Real-world security analyses of OAuth 2.0 and OpenID Connect

Wanpeng Li and Chris J Mitchell

Agenda

- Single sign-on and identity management
- OAuth 2.0
- Two case studies
- Security analyses
- OpenID Connect
- Security of Google SSO
- Concluding remarks
Single sign on (SSO)

• An Internet single sign on (SSO) system allows a user to log in to multiple web sites with just one authentication.

• Increasingly widely used, e.g. in form of
  – Facebook Connect (OAuth 2.0);
  – Google SSO service (formerly built using OpenID and now employing OpenID Connect).

Identity management

• An SSO system is just a special case of an identity management system.

• In general, in an ID management system, one or more third parties manage aspects of a user’s identity on behalf of a user, e.g. they
  – store user attributes;
  – authenticate users on behalf of other parties.
Identity management terminology

- **Identity Provider (IdP)** authenticates user and vouches for **User** identity to ...
- **Relying Parties (RPs)**, which rely on IdP and provide online services to ...
- **Users**, who employ ...
- **User Agents (UAs)** (typically web browsers), to interact with RPs.

Federation

- **Federation** is an important notion in many real-world identity management systems.
- Enables two entities to link (federate) their respective identities for a single user.
- Enables identity management functionality, since allows parties to exchange information about a user.
- Federation process needs to be secure!
Information Security Group

Agenda

• Single sign-on and identity management
• OAuth 2.0
• Two case studies
• Security analyses
• OpenID Connect
• Security of Google SSO
• Concluding remarks

Information Security Group

OAuth 2.0

• OAuth 2.0, published in 2012 (RFC 6819), is being widely used as the basis of SSO services, e.g. for Facebook Connect.
• It is also being very widely used for SSO by a range of popular IdPs in China.
• Issues with use of OAuth 2.0 by Facebook and others have already been identified.
• This motivated study of security of Chinese implementations.
OAuth design goals

• Original goal of OAuth (1.0 & 2.0) not SSO.
• OAuth allows a Client application to access information (belonging to a Resource Owner) held by a Resource Server, without knowing the Resource Owner’s credentials.
• Also requires an Authorization Server, which, after authenticating the Resource Owner, issues an access token to the Client, which sends it to the Resource Server to get access.

Use for SSO

• When used to support SSO:
  – IdP = Resource Server (stores user attributes) + Authorization Server (authenticates user);
  – RP = Client;
  – User = Resource Owner (owns user attributes);
  – UA = web browser.
• Access token used to provide SSO service (not really what it was intended for).
• OAuth supports four ways for a Client to get an access token.
• Of these, we focus on Authorization Code Grant.
OAuth 2.0/SSO – data flows

1. User clicks button on RP website, and UA sends HTTP request to RP.
2. RP sends OAuth 2.0 authorization request to UA, optionally including state variable (used to maintain state between request and response).
3. UA redirects request to IdP.
4. If necessary, IdP authenticates User.
5. IdP generates authorization response containing code (an authorization code), and the state value, and sends it to UA.
6. UA redirects response to RP.
7. RP sends access token request to IdP (directly) containing code and client_secret (shared by IdP and RP).
8. IdP checks request values and responds to RP with access token.
9. RP uses access token to retrieve user attributes (specifically the IdP user identifier) from IdP.

OAuth 2.0 – identity federation

- OAuth 2.0 specifications do not provide a standardised approach to identity federation.
- Not surprising given OAuth 2.0 not really designed for SSO.
- Commonly used (ad hoc) means of federation involves the RP binding the user-RP account to the user-IdP account, using the unique user ID generated by the IdP.
- The IdP account ID is fetched from the IdP in step 9 of previous slide.
OAuth 2.0 – identity federation II

• After receiving the access token (step 8), RP retrieves the user-IdP account ID.
• RP then binds user-RP account ID to user-IdP account ID.
• One method of achieving binding is:
  – user initiates binding after logging in to RP;
  – user required to log in to IdP;
  – user grants permission for binding;
  – RP completes binding process.

Wide use

• In the relatively short time since OAuth 2.0 specifications published, it has become widely used as basis for SSO (e.g. by Facebook).
• Particularly big uptake in China:
  – some Chinese language RPs support as many as eight (OAuth-based) IdPs;
  – at least ten major websites offer OAuth 2.0-based IdP services.
Known issues

• OAuth 2.0 has been critically examined by a number of authors.
  – Frostig & Slack (2011) found a Cross-Site Request Forgery (XSRF) attack in the Implicit Grant flow of OAuth 2.0.
  – Wang, Chen & Wang (2012) found a logic flaw in a range of SSO implementations.
  – Sun & Beznosov (2012) found flaws in OAuth 2.0 implementations.

• However, no published studies of real-life security of Chinese-language sites, despite large numbers and wide use of OAuth 2.0.

Attack countermeasures

• OAuth 2.0 specifications recommend use of state parameter in authorization request & response to protect against CSRF attacks.
• For it to work state must be non-guessable.
• Otherwise attacker could include guessed value in a CSRF-generated fraudulent authorization response.
• We observed that many real-world RPs either omit the state or use it incorrectly.
Scope of attacks

• New attacks we have discovered are more powerful than previously known attacks.

• Attacks using CSRFs enable false identity federations, i.e. binding attacker’s IdP identity to victim’s RP identity.

• After such a federation, an attacker can log in at will to victim accounts.

• Attacks do not require victim cooperation (except to visit a malicious website at some point prior to attempting a federation).

Agenda

• Single sign-on and identity management
• OAuth 2.0
• Two case studies
• Security analyses
• OpenID Connect
• Security of Google SSO
• Concluding remarks
General approach

- Investigated properties of range of real-world implementations of OAuth 2.0-based SSO.
- Looked at browser-relayed messages (BRMs) between RPs and IdPs.
- Used Fiddler (open source tool) to capture BRMs, and developed Java parser for BRMs.
- Focussed on attacks on the identity federation ‘binding’ process.

Scope of study

- We looked at 60 Chinese RPs supporting federation-based SSO using OAuth 2.0.
- Of these 60, 14 did not support the vulnerable binding method.
- Of the remaining 46, a total of 21 (i.e. nearly half) were found to be vulnerable to CSRF-based false binding attacks.
Renren Network

- Renren is a social networking site with 320 million users – the ‘Facebook of China’.
- It supports several OAuth 2.0-based IdPs for SSO, including Baidu and China Mobile (both major sites).
- We examined federation interactions between Renren and both Baidu and China Mobile.

Ctrip

- Ctrip is a China-focused travel agency with 60 million members.
- Ctrip supports eight OAuth 2.0-based SSO IdPs, including Renren, Wangyi, Taobao, MSN and Sina.
- We looked at federation interactions between Ctrip and Renren.
Agenda

• Single sign-on and identity management
• OAuth 2.0
• Two case studies
• Security analyses
• OpenID Connect
• Security of Google SSO
• Concluding remarks

Renren-Baidu binding attack I

• Suppose user logged in to Renren (RP) wants to bind Renren account to Baidu (IdP) account.
• Renren generates an auth request and redirects UA (user browser) to Baidu.
• Renren does not include state in auth request, i.e. no means of binding the auth request to the subsequent auth response.
• After authenticating the user, Baidu returns an auth response containing a code (via the UA) – the UA adds cookies containing session ID.
• Renren uses the code to get access token from Baidu, and then uses the access token to retrieve the Baidu account ID.
• Finally Renren binds its account ID to the Baidu account ID.
Renren-Baidu binding attack II

• Because no state value, attacker could replace the code in the auth response with a code generated by Baidu for a separate attacker-initiated interaction.
• Then the user ID that Renren later retrieves from Baidu will be attacker’s ID not the user’s ID.
• This means Renren will bind the attacker’s Baidu ID with the user’s Renren ID.
• Catastrophe!
• We tested this using a CSRF approach to perform the substitution, and it worked.

Renren-China Mobile binding attack

• In this case, both auth request and auth response contain a state value.
• However, state value is the same for multiple requests and responses (always ‘9’).
• Thus an attack almost identical to the Renren-Baidu attack works, enabling binding of attacker’s China Mobile account to victim’s Renren account.
Generic Ctrip binding attack I

- Looked at Renren-Ctrip binding process (Renren acting as IdP not RP as in previous case).
- No state value in auth request.
- However, code substitution attack did not work (not sure why).
- We observed that the initial HTTP request contained a Uid (a Ctrip-generated user ID).
- We speculated that if we replaced the Uid in an attacker-generated request with a victim’s Uid, then it might be possible to force Ctrip to bind the attacker’s IdP account to the victim’s Ctrip account.

Generic Ctrip binding attack II

- We tried it and it worked!
- We analysed this further, and found it would work with many IdPs working with Ctrip.
- The Ctrip implementation contained logic flaws.
- Getting Uid values for victims is simple using the Ctrip user forum.
- In all our attacks we used specially created accounts (no ‘real’ accounts were hacked).
Disclosures

• We notified all the affected RPs and IdPs several months before publication of our results.
• We got a mixed response – most major sites fixed the problems and thanked us.
• However, some sites denied that our attacks were a problem ...

Reasons for problems

• Perhaps the single most important reason that these attacks arise is because of the lack of standards for OAuth 2.0-based SSO and identity federation.
• This is now partly addressed by OpenID Connect, which builds a standardised identity layer on top of OAuth 2.0.
• However, problems remain as discussed in next part of talk!
Recommendations

- In absence of clear standards, guidance from IdPs critical.
- Some IdPs did not clarify use of state, and did not even include state in their sample code.
- Consequences of not using state value were not made clear to RPs.
- Have published detailed list of recommendations for IdPs and RPs.

Agenda

- Single sign-on and identity management
- OAuth 2.0
- Two case studies
- Security analyses
- OpenID Connect
- Security of Google SSO
- Concluding remarks
Building on OAuth 2.0

- OpenID Connect 1.0 is built as an identity layer on top of OAuth 2.0.
- Adds extra functionality aimed specifically at SSO.
- Adds a new type of token to OAuth 2.0, namely the id token [a JSON web token].
- The id token contains claims about authentication of end user – generated by entity known as OpenID Provider (OP) [=IdP].
- It is digitally signed by the OP.

Four ways to retrieve an id token

- OAuth (and hence OpenID Connect) supports four ways for a Client (the RP) to retrieve a token from the Authorization Server (IdP):
  - hybrid flow [token sent via the UA, using an RP-provided JavaScript client running on UA];
  - client-side flow [very similar to hybrid flow];
  - authorization code flow [token sent directly from authorization server (IdP) to client (RP)];
  - pure server-side flow [not supported by Google].
Agenda

- Single sign-on and identity management
- OAuth 2.0
- Two case studies
- Security analyses
- OpenID Connect
- Security review of Google SSO
- Concluding remarks

A large study

- We looked at the GTMetrix top 1000 websites providing an English language service.
- Of these, 103 support Google’s SSO service based on OpenID Connect.
- We examined all 103 in detail.
- As in OAuth study, we use Fiddler to capture browser-relayed messages, and developed a Python program to analyse these messages.
- No third party accounts were hacked.
Retrieving the *id token*

- As mentioned, OpenID Connect supports four ways for a Client (the RP) to retrieve a token from the Authorization Server (IdP).
- Of the 103 websites we examined:
  - 69 use the authorization code flow;
  - 33 use the hybrid flow;
  - just one uses the client-side flow.
- Look further at the two main cases.

Hybrid server-side flow

- We identified a wide range of serious vulnerabilities in many of the 33 RP sites implementing this approach.
- We next summarise some of the main issues we have identified.
Issue 1: Authentication by Google ID

- Three of the 33 do not use the *id token* or the *access token* for authentication.
- If the UA submits the appropriate Google ID to the RP, then the RP will treat the user as authenticated!
- The Google ID for a user is relatively easy to determine.
- We notified the three affected RPs – one fixed the problem, one withdrew support for Google SSO, and the other appeared to ignore our advice.

Issue 2: Using the wrong token

- As many as 15 of the 33 RPs base their authentication of the user on the *access token* and not the *id token*.
- Moreover, 13 of the 15 do not verify the *access token* before using it.
- Hence a malicious/fake RP could use a stolen *access token* to impersonate a user to any of these 13 sites.
- Unfortunately, a malicious RP can routinely obtain *access tokens* from the Google server.
Issue 3: Intercepting an *access token*

- Four of the 33 RPs arrange for an *access token* to be sent from the UA to the RP in cleartext.
- This contravenes the OAuth specifications.
- A passive interceptor, e.g. someone monitoring an unencrypted Wi-Fi network, could thus intercept the token.
- This has potentially serious consequences, given that some sites use the *access token* for authentication.

Issue 4: Privacy threats

- Intercepting an *access token* or an *id token* has potential privacy implications, since they both encode user attributes.
- As many as seven of the 33 RPs potentially leak a token (to a passive eavesdropper) through lack of SSL protection.
Issue 5: Session swapping

- The OpenID specifications recommend inclusion of a *state* value when JavaScript client on UA sends tokens back to the RP, where *state* is bound to browser session.
- This prevents session-swapping attacks.
- 24 out of the 33 RPs do not use a *state* value, or use it incorrectly, and are hence vulnerable!

Analysis

- Many of the problems arise because of incorrect implementation by the RPs.
- Many of the RPs have customised the hybrid flow to maximise efficiency at the cost of security.
- The problems with the *state* value arise partly because Google does not use the value properly in its sample code provided to RP developers.
- We believe Google could do much more to limit possibility of RP implementation errors.
Authorization Code flow

- The authorization code flow (used by 69 of 103 RPs) is inherently more secure than the hybrid flow.
- The tokens never pass through the UA, and hence are not at risk from malware running on the user machine.
- However, we still identified a range of security issues.

Authorization code flow issues

- Issues identified include:
  - sending an access token over an non-SSL protected link (4 out of 69);
  - stealing an access token using a common XSS vulnerability (possible for all 69);
  - sending user information unprotected across a link (11 out of 69);
  - session-swapping vulnerability (24 of 69);
  - CSRF-based forced login (24 of 69).
Disclosures

- As well as notifying the most seriously affected RPs, we also notified Google.
- This all occurred several months ago.
- Google have acknowledged receipt of our work, but have not commented further.

Recommendations

- RPs:
  - do not customise the hybrid flow;
  - deploy anti-CSRF countermeasures (*state* value);
  - use changing and secret *state* values.
- Google (& other OpenID Connect Providers):
  - don’t send *access tokens* – just send *id tokens*;
  - add a *state* value to the sample code;
  - improve handling of the *state* value.
Agenda

• Single sign-on and identity management
• OAuth 2.0
• Two case studies
• Security analyses
• OpenID Connect
• Security of Google SSO
• Concluding remarks

Common problems

• There seem to be two common threads in the problems with have identified with both OAuth 2.0 and OpenID Connect implementations:
  – RPs have difficulty in properly implementing the protocol, both at the RP server and in their JavaScript downloaded to UAs;
  – IdPs do not always provide the clearest advice, and sample code is sometimes less than ideal.
References