (i.e. non-guessable), problems can still arise if the RP does not perform the necessary checks. In particular, we discovered that some RPs fail to check that the *state* value in the request used to trigger binding correctly maps to the user's session identifier.

In summary, there are a variety of ways in which the binding vulnerability can arise. The common element is the lack of clear and detailed guidance for the use of CSRF countermeasures in the context of identifier binding for federation. This is hardly surprising since identity binding is not standardised within the OAuth specifications. This lack of clear standards for identity federation is the main underlying source of all the vulnerabilities we have observed.

Recommendations OAuth 2.0 SSO systems have been widely deployed by Chinese RPs and IdPs, and it appears likely that increasing numbers of Chinese RPs and IdPs will implement OAuth 2.0 for SSO. However, our study has revealed serious vulnerabilities in existing systems, and there is a significant danger that these vulnerabilities will be replicated in future systems. Below we make a number of recommendations, directed at both RPs and IdPs, designed to address the identified vulnerabilities. These recommendations should help to address problems in current systems was well as assist in ensuring that future systems are built in a more robust way. Ideally, a standardised federation system for OAuth 2.0 would be developed, and these recommendations are also intended as input to such work.

In OAuth 2.0 SSO systems supporting identity federation, RPs design the binding process. We have the following recommendations for RPs.

- **Deploy countermeasures against CSRF attacks**. One reason the OAuth 2.0 systems we investigated are vulnerable to CSRF attacks is that the RPs do not implement countermeasures. Many IdPs [23, 25, 29, 26] recommend RPs to include the *state* parameter in the OAuth 2.0 authorisation request, and RPs should follow such recommendations.

- **Do not use a constant or predictable *state* value**. Some RPs include a fixed *state* value in the OAuth 2.0 authorisation request. In this case an attacker can forge a response, since the RP cannot distinguish a legitimate response produced by a valid user from a forged response. Thus the inclusion of the *state* value does not mitigate CSRF attacks. Thus RPs must generate a non-guessable *state* value bound to the user's session identifier, so that the *state* value can be used to verify the validity of the response.
- **Check the *state* value**. RPs that include an opaque *state* value in their OAuth 2.0 request should check the *state* value in the response before completing binding. We recommend that RPs use a session-dependent *state* value, although such a procedure slightly enlarges the state table which the RP must maintain in order to validate the *state* value.
- **Require the user to input account information**. Perhaps the simplest way to prevent the CSRF attack is to require users to input their account names and passwords before completing binding. However, the user will then be required to 'log in' twice during a single session, damaging the user experience; this also goes against the OAuth 2.0 design goals.

In an OAuth 2.0 SSO system, the IdP designs the OAuth 2.0 protocol process and provides the API for RPs. An RP wishing to support a particular IdP must therefore comply with the requirements of that IdP, and so the IdPs play a critical role in the system. We have the following recommendations for IdPs.

- **Include the *state* in sample code**. IdPs typically provide sample code to help RP developers correctly code interactions with the IdP. However, many [23, 25, 24, 29, 26, 28, 30] fail to include the *state* value in their sample code. This may be the main reason why more than half of the RP-IdP interactions we analysed are vulnerable to CSRF attacks. Including the *state* value in IdP sample code should help encourage RPs to reduce the risk of CSRF attacks.
- **Emphasise the consequences of CSRF attacks**. Since IdPs are responsible for designing the way in which OAuth 2.0 is used, RP developers must use the IdP-provided documentation to enable interoperation. In the examples of IdP documentation we examined, many simply mention the possibility of CSRF attacks without emphasising the potentially very serious consequences. This may help explain why some RPs do not appear to take the CSRF threat as seriously as they should.

Concluding Remarks We studied the security of 60 implementations of OAuth 2.0 for federation-based SSO, as deployed by leading Chinese websites. We discovered that nearly half are vulnerable to CSRF attacks against the federation process, allowing serious compromises of user accounts. These attacks allow a malicious third party to bind its IdP-managed account to a user's IdP-managed account, without knowing the user's account name or password. As a result of the lack of a standardised federation process, we have further discovered logic flaws in real-world implementations of federation, which again allow binding of an attacker's IdP-managed account to a user's RP-managed account.

We reported our findings to all RPs and IdPs affected by the attacks; we also provided them with possible mitigations. We hope our study will be of broader value in warning IdPs and RPs of the dangers of CSRF attacks on OAuth 2.0 identity federation process. Ideally, a robust federation process for OAuth 2.0 will be standardised, helping to reduce the likelihood of future problems.

References

1. Hardt, D.: The OAuth 2.0 authorization framework (2012) http://tools.ietf.org/html/rfc6819.
2. Hanna, S., Shin, R., Akhawe, D., Boehm, A., Saxena, P., Song, D.: The emperor's new APIs: On the (in)secure usage of new client-side primitives. In: Proc. W2SP 2010. (2010)
3. Miculan, M., Urban, C.: Formal analysis of Facebook Connect Single Sign-On authentication protocol. In: Proc. SofSem 2011, OKAT (2011) 99–116
4. Sun, S.T., Beznosov, K.: The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems. In Yu, T., Danezis, G., Gligor, V.D., eds.: Proc. CCS '12, ACM (2012) 378–390
5. Wang, R., Chen, S., Wang, X.: Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In: Proc. IEEE Symp. on Security and Privacy 2012, IEEE (2012)
6. Recordon, D., Fitzpatrick, B.: Open ID Authentication 2.0 — Final. (2007) http://openid.net/specs/openid-authentication-2_0.html.
7. Morgan, R., Cantor, S., Carmody, S., Hoehn, W., Klingenstein, K.: Federated security: The Shibboleth approach. Educause Quarterly **27** (2004) 12–17
8. Scott, C., Kemp, J., Philpott, R., Maler, E.: Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. (2005) http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.
9. Lodderstedt, T., McGloin, M., Hunt, P.: OAuth 2.0 Threat Model and Security Considerations. (2013) http://tools.ietf.org/html/rfc6749.
10. Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S.: Formal vericication of OAuth 2.0 using alloy framework. In: Proc. CSNT 2011, IEEE (2011) 655–659
11. Jackson, D.: Alloy 4.1. (2010) http://alloy.mit.edu/community/.
12. Chari, S., Jutla, C.S., Roy, A.: Universally composable security analysis of OAuth v2.0. IACR Cryptology ePrint Archive **2011** (2011) 526
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. FOCS 2001, IEEE Computer Society (2001) 136–145
14. Slack, Q., Frostig, R.: Murphi Analysis of OAuth 2.0 Implicit Grant Flow. (2011) http://www.stanford.edu/class/cs259/WWW11/.
15. Dill, D.L.: The mur*phi* verification system. In Alur, R., Henzinger, T.A., eds.: Proc. CAV '96. Volume 1102 of LNCS., Springer (1996) 390–393
16. Burns, J.: Cross site reference forgery: An introduction to a common web application weakness. Security Partners, LLC (2005) http://dl.packetstormsecurity.net/papers/web/XSRF_Paper.pdf.
17. Jovanovic, N., Kirda, E., Kruegel, C.: Preventing cross site request forgery attacks. In: Proc. SecureComm 2006, IEEE (2006) 1–10
18. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In Ning, P., Syverson, P.F., Jha, S., eds.: Proc. CCS 2008, ACM (2008) 75–88

19. Zeller, W., Felten, E.W.: Cross-site request forgeries: Exploitation and prevention. Bericht, Princeton University (2008)
20. Mao, Z., Li, N., Molloy, I.: Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In Dingledine, R., Golle, P., eds.: Proc. FC 2009. Volume 5628 of LNCS., Springer (2009) 238–255
21. Shahriar, H., Zulkernine, M.: Client-side detection of cross-site request forgery attacks. In: Proc. ISSRE 2010, IEEE Computer Society (2010) 358–367
22. De Ryck, P., Desmet, L., Joosen, W., Piessens, F.: Automatic and precise client-side protection against CSRF attacks. In Atluri, V., Díaz, C., eds.: Proc. ESORICS 2011. Volume 6879 of LNCS., Springer (2011) 100–116
23. Baidu Inc.: Baidu Open Connect. (2014) `http://developer.baidu.com/wiki/index.php?title=docs/oauth/authorization`.
24. China Mobile Communications Corporation: ChinaMobile Open Connect. (2014) `http://dev.10086.cn/wiki/?p5_01_02`.
25. Renren Network: Renren Open Connect. (2014) `http://wiki.dev.renren.com/wiki/Authentication`.
26. Wangyi Inc.: Wangyi Open Connect. (2014) `http://reg.163.com/help/help_oauth2.html`.
27. Taobao Marketplace: Taobao Open Connect. (2014) `http://open.taobao.com/doc/detail.htm?id=118`.
28. Microsoft: Microsoft Live Connect. (2014) `http://msdn.microsoft.com/en-us/library/live/hh243647.aspx`.
29. Sina Corp: Sina Open Connect. (2014) `http://open.weibo.com/wiki/Oauth2/authorize`.
30. Douban.com: Douban Open Connect. (2014) `http://developers.douban.com/wiki/?title=oauth2`.