Vetting Single Sign-On SDK Implementations via Symbolic Reasoning

Ronghai Yang¹,², Wing Cheong Lau¹, Jiongyi Chen¹, Kehuan Zhang¹

¹The Chinese University of Hong Kong and ²Sangfor Technologies Inc.
Strong Adoption of Single Sign-On Services

- Janrain Report: 75% of users prefer SSO and 91% of them are satisfied
- 405 out of Top-1000 web applications support SSO services [1]
- 1372 out of 4151 Android apps support SSO services [2]

Basic Interactions of Single Sign-On (SSO)

- Three Parties: the third-party application (Relying Party, RP), the Identity Provider (IdP) and the client device

Who are you?

Who are you?

This is my identity proof

This is your identity proof

Welcome, Ronghai!

Welcome, Ronghai!

Tell IMDb my identity

Tell IMDb my identity
1). **Req0**: User visits App
Single Sign-On Protocol Flow

1). **Req0**: User visits App

2). Redirect_uri, state

https://www.facebook.com/dialog/oauth?
client_id=127059960673829&
response_type=code&
redirect_uri=https://secure.imdb.com/oauth/facebook&
scope=email,publish_stream,user_about_me&
state=13ce
Single Sign-On Protocol Flow

1). **Req0**: User visits App

2). Redirect_uri, state

3). User authentication & grant permissions
Single Sign-On Protocol Flow

1. Req0: User visits App
2. Redirect_uri, state
3. User authentication & grant permissions
Single Sign-On Protocol Flow

1). Req0: User visits App
2). Redirect_uri, state
3). User authentication & grant permissions
4). code + state
5). Req1: code + state

https://secure.imdb.com/oauth/facebook?
code=AQD-nc...PqjeZC4HDFzR3RWjOp9b3M&
state=13ce#_=_
Single Sign-On Protocol Flow

1). **Req0**: User visits App
2). Redirect_uri, state
3). User authentication & grant permissions
4). code + state
5). **Req1**: code + state
6). code + redirect_uri + client_secret

https://graph.facebook.com/v2.3/oauth/access_token?
**client_id**=127059960673829&
**redirect_uri**=https://secure.imdb.com/oauth/facebook&
**client_secret**={client-secret} &
**code**=AQD-nc...PqjeZC4HDFzR3RWjOp9b3MoS4oA
Single Sign-On Protocol Flow

1). Req0: User visits App

2). Redirect_uri, state

3) User authentication & grant permissions

{"token_type": "Bearer", "expires_in": 7104, "access_token": "CAABzj3PSN8C6OELrcr44hS1ITO6..."}

7). Access token
Single Sign-On Protocol Flow

1). Req0: User visits App
2). response_type=code
3). User authentication & grant permissions
4). code + state
5). Req1: code + state
6). code + redirect_uri + client_secret
7). Access token
8). API request for the user’s resource: access token
9). The user resource hosted by IdP
Single Sign-On Protocol Flow

1). **Req0**: User visits App

2). Redirect_uri, state

3). User authentication & grant permissions

4). code + state

5). **Req1**: code + state

6). code + redirect_uri + client_secret

7). Access token

8). API request for the user’s resource: access token

9). The user resource hosted by IdP

10). **Req2**: User info request
**Single Sign-On Protocol Flow**

1. **Req0**: User visits App
2. Redirect_uri, state
3. User authentication & grant permissions
4. code + state
5. **Req1**: code + state
6. code + redirect_uri + client_secret
7. Access token
8. API request for the user’s resource: access token
9. The user resource hosted by IdP
10. **Req2**: User info request
11. User info
Complication of Single Sign-On

- Multi-party systems
- Multi-step operations

SDKs are provided to help RP developers to implement SSO services.
SDK Usages

- SDKs are provided not only by IdPs but often by 4th-party SDK providers (for cross-IdP support of an application)
- If the SDK is not secure, any application using the SDK will be insecure!

<table>
<thead>
<tr>
<th>SDK Names</th>
<th># of Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook SDK</td>
<td>602,297</td>
</tr>
<tr>
<td>Request-OAuthLib</td>
<td>4,785,778</td>
</tr>
<tr>
<td>OAuthLib</td>
<td>6,476,894</td>
</tr>
<tr>
<td>Sinaweibopy</td>
<td>28,019</td>
</tr>
<tr>
<td>OAuth2Lib</td>
<td>Not found</td>
</tr>
<tr>
<td>Rauth</td>
<td>487,275</td>
</tr>
<tr>
<td>Python-weixin</td>
<td>1,404</td>
</tr>
<tr>
<td>BoxSDK</td>
<td>77,074</td>
</tr>
<tr>
<td>Renrenpy</td>
<td>10,387</td>
</tr>
<tr>
<td>Douban-client</td>
<td>30,601</td>
</tr>
</tbody>
</table>

*: The number is retrieved from PyPI statistics and is a conservative estimate. The installed number for many IdPs (e.g., Facebook, Wechat, Renren, Douban), may not be included in the statistics.
Possible Attacks due to Vulnerabilities in SDKs

• Many attacks are due to the incorrect implementations of SDKs
  - For example, the SDK does not check the existence of access token (profile vulnerability

[3,10])

RP server
IMDb

Who are you?
User

Tell IMDb my identity
IdP server

Welcome, Ronghai!

UID

Access_Token + UID

Prior Work on SSO Security

• Formal analysis of SSO protocol standards, including model checking [4,5,6] and cryptographic proof [7]


Prior Work on SSO Security (cont’d)

- Real-world vulnerability discovery using network traffic analysis [8,9,10] or Model-based automated testing [11,12]

- Discovery of hidden assumptions required for the proper use of SDK [13]


Prior Work on SSO Security (summary)

• Formal analysis of SSO protocol standards, including model checking [4,5,6] and cryptographic proof [7]

• Real-world vulnerability discovery using network traffic analysis [8,9,10] or model-based automated testing [11,12]

• Discovery of hidden assumptions required for the proper use of SDK [13]

Little effort has been devoted to a systematic testing of implementation flaws in SSO SDK internals
Goal & Scopes

• Is an SSO SDK vulnerable by itself?
  o Work properly under whatever inputs from the attacker?

• Focus on logic vulnerabilities of the RP server SDK internals
Threat Model

• The attacker can lure the victim to visit a malicious RP (mRP) server (to obtain a valid access token of the victim’s IdP account, but binding only to the mRP)

• The attacker can setup an external machine and use his/her own account to freely communicate with the client, the IdP server, and the RP server

• When HTTPS is NOT used, the attacker can eavesdrop victim’s communication
Roles of S3KVetter

- Single Sign-on SDK Vetter (S3KVetter)
  - Interact with the RP server and the IdP server as if it is the client device
  - Communicate with the RP server on behalf of the attacker
Overview

- Extract all the program paths from the SDK (via concolic execution)
- Define the security properties (i.e., expected behaviors) for SSO SDKs
- Check whether the security properties hold on every program path (via theorem prover)
Technical Challenges

• Multiple-party communication and multi-lock-step operations
  - Some nonce parameters (e.g. code, state) can only be used for once
  - Some parameters are remotely generated and consumed by the remote server

Extra effort required to create/manage inputs to properly feed SSO SDKs
Issues of Existing Approaches for performing Symbolic Execution on Multi-party Systems

• Run the external functions (of SDK) concretely
  ○ Remote IdP API imposes limit on API access rate

• Return a random value of the same return type of external functions without execution, e.g., DART
  ○ This causes false positives to the testing results

• Symbolically explore the external functions, e.g., KLEENet
  ○ We do not have access to the source code of the IdP server
S3KVetter System Architecture
Scheduling Request Orders

• Use a general and simple scheduling algorithm

• Interest in authentication property
  ○ Completed by the last request only
Coordinating Multi-party Communication

- S3KVetter simulates and modifies the external world for the SDK
  - S3KVetter generates a nonce parameter internally, on behalf of IdP
  - Use this nonce parameter, if it satisfies the conditions of the path to be explored
  - Locally solves the condition, if the nonce parameters does not satisfy the condition
Example Symbolic Predicate Tree of SDK

```python
def init():
    auth_url, state = oauth.authorization_url(base_url)
    return redirect(auth_url)
def callback():
    token = oauth.fetch_token(token_url, secret, auth_response=request.url)
    session['oauth_token'] = token
    return redirect(url_for('.profile'))
def profile():
    return oauth.get('https://idp.com/user')
def fetch_token(token_url, secret, auth_response):
    if state and params.get('state', None) !=
        state:
        raise MismatchingStateError()
```

Diagram: Symbolic Predicate Tree
- **start**
- **Req0Flag**
- **Req1[uri]**
  - `startwith('https://')`
  - `not startwith('https://')`
- **Req1[state]**
  - `= state`
  - `return None`
- **code in uri**
  - `in uri`
  - `not code in uri`
- **refresh_token in Req1**
  - `return uid`
- **post(code)**
  - `return uid`
- **path constraint**
- **network call**
- **simplified path**
- **symbolic variable**
Translate the Predicate Tree

- Represent the predicate tree with SMT-Lib v2.0

\[
\text{not } (\text{Req0Flag } 0) \text{ and } (\text{str.prefixof } "https" \text{ Req1[uri]}) \text{ and str.contains } \text{uri code and ...}\\
\]

or

\[
\text{not } (\text{Req0Flag } 0) \text{ and not } (\text{str.prefixof } "https" \text{ Req1[uri]}) \text{ and ...}\\
\]

or ...

\[
\text{or not } (\text{Req0Flag } 0) \text{ and not } (\text{str.prefixof } "https" \text{ Req1[uri]}) \text{ and ...}\\
\]
Formulation of Desired Security Properties

- Key observation: An RP server should login the user if and only if the exact user has actually authorized this specific RP

```python
RPsession.uid == TokenRecordsOnIdP[RPsessions.
    access_token].uid and
RPsession.uid == CodeRecordsOnIdP[RPsessions.
    code].uid and
RPsession.uid == TokenRecordsOnIdP[RPsessions.
    refresh_token].uid and
appid == TokenRecordsOnIdP[RPsessions.
    access_token].appid and
appid == TokenRecordsOnIdP[RPsessions.
    refresh_token].appid and
RPsession.uid == IdPsessions.uid
```
Results Overview

• Discover 7 types of vulnerabilities on 10 popular SDKs
  o Four types are previously unknown

• Consequences:
  o Hijack user’s account in the RP, e.g. due to Access Token injection
  o Sniff user’s activities in the RP due to Use-before-Assignment of state
Statistics of SDK under Study

- Compared to the Baseline, S3KVetter can achieve 2%-13% higher statement coverage and 2%-19% higher branch coverage for the SSO SDKs under test.
- Discover 8 additional vulnerabilities in ten SSO SDKs.
Summary of Discovered Vulnerabilities

- Use-before-Assignment of “State” variable => Allow sniffing of Victim activities via CSRF attacks
- Bypass MAC key, Refresh Token injection and Access Token Injection => Attacker can hijack Victim’s RP account.

These vulnerabilities have been fixed after we informed the developers of the corresponding SDKs.
Example Vulnerability in OAuthLib: Access Token Injection

• Root cause of access token injection

```python
def _populate_attributes(self, resp):
    if 'code' in resp:
        self.code = resp.get('code')
    if 'access_token' in resp:
        self.access_token = resp.get('access_token')
    if 'mac_key' in resp:
        self.mac_key = resp.get('mac_key')
```

• An attacker can remotely inject any access token of her choice

https://RP.com?state=xxx&code=fake code &access token=victim’s access token at mRP
Example Exploit for Access Token Injection

• Assume the attacker has Alice’s access token
  o Setup a malicious RP, and lure the victim to login
  o Inject victim’s access token into the RP server

• The RP can obtain victim user’s resource hosted in IdP

• The attacker can log into the RP as the victim user
Conclusion

• Measurement study and new findings
  o In-depth security analysis on ten popular SSO SDKs
  o 7 types of serious logic vulnerabilities, four are previously unknown

• Vulnerability detection for multi-party systems
  o Symbolizing request orders & multi-party coordination
  o Other usages: Payment system, etc.
Thank you!

Ronghai Yang$^{1,2}$, Wing Cheong Lau$^1$, Jiongyi Chen$^1$ and Kehuan Zhang$^1$

$^1$The Chinese University of Hong Kong and $^2$Sangfor Technologies Inc.