Research on the Security of OAuth-Based Single Sign-On Service

R. Zhu\(^1\), J. Xiang\(^1\), and D. Zha\(^3\)
\(^1\)Data Assurance and Communication Security Research Center, CAS, Beijing, China
\(^2\)State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
\(^3\)University of Chinese Academy of Sciences, Beijing, China

Abstract—OAuth 2.0 is an open standard for authorization, and provides a method for third-party applications to access users’ resources on the resource servers without sharing their login credentials. It is widely used in Single Sign-On (SSO) service due to its simple implementation and compatibility with a diversity of the third-party applications. It has been proved secure in several formal methods, but some vulnerabilities are exposed in practice. In this paper, we propose a general approach to analyze the security of OAuth-based SSO service. From the perspective of the parameters and flows defined in the protocol, we conduct firstly a careful analysis of its security and design five potential attacks. Then, we examine the typical identity providers (including qq.com, weibo.com, baidu.com and renren.com) and 50 relying parties (such as dianping.com, juemei.com). The results indicate several problems existing in the implementation details, such as the access token leakage. In conclusion, we come up with six recommendations in order to improve the OAuth-based SSO Service.

Keywords: Single Sign-On; OAuth 2.0; Security Detection

1. Introduction

With the development of the Internet, especially the wide use of web 2.0, web applications (such as online shopping, instant messaging and wiki) have become an important part of our lives.

Meanwhile, due to the requirement of user management, most of web applications need the authentication and authority. At present, most applications identify a user by password, which results in the increasing passwords and memory load. Besides, it is tiresome for users to enter the password frequently. In order to improve the situation, Single Sign-On (SSO) service emerges. In the SSO, users are allowed to access several Relying Parties (RP for short, such as dianping.com) while logging into Identity Provider (IdP) for short, such as weibo.com) only once, which is obviously convenient for users.

OAuth 2.0 [1]. OAuth for short, is widely used in SSO service due to its simple implementation and compatibility with a diversity of third-party applications compared with other protocols (such as OpenID [2] and SAML [3]). As an open standard for authorization, OAuth provides a method for third-party applications to access users’ resources on the resource servers without sharing their login credentials.

Although OAuth has been proved secure in several formal methods, the formal analyses were conducted in abstract models but not in the implementations. And several empirical studies have revealed a few vulnerabilities the implementation details, but these empirical studies focused on the case studies, which can not cover all the implementations. Hence, it is still in urgent need of a general approach to detect the security of SSO services.

In this paper, we propose a general approach to analyse the security of OAuth-based SSO service. Firstly, we elaborate the parameters and the flows defined in the protocol, and design five attacks in some presuppositions. Then the approach containing nine detection items is provided to check the presuppositions. Lastly, we conduct an experiment on some typical IdPs and RPs. The results indicate that the approach is available and easy-to-use.

2. Background and Related Work

In OAuth-based SSO service, a resource server is viewed as an IdP to manage users’ identities and to identify the user. While a third-party application is viewed as a RP to serve users, and authenticate the user by user’s profile gotten from IdP. In brief, the user’s profile hosted on IdP is authorized and shared for RP to identify the user. Note that IdP and RP are relative concepts, namely, one could be either IdP or RP in different scenarios.

2.1 Authentication Flows in OAuth

OAuth-based SSO service is implemented by browser redirection. When a user logs into RP with an account on IdP, RP requests IdP to authenticate the user by redirecting user’s browser to IdP. And IdP identifies the user before returning the authentication result to RP. During the identification, if the user authorizes RP to access his/her resource, IdP will issue RP an access token which is used to make IdP’s API calls for accessing or handling user’s resource hosted on IdP. In other words, RP authenticates the user with the user’s profile.

Two SSO flows are defined in OAuth: the server flow (the "Authorization Code Grant" in the specification) and the client flow (the "Implicit Grant"). They are identified by the parameter response_type. For the server flow with response_type = code, illustrated in Fig. 1, RP gets an access token from IdP directly by presenting the shared secret with IdP.
C0: **RP Register**: RP submits its profile including a redirection URI `redirect_uri` to IdP for register, and receives a client ID `client_id` and a shared secret `client_secret` from IdP.

C1: **Login Request**: Browser (or a user) sends a HTTP request to RP after the user clicks the login button.

C2: **Authentication Request**: RP responds a redirection message with the parameters to IdP including `response_type=code`, `client_id`, `redirect_uri` and an optional parameter `state`.

C3: **Identity Authentication**: After checking `client_id` and `redirect_uri` against its own local storage, IdP responds a login page to identify the user. And then an authentication session is created, namely session S. This step could be omitted if the session has been created.

C4: **Issuing Authorization Code**: IdP responds a redirection message with authorization code `code` and `state` (if presented) to the server identified by `redirect_uri`.

C5: **Access Token Request**: RP sends an access token request with the parameters including `code`, `redirect_uri`, `client_id` and `client_secret`.

C6: **Access Token Response**: After validating these parameters, IdP responds an access token `access_token` to RP.

C7: **User’s Profile Request**: RP makes an IdP’s API call with `access_token`.

C8: **User’s Profile Response**: IdP validates `access_token` and returns user’s profile to RP.

C9: **Login Response**: RP creates an authentication session.

For the client flow with response_type = token, IdP sends directly an access token to user’s browser after identifying the user. RP gets the access token from the browser using a script (e.g. a javascript file). Fig. 2 illustrates the client flow.

T0: **RP Register**: RP submits its profile including a redirect URI `redirect_uri` to IdP for register, but receives only a client ID `client_id` from IdP.

T1: **Login Request**: The same as C1.

T2: **Authentication Request**: The same as C2 except `response_type = token`.

T3: **Identity Authentication**: The same as C3.

T4: **Issuing Access Token**: IdP responds a redirection message with `access_token` appended as an URI fragment of `redirect_uri`. Note that the request to RP does not contain `access_token` in the URI fragment.

T5: **Extracting Access Token**: RP responds a script to extract `access_token` in the fragment.

T6: **Submitting Access Token**: The extracted `access_token` is submitted to RP by the script automatically.

T7-T9: The same as C7-C9 respectively.

### 2.2 Comparison of the Flows

In the both flows, the browser creates authentication sessions with both IdP and RP respectively in step C3/T3 and C9/T9. As long as possessing an access token, RP is able to make web API calls to access or handle the user’s resource in step C7 or T7.

IdP could validate RP with the shared secret in the server flow, while IdP sends an access token appended as an URI fragment to user’s browser without authenticating RP in the client flow, thus the latter is more vulnerable. What is worse, the user can hardly tell the difference of the both flows with the naked eye.

### 2.3 Related Work

In theory, several formal approaches have been used to examine the OAuth, such as Alloy framework used by Pai et al. [4], universally composable security framework used by Chari et al. [5] and Muiphi used by Slack et al. [6], and all the results were included in the official OAuth security guide [7]. Thus the implementation following the guide is secure.
in theory. On empirical analysis, Wang et al. [8] focused on the actual web traffic going through the browser, and discovered several logic flaws in some SSO services (e.g. Google ID, Facebook), which are used by attackers to tamper with the authentication messages. Sun et al. [9] examined the implementation details of three major OAuth-based IdPs including Facebook, Microsoft and Google, and uncovered several vulnerabilities that allowed attackers to gain access to the user’s profile and to impersonate the user on RP.

Actually, all the existing approaches are deficient. The formal analyses were conducted in the abstract models, and some implementation details could be left out, which has been proved by the empirical studies; while the existing empirical studies exposed the vulnerabilities in the case analyses, which could omit some proofs.

In OAuth 2.0, the cryptography is taken out, and its transport security depends on the protocol SSL/TLS or HTTPS. However, HTTPS protection has no effect on the messages hosted on the end-points (issuer, browser and receiver). Meanwhile, session management is a vital task during the authentication, especially the undetectable session \(S\) (cf. step C3 or T3). In this paper, we focus on the overall security of OAuth including transport security, endpoint security and session security, and propose a general approach to detect the security of SSO. On protocol analysis, we examine the flows and five variable parameters, and summarize their usages, requirements and potential risks. On empirical study, 5 attacks based on protocol analysis are proposed, and 10 IdPs (including qq.com, weibo.com, renren.com and baidu.com) and 136 login flows using by 50 RPs (such as dianping.com, jumei.com etc.) are checked to validate the availability of the approach.

3. Methodology

3.1 Attack Model

It is assumed that 1) use’s browser and computer are invulnerable, 2) both RP and IdP are not malicious, and 3) the direct communications between RP and IdP are secure. However, due to the universality of web vulnerabilities [10], IdP or RP may have some vulnerabilities, such as Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF). The purpose of an attacker is to gain an unauthorized access to user’s resources hosted on IdP or RP. In brief, the abilities of the attacker are limited as follows:

- The attacker is a web user who could set up a website, issue some links through the blog, microblog and email, and even launch an attack to a vulnerable website.
- The attacker is also a sniffer who could sniff the unencrypted traffic in the Internet, and tamper simply with the traffic.

3.2 Proposed Approach to Detect the Security of OAuth

Our analytic framework is to provide a general approach to detect the security of OAuth-based SSO service. It is a two-stage process including protocol analysis and empirical analysis, as illuminated in Fig. 3.

In the stage of protocol analysis, we review the protocol carefully to reveal the usages, requirements and potential risks of the five variable parameters and session \(S\) (\(X_1-X_5\) in Fig. 1 and Fig. 2).

Five attacks (identified by \(A_1\) to \(A_5\)) based on the former are provided in the latter stage, all of which have been proved available in the following experiment. And the analysis on the presuppositions of the five attacks reveals that we need only to execute an detection on item I1 to I9 as follows to evaluate the security of an SSO service:

11: Whether HTTPS protection is deployed by RP.
12: Whether an unpredictable state parameter is used by RP.
13: Whether RP is prevented against CSRF attack.
14: Whether RP stores the access token in the cookie or URI.
15: Whether the code is cross in use.
16: Whether IdP supports the two response types simultaneously and indiscriminately.
17: Whether any redirection URI in the realm of RP could pass IdP’s checking.
18: Whether the token is a bearer token.
19: Whether a mechanism to end the session \(S\) is provided.

Based on the analytic framework, the proposed approach can be described with the pseudo-code as follows:

```plaintext
// Parameters X[1-5] and S are defined in Section 3.3;
// Attacks A[1-5] are defined in Section 3.4;
// The symbol * means the attack has greater damage.
AttackList attacks = NULL; // all the potential attacks.

if(CHECK(X4.I1: HTTPS is employed by RP) = TRUE) {
    // Parameters X{1-5} and S are defined in Section 3.3;
    // Attacks A[1-5] are defined in Section 3.4;
    // The symbol * means the attack has greater damage.
    AttackList attacks = NULL; // all the potential attacks.
    AttackList Detection() {
        // Parameters X[1-5] and S are defined in Section 3.3;
        // Attacks A[1-5] are defined in Section 3.4;
        // The symbol * means the attack has greater damage.
        AttackList attacks = NULL; // all the potential attacks.
    }
```
if(CHECK(X4.I5: The code is cross in use) = TRUE)
attacks.Add(A1*);
else
attacks.Add(A1);
}
if(CHECK(X1.I6: Two response types are supported by IdP) = TRUE AND
CHECK(X5.I8: The token is a bearer token) = TRUE) {
if(CHECK(X2.I7: The realm URIs are checked out by IdP) = TRUE)
attacks.Add(A2*);
else
attacks.Add(A2);
}
if(CHECK(X5.I4: The access token is stored incorrectly by RP) = TRUE AND
CHECK(X5.I8: The token is a bearer token) = TRUE) {
if(CHECK(X3.I2: HTTPS is employed by RP) = FALSE)
attacks.Add(A3*);
else
attacks.Add(A3);
}
if(CHECK(X3.I3: Protection against CSRF is employed by RP) = FALSE)
attacks.Add(A4);
// check Item X3.I2': Whether state is invalid.
if(CHECK(X3.I2: The state parameter is used by RP) = TRUE)
Warning(The state parameter (X3) is INVALID);
if(CHECK(S.I9: Ending session S is provided by RP) = FALSE)
attacks.Add(A5);
return attacks;

3.3 Protocol Analysis on Parameters and Session

In the authentication flows, five variable parameters are defined, and two authentication sessions are created. In the rest part of this section, we discuss the usages, requirements and potential risks of the parameters and the session S.

X1: Response Type (response_type)
Usage: It is set by RP and used to select which flow to be used in the following sequence, that is, to decide the way for RP to gain an access token.
Risk: As mentioned in 2.2, the both flows are similar in user’s view, so attackers could set response_type = token, and gain the access token through a script without user’s awareness.

X2: Redirection URI (redirect_uri)
Usage: It is set by RP and used to decide the destination of request C6 or T6, which receives the code or access token.
Requirement: IdP should check it strictly to avoid sending the code or access token to other receivers except RP.
Risk: The receiver, besides an attacker, of the code or access token could log into RP as the victim, and gain the victim’s profile.

X3: State Parameter (state)
Usage: It is generated by RP for tracing the session between browser and RP to prevent against CSRF attack.
Requirement: It should be an unpredictable value bound to the session with RP.
Risk: It is an optional parameter. If it is lost, another countermeasure against CSRF attack MUST be taken specified in the protocol. In practice, CSRF attack is often neglected by developers.

X4: Authorization Code (code)
Usage: It is sent to RP in the server flow and used for RP to request an access token from IdP.
Requirement: It MUST be an expiring and one-time token, and be transported in a channel with HTTPS protection specified in the protocol.
Risk: If gaining it and using it before the victim, the attacker could impersonate the victim and log into RP, which has a greater risk if the code could be used for multiple RPs, e.g. the code sent to RP1 could be used to log into RP2.

X5: Access Token (access_token)
Usage: It is generated by IdP and used for RP to make web API calls to gain or handle user’s profile. RP use the received profile to authenticate the user.
Requirement: It should be an unpredictable value and not be sent to the others except RP.
Risk: Due to the decryptographic design of OAuth, the access token is often implemented as a bearer token. That is to say, any bearer of the token, e.g. an attacker, could access or handle user’s profile.

S: Session with IdP
Usage: It is created by IdP in step C3 or T3 and used to authenticate the user for the latter login.
Requirement: It is often maintained through the cookie technology.
Risk: After created, the session keeps alive and is only bound to cookies invisible to the user, so other mechanisms should be employed to clear the session.

In conclusion, the misuse of each of X1 to X5 and S could result in unsecure factors, thus understanding the strict requirements is a precondition for implementing an invulnerable OAuth-based SSO service.

3.4 Empirical Analysis

The former results reveal that the overall security requires competent developers to implement the SSO service. In practice, the simplicity of OAuth misleads the developers, and they often fail to make the implementations meet the requirements. Thus in this section, according to all the requirements, 10 IdPs and 136 authentications used by 50 RPs are examined, and the results indicate that risks lay in the carefullness though the OAuth-based SSO service is widely used.
A1: Stealing and Embezzling Authorization Code

In the server flow, HTTPS protection is used for all the communications with IdP, but not for ones submitting the authorization code to 90% of RPs (cf. Table 2) in step C4. Thus an attacker is able to steal the authorization code and makes the victim fail to submit it (e.g. by modifying the authorization code in the traffic). It allows the attacker to log into RP as the victim via the code.

Cross use of the code is not detected in our study, but it is reported that the authorization code generated by weibo.com for RP1 is allowed to be used to log into RP2 [11]. In this situation, the risk is of greater damage.

Interestingly, 30% of RPs protect their own login page with HTTPS, but only 9.56% of RPs provide HTTPS protection for the authorization code.

A2: Stealing Access Token via XSS

All the detected IdPs maintain the both flows and do not limit their RPs to select which flow to use. In this way, XSS vulnerability in redirect_uri page on RPs using the client flow, as Sun et al.[9] put it, allows attackers to launch the client-flow sequence and to exact the access token via script in step T4.

Meanwhile, for RP with XSS vulnerability, the attacker could construct a link with response_type = token and trick victims to click it, and the similar way could be used to gain the access tokens after victims enter their passwords.

What is worse, most of IdPs only check whether the redirection URI is in the realm of RP, which causes that this attack could be launched in case any page on RP has XSS vulnerability, that is to say, the attack surface is broadened.

A3: Eavesdropping or Stealing Access Token

The bearer token is deployed in all the detected IdPs. It is the only identity for RP to access user’s profile, that is to say, the one who possesses the token could be viewed as the right RP to access user’s profile. So RP is bound to guarantee its confidentiality.

In fact, a few RPs (nearly 9%, cf. Table 2) still keep the access token in the cookie or URI without any protections, hence it is likely to be stolen from the browser, e.g. by using script to read it from cookie without httponly attributes, or by examining the history of the browser. For RPs without HTTPS protection, even an attacker could eavesdrop on the open traffic to get the access tokens.

A4: CSRF Attack

It is specified in OAuth that CSRF attack, ranking 8 in Top-10 List issued by OWASP [10], must be handled. And the state parameter is recommended to handle it. Without a protection, an attacker could launch the attack as follows:

(a) The attacker starts a server-flow sequence with his/her own account, but stops before step C4 and saves the request URI named csrf_uri.
(b) csrf_uri is issued in the Internet via blog or microblog, and a victim clicking the csrf_uri will log into RP as the attacker.
(c) Due to the victim logs in as the attacker, the attacker may be able to record victim’s information.

On the surface, the authorization code, after all, is for one-time use, so CSRF attack is inefficient and negligible. Nevertheless, binding with a local RP account is required in some RPs (e.g. dianping.com), so the victim may bind one’s own RP account to the IdP account (or the attacker’s account), in other words, the attacker may control victim’s RP account via the IdP account.

The results manifest that only 44.12% of RPs take CSRF attack into consideration, and 39.71% adapt the recommendation using the state parameter. It is worth noting that a small part of RPs set the state parameter as a fixed value, and that about 25% of RPs do not check the parameter strictly and accept the request without the parameter after step C4, all of which fall flat against CSRF attack.

A5: Risk of the Implicit Session with IdP

As the session S (cf. Fig. 1 and Fig. 2) lays in the browser implicitly, the user can’t end it by closing some page after login, which may initiate the following risk:

The user logs into dianping.com with the account on renren.com on a common computer, as shown in Fig. 4, then logs out of dianping.com but not renren.com when leaving, that is, the session with renren.com is alive. The latter user on the same computer can log in to dianping.com, even other RPs, e.g. jumei.com, as the identity of the former user by just clicking a button.

For this reason, logging out of the IdP is a recommendation for a complete solution to refrain from the aforementioned risk. Nevertheless, none of the detected RPs maintains a mechanism to end session S though parts of IdPs offer an interface to log out.
4. Experiment

4.1 Experimental Environment

An investigation to a portal web site containing 211 links shows that there are 196 RPs using the SSO service and 28 IdPs providing it, and the top four of the most popular IdPs are qq.com, weibo.com, baidu.com and renren.com.

In our experiment, we analyze 136 authentication sequences used by 50 RPs with the four IdPs, and 10 IdPs using OAuth.

4.2 Methods to Detect the Items

The 9 items could be divided into two categories: static items, including I1, I2 and I4, which could be checked by going through the communications in the sequences for "https", "state", "access_token" and other related words; and dynamic items, including the rest of the items, which could only be checked by constructing or tampering with HTTP requests. Therefore, our detection contains three steps:

1) **Data Collection**: Log into RPs with an account on IdPs using Firefox, and record all the communications with Live HTTP Header plug-in.
2) **Data Analysis**: Depict the actual login sequence according to the communications, then review whether HTTPS is deployed by RP to submit the code, whether the state parameter is contained in the authentication request (I2), especially whether the state parameter is a fixed value, and whether the words, such as "access_token", "access_token" or other similar words, are contained in the cookie or URI (I4).
3) **Dynamic Detection**: Construct special requests conforming with the actual sequence to conduct the dynamic detection. Each detection methods are listed in Table 1.

4.3 Results

4.3.1 Result of Detection Items on RPs

50 RPs are checked on item I1 to item I4 and item I9 in the RP detection. The result is demonstrated in Table 2, which indicates 44 RPs with qq.com, 43 RPs with weibo.com, 25 RPs with baidu.com and 24 RPs with renren.com. We also find that 15 in 50 RPs (30%) deploy the HTTPS protection for their own login pages.

### Table 1: Detection Methods of Each Dynamic Items.

<table>
<thead>
<tr>
<th>Item</th>
<th>Detection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Log into RP using Firefox.</td>
</tr>
<tr>
<td>(2)</td>
<td>Stop in step C4, and save the request URI.</td>
</tr>
<tr>
<td>(3)</td>
<td>Send the request URI using IE.</td>
</tr>
<tr>
<td>(4)</td>
<td>Conclusion: If login succeeds using IE, NO protection against CSRF is deployed.</td>
</tr>
<tr>
<td>(5)</td>
<td>Repeat step 1 and 2.</td>
</tr>
<tr>
<td>(6)</td>
<td>Send the request URI using IE without the state parameter.</td>
</tr>
<tr>
<td>(7)</td>
<td>Conclusion: If login succeeds, the state parameter is INVALID.</td>
</tr>
</tbody>
</table>

### Table 2: Result of Detection Items on RPs

<table>
<thead>
<tr>
<th>IdP</th>
<th>OAuth</th>
<th>HTTPS(I1)</th>
<th>State(I2)</th>
<th>Invalid State(I2')</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>qq</td>
<td>44</td>
<td>9.09%</td>
<td>59.09%</td>
<td>46.15%</td>
<td>136</td>
</tr>
<tr>
<td>weibo</td>
<td>43</td>
<td>9.30%</td>
<td>25.58%</td>
<td>9.09%</td>
<td></td>
</tr>
<tr>
<td>baidu</td>
<td>25</td>
<td>8.00%</td>
<td>52.00%</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>renren</td>
<td>24</td>
<td>12.50%</td>
<td>16.67%</td>
<td>0.00%</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3: Result of Detection Items on IdPs

<table>
<thead>
<tr>
<th>Total</th>
<th>Code(I5)</th>
<th>response_type(I6)</th>
<th>redirect_uri(I7)</th>
<th>Bearer(I8)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

1) Code(I5): The count of IdPs using the cross-use code;
2) Response type(I6): The count supporting the two response types simultaneously;
3) Redirection type(I7): The count s without checking the URI strictly;
4) Bearer(I8): The count using bearer token.
of our expectation, except item I6, all the others could be used by attacker to launch an attack.

5. Conclusion

In this paper, we introduce briefly two flows defined in OAuth. And based on the analysis of the parameters and session management, five attacks are carried out. Above all, a general approach to detect the security of OAuth-based SSO service is provided, and detections for both RPs and IdPs are conducted. Nevertheless, an automatic detection tool is unavailable, which will be the next focus.

Meanwhile, in order to protect the users’ accounts, the security of the authentication is vital. However, the aforementioned analyses reveal that the implementations are vulnerable. As a result, several recommendations are proposed in order to improve the SSO service:

1) The confidentiality of the communications must be brought to the forefront. On one hand, HTTPS protection could be deployed. On the other hand, an alternative could be provided, e.g. during registration, a shared secret could be created between IdP and RP to encrypt the messages.

2) The protection against CSRF attack must be employed by RP. The state parameter must work if used.

3) Both RP and IdP should store the access token in a secure way, especially not expose it in the cookie or URI.

4) RP registers the response type and a specific redirection URI on IdP, while IdP validates the parameters strictly.

5) An interface should be provided by IdP to end the authentication session S, and RP should call the API when users log out.

6) The authorization code generated by IdP should be specialized to the right RP.

6. Acknowledgement

This work was supported by National Natural Science Foundation of China grant 70890084/G021102 and 61003274, Strategy Pilot Project of Chinese Academy of Sciences sub-project XDA06010702, and National High Technology Research and Development Program of China (863 Program, No. 2013AA01A214 and 2012AA013104).

References