Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization

Rui Wang\textsuperscript{1}\textsuperscript{*}, Yuchen Zhou\textsuperscript{2}\textsuperscript{*}\textsuperscript{†},
(*Lead authors, †Speaker)
Shuo Chen\textsuperscript{1}, Shaz Qadeer\textsuperscript{1}, David Evans\textsuperscript{2} and Yuri Gurevich\textsuperscript{1}

\textsuperscript{1}Microsoft Research and \textsuperscript{2}University of Virginia
Most modern apps are empowered by online services.
Single Sign-On Service
If the developers follow the guides properly, will the application be secure?

The requested response type, one of code or token. Defaults to code...
Possible Attack

Facebook back end

1. dialog/oauth...
2. access_token
3. access_token
4. access_token
5. exchange user info
6. user id/email

Malicious App Client

Foo App Server

Welcome, Alice!

Welcome Alice?!
Video Demo
Who’s to blame?

Developers?

SDK providers?

Developer guide does not explicitly state that token flow MUST not be used for server-side authentication.
If developers follow the guides properly, will the applications be secure?

No.

• Not because of vulnerabilities in SDKs.
• Due to *implicit assumptions* about how to use them.
Implicit Assumptions

Assumptions that are
  o not clearly stated in the SDK’s developer guides;
  o related to how the SDK should be used;
  o essential for application’s security property.

Goal of this project:
  systematically find these implicit assumptions
General Approach

Iterative process

Add necessary assumptions → Refine Model → Assumptions

Model checks? → Y → More to model

N → Counter-Example

Iterative process

Enrich Model → Add related assertions

Y → Iterative process

Add necessary assumptions → Refine Model → Assumptions

Model checks? → Y → More to model

N → Counter-Example
General Approach

Target: Single-Sign-On Systems
System Architecture

FooApp<sub>C</sub>
- Client SDK
- Client runtime

Identity Provider (IdP)

FooApp<sub>S</sub>
- Service SDK
- Service runtime
Threat model

- FooApp
- Client SDK
- Client runtime
- MalApp
- Service SDK
- Service runtime
- Identity Provider (IdP)
- Mallory
- FooApp$_{\text{C}}$
- FooApp$_{\text{S}}$
Desired Security Properties

**Authentication**
Mallory cannot sign into Foo app as Alice.

**Authorization**
Mallory cannot access data that Alice have not granted permission to Mallory.

**Association**
The *user identity*, *user’s permission* (authorization result), and *session identity* must represent the same person.
System Architecture

FooApp\textsubscript{C}

- Client SDK
- Client runtime

Identity Provider (IdP)

FooApp\textsubscript{S}

- Service SDK
- Service runtime
Modeling SDKs

FooApp_{C}  
Client SDK  
Client runtime  

FooApp_{S}  
Service SDK  
Service runtime  

Identity Provider (IdP)  

Concrete module with src or documentation
Modeling underlying system layer
Modeling Foo application
Modeling Mallory

Abstract module subject to dev guide
Concrete module with src or documentation
Black-box concrete module
Abstract module subject to knowledge pool
Modeling Mallory

- **Concrete module with src or documentation**
- **Black-box concrete module**
- **Abstract module subject to dev guide**
- **Abstract module subject to knowledge pool**
SDK documentation and *previously uncovered assumptions*

 Assumptions  

 All relevant system components

 Model  

 Assertions

 3 security properties

**Verifier**
Boogie/BoogiePL: Symbolic Execution engine

Supports loop invariants and function pre/post conditions to enable unbounded verification

SDK documentation and previously uncovered assumptions

Assumptions

All relevant system components

Model

3 security properties

Assertions

Verifier: Boogie[1]

Results

Verifier

Add necessary assumptions
Refine Model
Enrich Model
Add related assertions

Assumptions
Model
Assertions

Model checks?

Counter-Example

More to model

N
Y
N
Y

Counter-Example

Add necessary assumptions
Refine Model
Enrich Model
Add related assertions

Assumptions
Model
Assertions

Model checks?

Counter-Example

More to model

N
Y
N
Results overview

Explicated three SDKs: (6 months)

Many implicit assumptions were found:

- Facebook SSO PHP SDK
  - 5 cases reported,
  - 4 fixed, 3 bounties (3x).

- Windows 8 SDK for modern apps
  - One case reported;
  - documentation revised.

- Windows Live connect SDK
  - Paragraph added to OAuth 2.0 standard.

Majority of tested apps vulnerable due to failure to ensure at least one uncovered assumption.
SDK models

Facebook PHP Source code

protected function getUserFromAvailableData() {
    if ($signed_request) {
        ...
        $this->setPersistentData('user_id', $signed_request['user_id']);
    }
    $user = $this->getPersistentData('user_id', $default = 0);
    $persist_token =
        $this->getPersistentData('access_token');
    $access_token = $this->getAccessToken();
    if ($access_token &&
        (!$user || $persist_token == $access_token)) {
        $user = $this->getUserFromAccessToken();
        if ($user)
            $this->setPersistentData('user_id', $user);
        else
            $this->clearAllPersistentData();
    }
    return $user;
}

public function logoutUrl() {
    return $this->getUrl('www', 'logout.php',
        array_merge(array(
            'next' => $this->getCurrentUrl(),
            'access_token' => $this->getAccessToken(), ), ...));
}

Boogie PL Model

procedure {inline 1} getUserFromAvailableData() returns (user:User) {
    if (idP_Signed_Request_Records__user_ID[signed_request] != _nobody) {
        ...
        user := idP_Signed_Request_Records__user_ID[signed_request];
        call setPersistentData__user_id(user);
        return;
    }
    call user := getPersistentData__user_id();
    call persisted_access_token := getPersistentData__access_token();
    call access_token := getAccessToken();
    if (access_token >= 0 &&
        (!user || _nobody && persisted_access_token == access_token)) {
        call user := getUserFromAccessToken(access_token);
        if (user != _nobody) {
            call setPersistentData__user_id(user);
        } else {
            call clearAllPersistentData();
        }
    }
    return;
}

procedure {inline 1} logoutUrl() {
    returns (API_id: API_ID, next__domain: Web_Domain, next__API: API_ID, own_login: access_token: int) {
        API_id := API_id_FBConnectServer_login_php;
        call access_token := getAccessToken();
        call next__domain, next__API := getCurrentUrl();
    }
API models

Constructing a URL to the OAuth Dialog
To invoke the OAuth Dialog, redirect the user's browser to a URL of the form:

https://www.facebook.com/dialog/oauth?
client_id=YOUR_APP_ID&
redirect_uri=YOUR_REDIRECT_URI&
state=YOUR_STATE_VALUE&
scope=COMMA_SEPARATOR_LIST_OF_PERMISSION_NAMES

Parameters
The OAuth Dialog supports the following parameters which may be passed in the URL string:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_id</td>
<td>Yes**</td>
<td>Your App ID. This is called client_id instead of app_id for this particular method in order to be compliant with the OAuth 2.0 specification.</td>
</tr>
<tr>
<td>redirect_uri</td>
<td>Yes**</td>
<td>Should not be set when using the JS SDK to invoke the dialog. The URL to redirect to after the user clicks a button in the dialog. The URL you specify must be a URL of with the same base domain as specified in your app's settings, a Canvas URL of the form <a href="https://apps.facebook.com/your_app_name">https://apps.facebook.com/your_app_name</a> or a Page Tab URL of the form <a href="https://www.facebook.com/PAGE_USERNAME/app/your_APP_ID">https://www.facebook.com/PAGE_USERNAME/app/your_APP_ID</a>.</td>
</tr>
<tr>
<td>scope</td>
<td>No</td>
<td>A comma separated list of permission names which you would like the user to grant your application. Only the permissions which the user has not already granted your application will be shown.</td>
</tr>
<tr>
<td>state</td>
<td>No</td>
<td>A unique string used to maintain application state between the request and callback. When Facebook redirects the user back to your redirect_uri, this parameter's value will be included in the response. You should use this to protect against Cross-Site Request Forgery.</td>
</tr>
<tr>
<td>response_type</td>
<td>No</td>
<td>The requested response type, one of code or token. Defaults to code. If left unset, or set to code the Dialog's response will include an OAuth code which can be exchanged for an access token as per the server-side authentication flow. If set to token, the Dialog's response will include an oauth user access token in the fragment of the URL the user is redirected to - as per the client-side authentication flow.</td>
</tr>
<tr>
<td>display</td>
<td>No</td>
<td>The display mode with which to render the Dialog. One of page, popup or touch. Defaults to page when the user is using a desktop browser or the dialog is invoked on the <a href="http://www.facebook.com">www.facebook.com</a> domain. Defaults to touch when the user is using a mobile browser or the dialog is invoked on m.facebook.com domain. No other display type is allowed on m.facebook.com. In page mode, the OAuth dialog is displayed in the full Facebook chrome. In 'popup' mode, the OAuth dialog is displayed in a form suitable for embedding in a popup window. This parameter is automatically specified by most Facebook SDK, so may not need to be set explicitly.</td>
</tr>
</tbody>
</table>

**Important:** When using the JS SDK do not specify client_id or redirect_uri - these will be set by the SDK.

procedure {:inline 1} dialog_oauth(IdPLoggedInUser:User, client_id: AppID, redirect_domain: Web_Domain, scope:Scope, response_type:ResponseType) returns (r:int, Response_data: int) modifies Access_Tokens__TokenValue, Access_Tokens__user_ID, Access_Tokens__Scope; modifies Codes__user_ID,Codes__App_ID,Codes__Scope; modifies …

{ var access_token:int, code:int, sr:int;

  if (response_type==_Token || response_type==_Signed_Request){
    havoc access_token; //it means "access_token := *;"
  } else if (response_type==_Token) {  
    Response_data:=access_token;
  } else if (response_type==_Code) { 
    Response_data:=code;
  } else {
    Response_data:=sr;
  }
  r:=200;  
}
Example vulnerability from Facebook SDK

```
require 'facebook-php-sdk/src/facebook.php';

$Facebook = new Facebook(array('appId' => 'YOUR_APP_ID','secret' => 'YOUR_APP_SECRET',));

// Get User ID
$user = $Facebook->getUser();
```

1. Example code snippet from Facebook SDK with comments.

```
if ($user) {
    $logoutUrl = $facebook->getLogoutUrl();
} else {
    $loginUrl = $facebook->getLoginUrl();
}
```

3. Login or logout url will be needed depending on current user state.
Example assumption from Facebook SDK

```plaintext
procedure {:inline 1} getLogoutUrl()
  returns (API_id: API_ID, …)
  modifies …;
  {
    var test_t:int;
    call access_token := getAccessToken();
    call test_t := getApplicationAccessToken();
    assume(access_token != test_t);
    API_id := API_id_FBConnectServer_login_php;
    ...
    return;
  }
```

Assumption (in plain text):
`getLogoutUrl()` must not return an application access token to the client.

Assumption (in the model)

Best outcome:
Facebook fixed their SDK code so this assumption is not needed in the newer version.
Example assumption from Windows Live

```php
function saveRefreshToken($refreshToken) {
    // save the refresh token associated with the user ID on the site.
    saveRefreshToken($refreshToken);
}

function handleTokenResponse($token, $error = null) {
    $authCookie = $_COOKIE[AUTHCOOKIE];
    $cookieValues = parseQuerystring($authCookie);
    ...
    if (!empty($token->REFRESH_TOKEN)) {
        saveRefreshToken($token->REFRESH_TOKEN);
    }
    ...
}
```

Two interpretations

- original Live ID developer guide

```plaintext
procedure {:inline 1} saveRefreshToken (refresh_token_index: int, RP_Cookie_index: int)
modifies RP_Refresh_Token_index;
{
    var user: User;
    //call user := getUserID(RP_Cookie_index);
    user := Refresh_Token__user_ID[refresh_token_index];
    if (user == _nobody) {return;}
    RP_Refresh_Token_index[user] := refresh_token_index;
}
```
Example assumption from Windows Live

```plaintext
procedure {:inline 1} saveRefreshToken (refresh_token_index: int,
RP_Cookie_index: int)
modifies RP_Refresh_Token_index;
{
    var user: User;
    call user := get_User_ID(RP_Cookie_Index);
    user := Refresh_Token__user_ID[refresh_token_index];
    if (user == _nobody) {return;}
    RP_Refresh_Token_index[user] := refresh_token_index;
}
```
Example assumption from Windows Live

```php
function saveRefreshToken($refreshToken)
{
    // save the refresh token and associate it with the
    // user identified by your site credential system.
}

function handleTokenResponse($token, $error = null)
{
    $authCookie = $_COOKIE[AUTHCOOKIE];
    $cookieValues = parseQueryString($authCookie);

    if (!empty($token))
    ...
}
```

new Live ID developer guide
Testing Popular Apps

Facebook showcase applications

Popular Windows 8 modern applications using Facebook SSO
## Testing Results

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Number of Apps</th>
<th>Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illustrative example</td>
<td>27</td>
<td>21 (78%)</td>
</tr>
<tr>
<td>Assumption A1 (in the paper)</td>
<td>7</td>
<td>6 (86%)</td>
</tr>
<tr>
<td>Assumption A6 (in the paper)</td>
<td>21</td>
<td>14 (67%)</td>
</tr>
</tbody>
</table>
Conclusion

Missed implicit assumptions can lead to many vulnerabilities when integrating third-party services.

What we advocate: SDK providers explicate SDKs before release.

  o Fix SDK Code
  o Develop Automated Checker
  o Improve SDK documentation
Visit project website for more info:
http://www.cs.virginia.edu/~yz8ra/ExplicatingSDKs_website/

Models available at: https://github.com/sdk-security/Explicated-SDKs

Rui Wang\textsuperscript{1*}, Yuchen Zhou\textsuperscript{2*†},
(* Lead authors, †Speaker)
Shuo Chen\textsuperscript{1}, Shaz Qadeer\textsuperscript{1}, David Evans\textsuperscript{2} and Yuri Gurevich\textsuperscript{1}

\textsuperscript{1}Microsoft Research and \textsuperscript{2}University of Virginia