Your Code Is My Code: Exploiting a Common Weakness in OAuth 2.0 Implementations

Wanpeng Li[1(✉)], Chris J. Mitchell[2], and Thomas Chen[1]

[1] Department of Electrical and Electronic Engineering, City, University of London, London, UK
{Wanpeng.Li,Tom.Chen.1}@city.ac.uk
[2] Information Security Group, Royal Holloway, University of London, Egham, UK
me@chrismitchell.net

Abstract. Many millions of users routinely use their Google, Facebook and Microsoft accounts to log in to websites supporting OAuth 2.0-based single sign on. The security of OAuth 2.0 is therefore of critical importance, and it has been widely examined both in theory and in practice. In this paper we disclose a new class of practical attacks on OAuth 2.0 implementations, which we call Partial Redirection URI Manipulation Attacks. An attack of this type can be used by an attacker to gain a victim user's OAuth 2.0 code (a token representing a right to access user data) without the user's knowledge; this code can then be used to impersonate the user to the relevant relying party website. We examined 27 leading OAuth 2.0 identity providers, and found that 19 of them are vulnerable to these attacks.

1 Introduction

Since the OAuth 2.0 authorisation framework was published at the end of 2012 [8], it has been adopted by a large number of websites worldwide as a means of providing single sign-on (SSO) services. By using OAuth 2.0, websites can reduce the burden of password management for their users, as well as saving users the inconvenience of re-entering attributes that are instead stored by identity providers and provided to relying parties as required.

There is a correspondingly rich infrastructure of identity providers (IdPs) providing identity services using OAuth 2.0. This is demonstrated by the fact that some Relying Parties (RPs), such as the website USATODAY[1], support as many as six different IdPs—see Fig. 1.

As discussed in Sect. 4, the security of OAuth 2.0 has been analysed both theoretically, e.g. using formal methods, and practically, involving looking at implementations of OAuth 2.0. The research methodology used in most of this

[1] https://login.usatoday.com/USAT-GUP/authenticate/?.

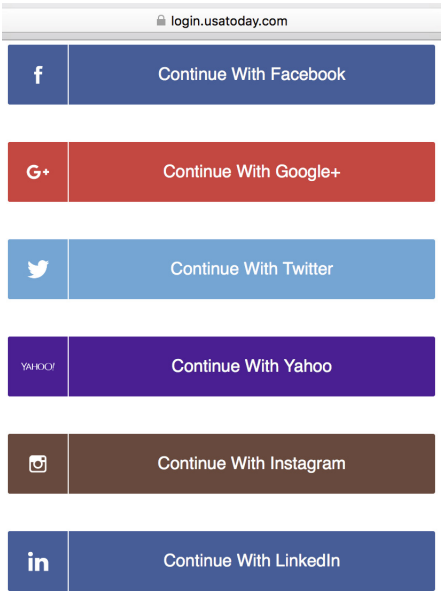

Fig. 1. The OAuth 2.0 IdPs supported by USATODAY.

work involves treating RPs and IdPs as black boxes; because of the inherent limitations of this approach, it is likely that potential implementation flaws and attack vectors exist that have yet to be found. Illustrating this, in this paper we disclose a new class of practical attacks on OAuth 2.0 implementations, which we call Partial Redirection URI Manipulation (PRURIM) attacks, that affect many leading real-world IdPs. These attacks either allow an attacker to log in to the RP as the victim user or enable compromise of potentially sensitive user information. We examined 27 leading OAuth 2.0 identity providers, and found that 19 of them are vulnerable to PRURIM attacks.

OAuth 2.0 is used to protect many millions of user accounts and sensitive user information stored at IdPs (e.g. Facebook, Google and Microsoft) and RP servers around the world. It is therefore vitally important that the issues we have identified are addressed urgently, and that IdPs take actions to mitigate the threats from PRURIM attacks. We have therefore notified the IdPs we have found to be vulnerable to these attacks.

To summarise, we make the following contributions:

- We describe a new class of practical attacks, PRURIM attacks, on OAuth 2.0 implementations. These attacks can be used to gain a victim user's OAuth 2.0 code without the user's knowledge.
- We examined the security of 27 leading OAuth 2.0 identity providers, and found that 19 of them are vulnerable to PRURIM attacks.
- We propose practical improvements which can be adopted by OAuth 2.0 RPs and IdPs that address the identified problems.

– We reported our findings to the affected IdPs and helped them fix the problems we identified.

The remainder of this paper is structured as follows. Section 2 provides background on OAuth 2.0. In Sect. 3 we describe implementation strategies that RPs use to support multiple IdPs. Section 4 summarises previous work analysing the security of real world OAuth 2.0 implementations. Section 5 describes the PRURIM attacks, which are a threat to RPs that support multiple IdPs. In Sect. 6, we report our findings and discuss why PRURIM attacks are possible. In Sect. 7, we propose possible mitigations for these attacks. Section 8 describes the disclosures made to affected IdPs, and the responses we received from them. Section 9 concludes the paper.

2 Background

2.1 OAuth 2.0

The OAuth 2.0 specification [8] describes a system that allows an application to access resources (typically personal information) protected by a *resource server* on behalf of the *resource owner*, through the consumption of an *access token* issued by an *authorization server*. In support of this system, the OAuth 2.0 architecture involves the following four roles (see Fig. 2).

1. The *Resource Owner* is typically an end user.
2. The *Resource Server* is a server which stores the protected resources and consumes access tokens provided by an authorization server.
3. 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).
4. The *Authorization Server* generates access tokens for the client, after authenticating the resource owner and obtaining its authorization (the *Resource Server* and *Authorization Server* together constitute the IdP when OAuth 2.0 is used for SSO).

Figure 2 provides an overview of the operation of the OAuth 2.0 protocol. The client initiates the process by sending (1) an authorization request to the resource owner. In response, the resource owner generates an authorization grant (or authorization response) in the form of a *code*, and sends it (2) to the client. After receiving the authorization grant, the client initiates an access token request by authenticating itself to the authorization server and presenting the authorization grant, i.e. the code issued by the resource owner (3). The authorization server issues (4) an access token to the client after successfully authenticating the client and validating the authorization grant. The client makes a protected source request by presenting the access token to the resource server (5). Finally, the resource server sends (6) the protected resources to the client after validating the access token.

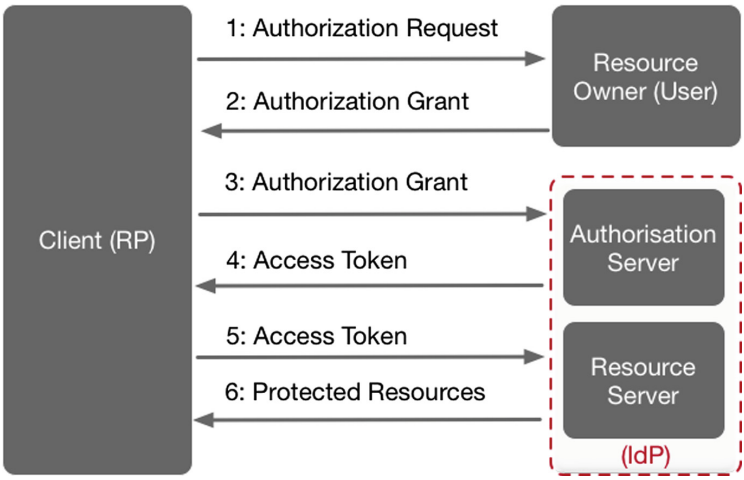

Fig. 2. OAuth 2.0 protocol flow.

2.2 OAuth 2.0 Used for SSO

In order to use OAuth 2.0 as the basis of an SSO system, the following role mapping is used:

- the resource server and authorization server together play the IdP role;
- the client plays the role of the RP;
- 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 RP which consumes the access token generated by the IdP. The UA is typically a web browser. The IdP provides ways to authenticate the user, asks the user to grant permission for the RP to access the user's attributes, and generates an access token on behalf of the user. After receiving the access token, the RP can access the user's attributes using the API provided by the IdP.

The OAuth 2.0 framework defines four ways for RPs to obtain access tokens, namely Authorization Code Grant, Implicit Grant, Resource Owner Password, and Client Credentials Grant. In this paper we are only concerned with the Authorization Code Grant and Implicit Grant protocol flows. Note that, in the descriptions below, protocol parameters given in bold font are defined as required (i.e. mandatory) in the OAuth 2.0 Authorization Framework [8].

RP Registration. The RP must register with the IdP before it can use OAuth 2.0. During registration, the IdP gathers security-critical information about the RP, including the RP's redirect URI, i.e. ***redirect_uri***, the URI to which the user agent is redirected after the IdP has generated the authorization response and sent it to the RP via the UA. As part of registration, the IdP issues the RP

with a unique identifier (***client_id***) and, optionally, a secret (*client_secret*). If defined, *client_secret* is used by the IdP to authenticate the RP when using the Authorization Code Grant flow.

Authorization Code Grant. We next briefly review the operation of OAuth 2.0 Authorization Code Grant. This flow relies on certain information having been established during the registration process, as described in Sect. 2.2. An instance of use of the protocol proceeds as follows.

1. U → RP: The user clicks a login button on the RP website, as displayed by the UA, which causes the UA to send an HTTP request to the RP.
2. RP → UA: The RP produces an OAuth 2.0 authorization request and sends it back to the UA. The authorization request includes ***client_id***, the identifier for the client which the RP registered with the IdP previously; ***response_type=code***, indicating that the Authorization Code Grant method is requested; *redirect_uri*, the URI to which the IdP will redirect the UA after access has been granted; *state*, an opaque value used by the RP to maintain state between the request and the callback (step 6 below); and *scope*, the scope of the requested permission.
3. UA → IdP: The UA redirects the request which it received in step 2 to the IdP.
4. IdP → UA: The IdP first compares the value of *redirect_uri* it received in step 3 (embedded in the authorization request) with the registered value (how *redirect_uri* is compared is described in Sect. 3.1); if the comparison fails, the process terminates. If the user has already been authenticated by the IdP, then the next step is skipped. If not, the IdP returns a login form which is used to collect the user authentication information.
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 → RP: After (if necessary) using the information provided in the login form to authenticate the user, the IdP generates an authorization response and redirects the UA back to the RP. The authorization response contains ***code***, the authorization code (representing the authorization grant) generated by the IdP; and *state*, the value sent in step 2.
7. 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 ***grant_type=authorization_code***, ***client_id***, *client_secret* (if the RP has been issued one), ***code*** (generated in step 6), and the ***redirect_uri***.
8. IdP → RP: The IdP checks ***client_id***, *client_secret* (if present), ***code*** and ***redirect_uri*** and, if the checks succeed, responds to the RP with *access_token*.
9. RP → IdP: The RP passes *access_token* to the IdP via a defined API to request the user attributes.
10. IdP → RP: The IdP checks *access_token* (how this works is not specified in the OAuth 2.0 specification) and, if satisfied, sends the requested user attributes to the RP.

Implicit Grant. The Implicit Grant protocol flow has a similar sequence of steps to Authorization Code Grant. We specify below only those steps where the Implicit Grant flow differs from the Authorization Code Grant flow.

2. RP → UA: The RP produces an OAuth 2.0 authorization request and sends it back to the UA. The authorization request includes ***client_id***, the identifier for the client which the RP registered with the IdP previously; ***response_type=token***, indicating that the Implicit Grant is requested; *redirect_uri*, the URI to which the IdP will redirect the UA after access has been granted; *state*, an opaque value used by the RP to maintain state between the request and the callback (step 6 below); and *scope*, the scope of the requested permission.
6. IdP → UA → RP: After (if necessary) using the information provided in the login form to authenticate the user, the IdP generates an access token and redirects the UA back to the RP using the value of *redirect_uri* provided in step 2. The access token is appended to *redirect_uri* as a URI fragment (i.e. as a suffix to the URI following a # symbol).

As URI fragments are not sent in HTTP requests, the access token is not immediately transferred when the UA is redirected to the RP. Instead, the RP returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI, including the fragment retained by the UA, and extracting the access token (and other parameters) contained in the fragment; the retrieved access token is returned to the RP. The RP can now use this access token to retrieve data stored at the IdP.

3 Supporting Multiple IdPs

As described in Sect. 1, many RPs support more than one IdP. This recognises the fact that users will have trust relationships with varying sets of IdPs — for example, one user may prefer to trust Facebook, whereas another may prefer Google.

In this section we describe two ways in which this is achieved in practice. The first approach (using redirect URIs) gives rise to the new class of attacks which we describe in Sect. 5. The second approach (explicit user intention tracking) gives rise to the IdP mix-up attacks described by Fett et al. [7].

3.1 Using Redirect URIs

One way in which an RP can support multiple IdPs is to register a different *redirect_uri* with each IdP, and to set up a sign-in endpoint for each. It can then use the endpoint on which it receives an authorization response to recognise which IdP sent it. For example, AddThis[2] has registered the URIs

– https://www.addthis.com/darkseid/account/register-facebook-return as its *redirect_uri* for Facebook, and

[2] http://www.addthis.com/.

– https://www.addthis.com/darkseid/account/register-google-return as its *redirect_uri* for Google.

If AddThis receives an authorization response at the endpoint https://www.addthis.com/darkseid/account/register-facebook-return?code=[code_generated_by_Facebook], (in step 7 of Sect. 2.2), it assumes that this response was generated by Facebook, and thus sends the authorization *code* to the Facebook server (step 8 of Sect. 2.2) to request an *access_token*.

The *redirect_uri* in OAuth 2.0. As described in Sect. 2.2, an RP must register with an IdP before it can use OAuth 2.0. The OAuth 2.0 Authorization Framework [8] defines the following two ways in which an IdP can register *redirect_uri* for an RP.

1. The IdP **should** require the RP to provide the complete redirection URI.
2. If requiring the registration of the complete redirection URI is not possible, the IdP **should** require the registration of the URI scheme, authority, and path. This allows the RP to dynamically vary only the query component of the redirection URI when requesting authorization.

As described in §3.1.2 of the OAuth 2.0 Authorization Framework [8], the redirection endpoint URI **must** be an absolute URI. The framework requires the authorization server to match the received *redirect_uri* value against the redirection URIs registered by the RP when a redirection URI is included in an authorization request. Also, if the *redirect_uri* registered by the RP includes the full redirection URI, the IdP **must** compare the two URIs using a simple string comparison [15].

Real-World Implementations of *redirect_uri* Checks. As noted above, the OAuth 2.0 Authorization Framework [8] requires the IdP to check the two URIs using a simple string comparison if the registered *redirect_uri* value includes the full redirection URI; however, this is not always done. In practice, we have identified three approaches used by real-world IdPs to check the *redirect_uri*.

– **Checking only the origin of *redirect_uri***. Many IdPs, including Facebook[3], Yahoo[4] and Microsoft[5], only check the *origin* part of *redirect_uri*. For example, suppose an RP registers https://www.RP.com/facebook-return as its *redirect_uri* with Facebook. When Facebook receives an authorization request generated by this RP, it only checks whether the origin part of *redirect_uri* in the authorization request matches https://www.RP.com, i.e. it ignores `/facebook-return`.
– **Checking *redirect_uri* using a simple string comparison**. Some IdPs, such as Google[6] and Amazon[7], execute a simple string comparison when performing a *redirect_uri* check (as required in [8]) on the authorization request.

[3] https://developers.facebook.com/docs/facebook-login/web.
[4] https://developer.yahoo.com/oauth2/guide/.
[5] https://msdn.microsoft.com/en-us/library/hh243647.aspx.
[6] https://developers.google.com/identity/protocols/OAuth2.
[7] http://login.amazon.com/website.

Other IdPs, such as OK[8] and Yandex[9], perform a *redirect_uri* check by executing a simple string comparison only when generating the authorization response, i.e. they accept an unauthorised OAuth 2.0 request as described in Listing 1.1, but refuse to generate an OAuth 2.0 response for such a request.

– **Issuing an IdP-generated value for *redirect_uri***. Some IdPs, such as ebay[10], issue a *redirect_uri* value (e.g. `Jerry_Smith-JerrySmi-TestOA-pkvmjju`) to the RP when the RP registers with the IdP. When the IdP receives an authorization request generated by this RP, it first compares the *redirect_uri* (i.e. `Jerry_Smith-JerrySmi-TestOA-pkvmjju` in this example) in the authorization request with the value it has stored in its database. If the two values agree, it generates an authorization response and sends it to the redirect URI that the IdP retrieved using the *redirect_uri* value (i.e. `Jerry_Smith-JerrySmi-TestOA-pkvmjju` in this example).

3.2 Explicit User Intention Tracking

Registering a different redirection URI for each IdP is not the only approach that could be used by an RP to support multiple IdPs. An RP can instead keep a record of the IdP each user wishes to use to authenticate (e.g. it could save the identity of the user's selected IdP to a cookie).

In this case, when a authorization response is received by the RP, the RP can retrieve the identity of the IdP from the cookie and then send the *code* to this IdP. This method is typically used by RPs that allow for dynamic registration, where using the same URI is an obvious implementation choice [7].

4 Security Properties of OAuth 2.0

OAuth 2.0 has been analysed using formal methods [1–4,7,17,20]. Pai et al. [17] confirmed a security issue described in the OAuth 2.0 Thread Model [14] using the Alloy Framework [9]. Chari et al. analysed OAuth 2.0 in the Universal Composability Security framework [4] and showed that OAuth 2.0 is secure if all the communications links are SSL-protected. Frostig and Slack [20] discovered a cross site request forgery attack in the Implicit Grant flow of OAuth 2.0, using the Murphi framework [6]. Bansal et al. [1] analysed the security of OAuth 2.0 using the WebSpi [2] and ProVerif models [3]. However, all this work is based on abstract models, and so delicate implementation details are ignored.

The security properties of real-world OAuth 2.0 implementations have also been examined by a number of authors [5,10,11,13,18,21,22,24]. Wang et al. [22] examined deployed SSO systems, focussing on a logic flaw present in many such systems, including OpenID. In parallel, Sun and Beznosov [21] also studied deployed OAuth 2.0 systems. Later, Li and Mitchell [10] examined the security

[8] https://apiok.ru/ext/oauth/.
[9] https://tech.yandex.com/oauth/.
[10] https://developer.ebay.com/Devzone/merchant-products/account-management/HowTo/oauth.html.

of deployed OAuth 2.0 systems providing services in Chinese. In parallel, Zhou and Evans [24] conducted a large scale study of the security of Facebook's OAuth 2.0 implementation. Chen et al. [5], and Shehab and Mohsen [18] have looked at the security of OAuth 2.0 implementations on mobile platforms. Finally, Li and Mitchell [11] conducted an empirical study of the security of the OpenID Connect-based SSO service provided by Google.

We conclude this review by mentioning prior art that has a close relationship to the PRURIM attacks described below.

- The **cross social-network request forgery** attack was described by Bansal, Bhargavan and Maffeis [1]. It applies to RPs using third party libraries, such as JanRain or GigYa, to manage their IdPs, as these RPs use the same login endpoint for all IdPs.
- A similar attack, the **Redirection URI Manipulation Attack**, is defined in §10.6 of the OAuth 2.0 Authorization Framework; in this attack, the attacker sets the *redirect_uri* in the authorization request to that of the attacker's own website (e.g. https://www.attacker.com).
- Another attack with a similar outcome, the **IdP mix-up attack** due to Fett et al. [7], works in the context of RPs using explicit user intention tracking to support multiple IdPs, as described in Sect. 3.2. For it to work, a network attack is needed to modify the http or https messages generated by the RP in step 1 (see Sect. 2.2). Li and Mitchell [12] argued that this attack would not be a genuine threat to the security of OAuth 2.0 if IdP implementations strictly follow the standard.

5 A New Class of Attacks

We now introduce PRURIM attacks, which can be used by a malicious party to collect a *code* belonging to a victim user without the user being aware. These attacks exploit the fact that many IdPs only check the origin part of the *redirect_uri* (as discussed in Sect. 3.1). In Sects. 5.2 and 5.3 we describe two variants of the attack with differing assumptions about the capabilities of the attacker.

5.1 Adversary Model

We suppose that the adversary has the capabilities of a **web attacker**, i.e. it can share malicious links or post comments which contain malicious content (e.g. stylesheets or images) on a benign website, and/or can exploit vulnerabilities in an RP website. The malicious content might trigger the web browser to send an HTTP/HTTPS request to an RP and IdP using either the GET or POST methods, or execute JavaScript scripts crafted by the attacker.

In addition, in the first of the two variants of the PRURIM attack described in Sect. 5.2, we suppose that the adversary can set up a server which acts as an OAuth 2.0 IdP; we refer to this as a Malicious IdP (MIdP). In the second PRURIM variant (see Sect. 5.3) we assume instead that the RP website contains a Cross-site scripting (XSS) vulnerability.

5.2 Using a MIdP

We divide our discussion of the first PRURIM attack variant into three parts. We first describe the core of the attack, in which the attacker is able to obtain a victim user's *code*. We then describe two ways in which knowledge of this *code* can be used to perform unauthorised actions.

This attack applies to both the authorization code grant and implicit grant flows. For simplicity we only present the attack for the authorization code grant flow. We describe real-world examples of these attacks in Sect. 6.

Obtaining the *Code*. As described in Sect. 3.1, many IdPs only check the origin of the *redirect_uri*. If the *redirect_uri* is not fully checked, an attacker can change part of it without the change being detected by the IdP. This observation underlies the following attack.

Suppose an attacker can, in some way, cause a victim user's browser to generate (unknown to the user) an unauthorised authorization request for the target IdP (TIdP) of the form given in Listing 1.1. This might, for example, be achieved by inserting the request in an *iframe* or *img* in an apparently innocent web page, which the victim user is persuaded to visit. When it receives this request, the TIdP will assume that it is a normal authorization request generated by the RP, as it only checks the origin part of the *redirect_uri*. It then authenticates the victim user, if necessary (see step 4 in Sect. 2.2), and then generates an authorization response. This response is sent to the URL https://RP.com/MIdP-return?code=[code_generated_by_TIdP].

When the RP receives this *code*, it first constructs an access token request which includes the *code*, and then sends it to the MIdP. The attacker (MIdP) now has the user's *code*; this *code* can now be used for a range of malicious purposes. We describe below two examples of how this value might be used.

```
1  // a normal authorization request generated by the RP supporting
       for target IdP (TIdP)
2  https://TIdP.com/auth2?
3  client_id=[client_id_generated_by_TIdP]&
4  redirect_uri=https://RP.com/TIdP-return&
5  response_type=code
6
7  // an unauthorised authorization request crafted by the attacker
       (MIdP)
8  https://TIdP.com/auth2?
9  client_id=[client_id_generated_by_TIdP]&
10 redirect_uri=https://RP.com/MIdP-return&
11 response_type=code
```

Listing 1.1. The partial redirect URI manipulate attack

An Impersonation Attack. An attacker with access to a victim user's *code* for a particular TIdP can use it to impersonate this user in the following way. The attacker first initiates a new login process at an RP using the attacker's own browser (we suppose this RP supports SSO using the TIdP). The attacker

chooses the TIdP as the IdP for this login process, and the attacker's browser is accordingly redirected to the TIdP. The attacker provides his/her own account information to the TIdP. After authenticating the attacker, the TIdP generates an authorization response containing a *code* and tries to redirect the attacker's browser back to the RP website (step 6 in Sect. 2.2).

The attacker intercepts this redirection, replacing the TIdP-supplied *code* in the authorization response with the stolen *code* for the victim user. It now forwards the modified response to the RP.

The RP next uses the supplied (stolen) *code* to retrieve an access token from the TIdP. The retrieved access token is then used to retrieve the victim user's id. The RP now believes that the attacker is the owner of the victim user's account, and issues a session cookie for this account to the attacker. The attacker is now logged in to the RP as the victim user and can access the victim user's protected resources stored at the RP.

Accessing User Data Stored by the TIdP. Suppose an attacker has the *code* for a particular victim user at the TIdP, and suppose also that the TIdP did not issue a *client_secret* to the RP (this is possible because *client_secret* is an optional parameter in the OAuth 2.0 Authorization Framework). In this case, the attacker uses the *code* to construct an access token request (see step 8 in Sect. 2.2) and sends it to the TIdP. The TIdP, in return, sends an access token for the victim user to the attacker. The attacker can now use this access token to access the victim user's protected resources stored at the TIdP.

5.3 Using an XSS Vulnerability at the RP

This second variant of the PRURIM attack again applies to both the authorization code grant and implicit grant flows. As above, we only present the attack for the authorization code grant flow.

According to the OWASP Top 10 – 2013 report [16], XSS attacks are ranked as the third most critical web application security risk. That implies that it is likely that at least some RP websites contain an XSS vulnerability.

```
// an unauthorised authorization request crafted by the attacker
https://TIdP.com/auth2?
client_id=[client_id_generated_by_TIdP]&
redirect_uri=https://RP.com/XXSVul&
response_type=code
// JavaScripts used to extract the code from the authorization
    response
<script>
var code = document.URL.replace("?", "&");
var src = "http://www.attack.com?RP=" + code;
var img = document.createElement("img");
img.src = src;
document.appendChild(img);
</script>
```

Listing 1.2. The redirect URI manipulate attack

For the purposes of describing this attack we assume that the RP has a XSS vulnerability at https://RP.com/XXSVul which is under the control of the attacker. The attacker first (by some means) causes a victim user to generate an unauthorised authorization request for the target IdP (TIdP) of the form given in Listing 1.2. When it receives this request, the TIdP assumes that it is a normal authorization request generated by the RP, as it only checks the origin part of the *redirect_uri*. It then authenticates the victim user, if necessary (see step 4 in Sect. 2.2), and then generates an authorization response. This response is sent to the URL https://RP.com/XXSVul?code=[code_generated_by_TIdP].

The script (see Listing 1.2) crafted by the attacker at `XXSVul` is assumed to be able to extract the value of https://RP.com/XXSVul?code=[code_generated_by_TIdP]; once it has done this it sends it back to the attacker. The attacker now has the user's *code*, which can now be used to conduct an impersonation attack and/or access user data stored at TIdP, as described in Sects. 5.2 and 5.2.

5.4 Discussion

As noted above, the attack variants described in Sects. 5.2 and 5.3 also apply to the implicit grant flow. Depending on the precise type of attack (and assumptions about the capabilities of the attacker), an attacker is able to obtain varying sets of sensitive values—see Table 1.

The *no state* in the table means that the attack only works if the RP fails to implement CSRF countermeasures at its MIdP sign-in endpoint. This might be made more likely if the MIdP provides sample code without the *state* parameter in the OAuth 2.0 authorization request, or configures the MIdP to not include the *state* in the authorization response before it is sent to the RP.

Table 1. Redirect URI manipulate attacks.

	Authorization code grant		Implicit grant	
PRURIM attacks	Using MIdP	Using XSS	Using MIdP	Using XSS
Attack Assumption	MIdP, web attacker, no state	XSS vul at RP, web attacker	MIdP, web attacker, no state	XSS vul at RP, web attacker
Attackers can get	*access_token*, *code*	*access_token*, *code*	*access_token*	*access_token*

5.5 Relationship to the Prior Art

We conclude this section by describing how the PRURIM attack differs from three somewhat similar attacks described in Sect. 4.

– The **cross social-network request forgery** attack, due to Bansal et al. [1], applies to RPs that use third party libraries, as these RPs use the same login endpoint for all IdPs. By contrast, the PRURIM attack works in situations

where IdPs only check the origin of the *redirect_uri*. While the Bansal et al. attack only works for a special category of RPs, PRURIM attacks apply to all IdPs not strictly checking the *redirect_uri*, and to all RPs using these IdPs.
- In the **Redirection URI Manipulation Attack**, the attacker sets the *redirect_uri* in the authorization request to that of the attacker's own website (e.g. https://www.attacker.com). The key difference between this attack and the PRURIM attacks is that, in a PRURIM attack, the attacker is not required to change the origin of the *redirect_uri*, making it a much greater threat in practice.
- The **IdP mix-up attack** due to Fett et al. [7] works in the context of RPs using explicit user intention tracking to support multiple IdPs; for it to work, a network attack is needed to modify the http or https messages generated by the RP. PRURIM attacks, by contrast, apply to RPs using different *redirect_uri* values to support multiple IdPs. IdP mix-up attacks need a **network attacker** and a MIdP to operate; PRURIM attacks only need a **web attacker** and a MIdP to work, making them a much greater threat in practice.

6 Our Findings

6.1 Summary

We examined the implementations of 27 popular OAuth 2.0 IdPs providing services in English, Russian and Chinese (see Table 2)[11]. Unfortunately, our study revealed that 19 of them (70%) are vulnerable to PRURIM attacks (see Fig. 3). Among the 19 affected IdPs, one is Russian-language, namely mail.ru; four provide services in English, namely Facebook, Microsoft, Instagram and Yahoo; and as many as 14 IdPs are providing services in Chinese, meaning that 88% of the IdPs in China in our study are vulnerable to PRURIM attacks.

6.2 Implications

As described in 3.1, in order to allow the RP to dynamically vary only the query component of the redirection URI when requesting authorization, many IdPs only require an RP to register the URI scheme, authority, and path. For example, iQiyi[12] registers http://passport.iqiyi.com/apis/thirdparty/ncallback.action (together with a varying query component) with every IdP it supports, and it uses the query component in the *redirect_uri* to determine the IdP used (e.g.

[11] Most of the English and Russian language IdPs were chosen from the login page of https://badoo.com/ and https://usatoday.com/. Most of the Chinese-language IdPs were chosen from the login page of http://youku.com, http://www.iqiyi.com and http://ctrip.com.

[12] http://www.iqiyi.com/.

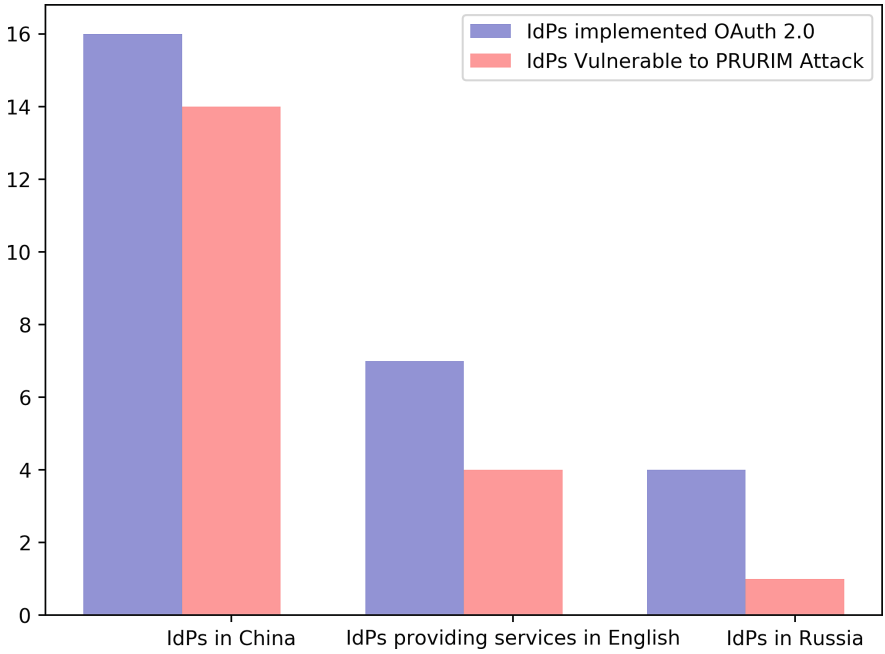

Fig. 3. IdP vulnerabilities by language of site.

http://passport.iqiyi.com/apis/thirdparty/ncallback.action?from=2 is the *redirect_uri* registered with IdP Wangyi, http://passport.iqiyi.com/apis/thirdparty/ncallback.action?from=30 is the *redirect_uri* registered with IdP Xiaomi). This reduces the effort for the RP to manage *redirect_uri* values for multiple IdPs, and gives the RP the ability to customize its OAuth 2.0 sign-in endpoint.

It is interesting to speculate why the standard does not define a single mandatory approach for the IdP to register a *redirect_uri* value with an RP; it seems plausible that this is to give maximum flexibility for RP implementations. As a result, many IdPs allow RPs to register a range of types of *redirect_uri*, and in many cases the IdP only checks the origin part of a *redirect_uri* in an authorization request. This flexibility gives rise to the attacks we have described.

7 Mitigations for PRURIM Attacks

7.1 Impose Strict Redirect URI Checking

PRURIM attacks are made possible if an IdP only checks part of the *redirect_uri*. A simple mitigation for this attack is therefore for the IdP to always check the complete *redirect_uri* using a simple string comparison [15]. However, this can cause problems for those RPs that rely on the origin of the *redirect_uri* to deliver an authorization response. In such cases, the OAuth 2.0 service would stop working if a strict check is always performed.

Table 2. IdPs examined.

	IdP	Vulnerable to PRURIM
1	Amazon	No
2	ebay	No
3	Facebook	Yes
4	Google	No
5	Microsoft	Yes
6	Instagram	Yes
7	Yahoo	Yes
8	mail.ru	Yes
9	OK	No
10	VK	No
11	Yandex	No
12	Baidu	Yes
13	Douban	No
14	Jindong	No
15	Mi	Yes
16	QQ	Yes
17	QQ Weibo	Yes
18	Sina	Yes
19	Taobao	Yes
20	Wangyi	Yes
21	Wechat	Yes
22	anonymised-site-1	Yes
23	anonymised-site-2	Yes
24	anonymised-site-3	Yes
25	anonymised-site-4	Yes
26	anonymised-site-5	Yes
27	anonymised-site-6	Yes

7.2 Implement CSRF Countermeasures

While the main reason that the MIdP-based PRURIM attack is possible is the failure to strictly check the *redirect_uri*, to make the process work the attacker also needs to use a CSRF attack to cause the victim user to visit the site serving the malicious authorization request. This means that the implementation of appropriate CSRF countermeasures by RPs (e.g. including a state value in the authorization request) would help to mitigate the threat of the PRURIM attacks described in Sect. 5.2.

However, in practice, RPs do not always implement CSRF countermeasures in the recommended way. A study conducted by Shernan et al. [19] in 2015 found that 25% of websites in the Alexa Top 10,000 domains using Facebook's OAuth 2.0 service appear vulnerable to CSRF attacks. Further, a 2016 study conducted by Yang et al. [23] revealed that 61% of 405 websites using OAuth 2.0 (chosen from the 500 top-ranked US and Chinese websites) did not implement CSRF countermeasures.

While it is up to the RP to implement CSRF countermeasures, a MIdP can make it less likely that this will happen, e.g. by not including a *state* variable in its sample code, or by not including a *state* value in an authorization response even if it is included in the authorization request.

8 Responsible Disclosure

We reported our findings to all the affected IdPs that provide services in English or Russian. However, reporting our finding to the affected Chinese IdPs was a little more difficult; since 20th July, 2016, China's biggest bug report platform Wooyun[13] has been closed. We reported the problem to the eight IdPs that have set up a security response centre in China; for the other six IdPs affected by the PRURIM attacks, for which we had no obvious way to report our findings, we have simply chosen not to disclose their identities in this paper.

We received positive responses from Yahoo, Microsoft, mail.ru, Sina and Wangyi. These IdPs all stated that they are working on a fix to the PRURIM attack. Facebook also acknowledged our report, but did not commit to making any changes. However, Tencent (the largest Chinese IdP, including QQ IdP, Wechat IdP and QQWeibo IdP) and Baidu both stated that the attack is caused by the RP redirection configuration and do not propose to take any action. Similarly, the response from Xiaomi IdP was "Xiaomi's responsibility of its OAuth 2.0 system is only to authorize user, it is the RP's responsibility to protect the authorization", and thus it seems reasonable to assume that it will not take any action to address the problem. Finally, Taobao IdP (owned by Alibaba) stated that the attacker cannot get the user's code, and hence they do not propose to take any action.

9 Conclusion

In this paper, we described the PRURIM attacks, a new class of attacks against OAuth 2.0. These attacks work against RPs supporting multiple OAuth 2.0 IdPs. We examined 27 IdPs providing services in English, Russian and Chinese. Given the fact that OAuth 2.0 has been widely adopted by IdPs around the world, our study only covers the tip of the iceberg of real-world OAuth 2.0 implementations that are potentially vulnerable to PRURIM attacks.

We have also proposed mitigations for this new attack which can be adopted by IdPs and RPs.

[13] http://www.wooyun.org.

References

1. Bansal, C., Bhargavan, K., Delignat-Lavaud, A., Maffeis, S.: Discovering concrete attacks on website authorization by formal analysis. J. Comput. Secur. **22**(4), 601–657 (2014). https://doi.org/10.3233/JCS-140503
2. Bansal, C., Bhargavan, K., Maffeis, S.: WebSpi and web application models (2011). http://prosecco.gforge.inria.fr/webspi/CSF/
3. Blanchet, B., Smyth, B.: ProVerif: cryptographic protocol verifier in the formal model. http://prosecco.gforge.inria.fr/personal/bblanche/proverif/
4. Chari, S., Jutla, C.S., Roy, A.: Universally composable security analysis of OAuth v2.0. IACR Cryptology ePrint Archive 2011, 526 (2011)
5. Chen, E.Y., Pei, Y., Chen, S., Tian, Y., Kotcher, R., Tague, P.: OAuth demystified for mobile application developers. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 3–7 November 2014, Scottsdale, AZ, USA, pp. 892–903. ACM (2014). https://doi.org/10.1145/2660267.2660323
6. Dill, D.L.: The murphi verification system. In: Alur, R., Henzinger, T.A. (eds.) Computer Aided Verification. LNCS, pp. 390–393. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61474-5
7. Fett, D., Küsters, R., Schmitz, G.: A comprehensive formal security analysis of OAuth 2.0. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 24–28 October 2016, Vienna, Austria, pp. 1204–1215. ACM (2016). https://doi.org/10.1145/2976749.2978385
8. Hardt, D. (ed.): RFC 6749: the OAuth 2.0 authorization framework, October 2012. http://tools.ietf.org/html/rfc6749
9. Jackson, D.: Alloy 4.1 (2010). http://alloy.mit.edu/community/
10. Li, W., Mitchell, C.J.: Security issues in OAuth 2.0 SSO implementations. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 529–541. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13257-0_34
11. Li, W., Mitchell, C.J.: Analysing the security of Google's implementation of OpenID connect. In: Caballero, J., Zurutuza, U., Rodríguez, R.J. (eds.) DIMVA 2016. LNCS, vol. 9721, pp. 357–376. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40667-1_18
12. Li, W., Mitchell, C.J.: Does the IdP mix-up attack really work? (2016). https://infsec.uni-trier.de/download/oauth-workshop-2016/OSW2016_paper_1.pdf
13. Li, W., Mitchell, C.J., Chen, T.: Mitigating CSRF attacks on OAuth 2.0 and OpenID Connect. CoRR abs/1801.07983 (2018). https://arxiv.org/abs/1801.07983
14. Lodderstedt, T., McGloin, M., Hunt, P.: RFC 6819: OAuth 2.0 threat model and security considerations (2013). http://tools.ietf.org/html/rfc6819
15. Masinter, L., Berners-Lee, T., Fielding, R.T.: RFC 3986: uniform resource identifier (URI): Generic syntax (2005). https://www.ietf.org/rfc/rfc3986.txt
16. OWASP Foundation: Owasp top ten project (2013). https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013
17. Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S.: Formal verification of OAuth 2.0 using Alloy framework. In: Proceedings of the International Conference on Communication Systems and Network Technologies, CSNT 2011, pp. 655–659. IEEE (2011)

18. Shehab, M., Mohsen, F.: Securing OAuth implementations in smart phones. In: Bertino, E., Sandhu, R.S., Park, J. (eds.) Fourth ACM Conference on Data and Application Security and Privacy, CODASPY 2014, 03–05 March 2014, San Antonio, TX, USA, pp. 167–170. ACM (2014). https://doi.org/10.1145/2557547.2557588
19. Shernan, E., Carter, H., Tian, D., Traynor, P., Butler, K.: More guidelines than rules: CSRF vulnerabilities from noncompliant OAuth 2.0 implementations. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) DIMVA 2015. LNCS, vol. 9148, pp. 239–260. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20550-2_13
20. Slack, Q., Frostig, R.: Murphi analysis of OAuth 2.0 implicit grant flow (2011). http://www.stanford.edu/class/cs259/WWW11/
21. 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.) the ACM Conference on Computer and Communications Security, CCS 2012, 16–18 October 2012, Raleigh, NC, USA, pp. 378–390. ACM (2012)
22. 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: IEEE Symposium on Security and Privacy, SP 2012, 21–23 May 2012, San Francisco, California, USA, pp. 365–379. IEEE Computer Society (2012)
23. Yang, R., Li, G., Lau, W.C., Zhang, K., Hu, P.: Model-based security testing: An empirical study on OAuth 2.0 implementations. In: Chen, X., Wang, X., Huang, X. (eds.) Proceedings of the 11th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2016, 30 May–3 June 2016, Xi'an, China, pp. 651–662. ACM (2016). https://doi.org/10.1145/2897845.2897874
24. Zhou, Y., Evans, D.: SSOScan: automated testing of web applications for single sign-on vulnerabilities. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, 20–22 August 2014, San Diego, CA, USA, pp. 495–510. USENIX Association (2014). https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/zhou