Towards Enhancing the Security of OAuth Implementations in Smart Phones

Mohamed Shehab and Fadi Mohsen

Department of Software and Information Systems
College of Computing and Informatics
University of North Carolina at Charlotte
Introduction

• What is OAuth?
Introduction

• The Open Authorization (OAuth) standard, enables the resource owner (*user*) to grant permissions to a third-party (*mobile app*) access to their resources that are hosted on a resource provider (*Facebook*).

• With OAuth, the users are no longer required to share their credentials with third party apps in order to grant them authorizations.

• Who uses OAuth? All major service and resource providers such as Google, Facebook, Microsoft, Twitter, Dropbox, GitHub, Salesforce and many others.
Introduction (OAuth Flow)
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue *access tokens* to the third party apps to make *api requests* on behalf of the user.
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue *access tokens* to the third party apps to make *api requests* on behalf of the user.

• These access tokens have limited *scope*. Which limits the permissions granted to the third party app.
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue *access tokens* to the third party apps to make *api requests* on behalf of the user.

• These access tokens have limited *scope*. Which limits the permissions granted to the third party app.

User
(Resource Owner)
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue access tokens to the third party apps to make api requests on behalf of the user.

• These access tokens have limited scope. Which limits the permissions granted to the third party app.
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue access tokens to the third party apps to make api requests on behalf of the user.

• These access tokens have limited scope. Which limits the permissions granted to the third party app.

User (Resource Owner)

Authorization Server

Client

farms Ville
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue *access tokens* to the third party apps to make *api requests* on behalf of the user.

• These access tokens have limited *scope*. Which limits the permissions granted to the third party app.
Introduction (OAuth Flow)

- The OAuth framework enables the user to issue access tokens to the third party apps to make api requests on behalf of the user.

- These access tokens have limited scope. Which limits the permissions granted to the third party app.

1. Authenticate (username/password)

User (Resource Owner)

Authorization Server

Client

facebook
Introduction (OAuth Flow)

• The OAuth framework enables the user to issue access tokens to the third party apps to make api requests on behalf of the user.

• These access tokens have limited scope. Which limits the permissions granted to the third party app.

1. Authenticate (username/password)
2. Approve requested access

User (Resource Owner)

Client

Authorization Server

facebook
Introduction (OAuth Flow)

- The OAuth framework enables the user to issue access tokens to the third party apps to make api requests on behalf of the user.

- These access tokens have limited scope. Which limits the permissions granted to the third party app.

1. Authenticate (username/password)
2. Approve requested access
3. send access token
Introduction (OAuth Flow)

- The OAuth framework enables the user to issue **access tokens** to the third party apps to make **api requests** on behalf of the user.

- These access tokens have limited **scope**. Which limits the permissions granted to the third party app.

1. Authenticate (username/password)
2. Approve requested access
3. send access token

---

**User** (Resource Owner)

**Client**

Token

scope: {post on wall, read photos}

**Authorization Server**

Facebook
Introduction (OAuth Flow)

• User visits the client site.
Introduction (OAuth Flow)

- User visits the client site.
Introduction (OAuth Flow)

• User asked to connect or login using Facebook
Introduction (OAuth Flow)

• User asked to connect or login using Facebook
Introduction (OAuth Flow)

- User asked to connect or login using Facebook
Introduction (OAuth Flow)

- User asked to connect or login using Facebook
Introduction (OAuth Flow)

- User asked to connect or login using Facebook
Introduction (OAuth Flow)

• User asked to connect or login using Facebook

Isolation provided by browser
Introduction (OAuth Flow)

• User asked to grant authorization (scope).
Introduction (OAuth Flow)

• User asked to grant authorization (scope).
Introduction (OAuth Flow)

• User asked to grant authorization (scope).
Introduction (OAuth Flow)

- User asked to grant authorization (scope).

Isolation provided by browser
Introduction (OAuth Flow)

- User asked to grant authorization (scope).

Isolation provided by browser

Requested Permissions (scope)
Introduction (OAuth Flow)

• User asked to grant authorization (scope).
Introduction (OAuth Flow)

• User asked to grant authorization (scope).
Introduction (OAuth Flow)

- User asked to grant authorization (scope).

Requested Permissions (scope)
Introduction (OAuth Flow)

- App gains access to user resources.
Web App OAuth Flow

- The user agent is usually the browser
  - The user interacts through the browser
  - The browser provides the required isolation between the client and the authorization server.

![Diagram of the OAuth Flow]

- Resource Owner
- User-Agent (B)
- Authorization Server
- (A) Client Identifier & Redirection URI
- (B) User Authenticates
- (C) Authentication Code
- (D) Authorization Code & Redirection URI
- (E) Access Token
- Client

IEEE MS 2014. Anchorage, AK
Web App OAuth Flow

• **The user agent is usually the browser**
  • The user interacts through the browser
  • The **browser** provides the required **isolation** between the **client** and the **authorization server**.
Web App OAuth Flow

• The user agent is usually the browser
  • The user interacts through the browser
  • The browser provides the required isolation between the client and the authorization server.
Web App OAuth Flow

- **The user agent is usually the browser**
  - The user interacts through the browser
  - The browser provides the required **isolation** between the client and the authorization server.
OAuth in Smart Phones
OAuth in Smart Phones

- Several smart phone apps request access to user resources that are hosted in resource providers.
  - For example, photo sharing application requesting access to user’s Facebook photo albums.
OAuth in Smart Phones

• Several smart phone apps request access to user resources that are hosted in resource providers.
  • For example, photo sharing application requesting access to user’s Facebook photo albums.

• The main challenges in OAuth implementation in smart phone apps are:
  • How to implement the user-agent?
  • How to communicate the token from the user-agent to the app (client).
Our Contributions
Our Contributions

• We identified the different OAuth implementations in smartphone frameworks, and summarized the vulnerabilities present in each of the implementations.
Our Contributions

• We identified the different OAuth implementations in smart phone frameworks, and summarized the vulnerabilities present in each of the implementations.

• We conducted an empirical study on the OAuth implementations in the SDKs offered by the popular resource providers, and by the app developers.
Our Contributions

• We identified the different OAuth implementations in smartphone frameworks, and summarized the vulnerabilities present in each of the implementations.

• We conducted an empirical study on the OAuth implementations in the SDKs offered by the popular resource providers, and by the app developers.

• We proposed a framework (OAuth Manager) that can provide protections against current OAuth vulnerabilities in smart phones.
Our Contributions

• We identified the different OAuth implementations in smart phone frameworks, and summarized the vulnerabilities present in each of the implementations.

• We conducted an empirical study on the OAuth implementations in the SDKs offered by the popular resource providers, and by the app developers.

• We proposed a framework (OAuth Manager) that can provide protections against current OAuth vulnerabilities in smart phones.

• We compared our framework with other OAuth implementations in terms of performance and security.
OAuth in Smart Phones

• There are three main approaches for implementing OAuth in smart phone apps
  • Type 1: Through an *Embedded Web Browser Component*.
  • Type 2: Using the *Native Browser*.
  • Type 3: Using the *Provider’s Native App*. 
Type 1: Embedded Web Browser Component
Type 1: Embedded Web Browser Component

• The embedded web browser component is a UI component that can be embedded in a mobile app to render online content within the hosting app.
  • WebView in Android.
  • UIWebView in iOS.
  • WebBrowser in Windows.
Type 1: Embedded Web Browser Component

• The embedded web browser component is a UI component that can be embedded in a mobile app to render online content within the hosting app.
  • WebView in Android.
  • UIWebView in iOS.
  • WebBrowser in Windows.

• The embedded web browser component executes in the context of the hosting app and can be controlled, monitored and manipulated by the hosting app.
Type 1: Embedded Web Browser Component

- The embedded web browser component is used to:
  - present the user with the authentication page (username/password).
  - present the user with the authorization page listing the required permissions.
Type 1: Embedded Web Browser Component

- The embedded web browser component is used to:
  - present the user with the authentication page (username/password).
  - present the user with the authorization page listing the required permissions.
Type 1: Embedded Web Browser Component

• The embedded web browser component is used to:
  • present the user with the authentication page (username/password).
  • present the user with the authorization page listing the required permissions.
Type 1: Embedded Web Browser Component

- The embedded web browser component is used to:
  - present the user with the authentication page (username/password).
  - present the user with the authorization page listing the required permissions.
Type 1: Embedded Web Browser Component

- The embedded web browser component is used to:
  - present the user with the authentication page (username/password).
  - present the user with the authorization page listing the required permissions.

No Isolation!!
Type 1: Embedded Web Browser Component
Type 1: Embedded Web Browser Component

• No isolation between the user-agent and the client (app). The client app is both the user-agent and the client app.
Type 1: Embedded Web Browser Component

- No isolation between the user-agent and the client (app). The client app is both the user-agent and the client app.

- A malicious hosting app can take control of the hosted web browser component and launch attacks on both the user authentication and application authorization pages:
Type 1: Embedded Web Browser Component

- No isolation between the user-agent and the client (app). The client app is both the user-agent and the client app.

- A malicious hosting app can take control of the hosted web browser component and launch attacks on both the user authentication and application authorization pages:
  - Can steal the user password
Type 1: Embedded Web Browser Component

- No isolation between the user-agent and the client (app). The client app is both the user-agent and the client app.

- A malicious hosting app can take control of the hosted web browser component and launch attacks on both the user authentication and application authorization pages:
  - Can steal the user password
  - Can modify the authorization page to spoof the user into authorizing permissions to the hosted app.
Type 1: Embedded Web Browser Component

- Can register event handlers in the loaded page to send the username/password to the hosting app.
Type 1: Embedded Web Browser Component

- Can register event handlers in the loaded page to send the username/password to the hosting app.
Type 1: Embedded Web Browser Component

- Can register event handlers in the loaded page to send the username/password to the hosting app.
**Type 1: Embedded Web Browser Component**

- Can register event handlers in the loaded page to send the username/password to the hosting app.

```
//Java (Native Client App)
myWebView.getSettings().setJavaScriptEnabled(true);
myWebView.addJavaScriptInterface(this, "JSInterface");
myWebView.loadUrl("javascript:" + contents of attack.js);

//JavaScript (attack.js)
var submitBtn = document.getElementById('btn_id');
submitBtn.onclick = function(){
  var email = document.getElementById('email_id').value;
  var password = document.getElementById('pwd_id').value;
  JSInterface.jsCall(email, password);
  return true;
}
```
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.

Original requested permissions:

- Your basic info
- Your email address
- Your profile info: activities, birthday, education history, groups, hometown, interests, likes, location, questions, relationship status, relationship details, religious and political views, subscribers and subscribees, website and work history
- Your stories: checkins, events, notes, photos, status updates and videos
Type 1: Embedded Web Browser Component

• Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.

Original requested permissions

Manipulated requested permissions
Type 1: Embedded Web Browser Component

- Manipulate the authorization page to show a different set of permissions than what the user is actually authorizing.

```javascript
var permsUL = document.getElementById('perm_ul');
permsUL.innerHTML = '<li><div>Your photos</div></li>';
```
Type 2: Using the Native Browser

• The client app registers a system wide handler to listen to specific data requests of type “data-code://auth-token“

• The client app sends the user to the native web browser to perform the authentication and authorization stages.

• The browser sends the access token by invoking a request to a url “data-code://auth-token“
Type 2: Using the Native Browser

- The client app registers a system wide handler to listen to specific data requests of type "data-code://auth-token"
- The client app sends the user to the native web browser to perform the authentication and authorization stages.
- The browser sends the access token by invoking a request to a url "data-code://auth-token"

Message passing managed by the mobile framework.
Type 2: Using the Native Browser

- The client app registers a system wide handler to listen to specific data requests of type "data-code://auth-token"
- The client app sends the user to the native web browser to perform the authentication and authorization stages.
- The browser sends the access token by invoking a request to a url "data-code://auth-token"

Message passing managed by the mobile framework.
Type 2: Using the Native Browser

- The client app registers a system wide handler to listen to specific data requests of type "data-code://auth-token"
- The client app sends the user to the native web browser to perform the authentication and authorization stages.
- The browser sends the access token by invoking a request to a url "data-code://auth-token"

Message passing managed by the mobile framework.
Type 2: Using the Native Browser

- The client app registers a system wide handler to listen to specific data requests of type "data-code://auth-token"
- The client app sends the user to the native web browser to perform the authentication and authorization stages.
- The browser sends the access token by invoking a request to a url "data-code://auth-token"
Type 2: Using the Native Browser

- Using the native browser provides the required isolation, however the token can be stolen when it is being returned to the client app.
- A malicious app can exploit the channel between the browser and the client app.
- **Impersonation Attack:** A malicious app can register to listen to the same specific data request that the client app is registered to listen to, which could result in passing the access token to the malicious app.
Type 2: Using the Native Browser

• Because more than one app has registered to listen to the same data type, the user will be asked to choose which app to start.
Type 2: Using the Native Browser

• Because more than one app has registered to listen to the same data type, the user will be asked to choose which app to start.
Type 2: Using the Native Browser

- Because more than one app has registered to listen to the same data type, the user will be asked to choose which app to start.
Type 3: Using the Resource Provider’s App

• This approach requires the resource provider’s native app to be an installed on the smart phone. It is assumed that the provider’s app is trusted.

• The client app sends the user to the provider’s native app to perform the authentication and authorization stages.
Type 3: Using the Resource Provider’s App

• This approach requires the resource provider’s native app to be an installed on the smart phone. It is assumed that the provider’s app is trusted.

• The client app sends the user to the provider’s native app to perform the authentication and authorization stages.

![Diagram of the process](image-url)
Type 3: Using the Resource Provider’s App

- This approach requires the resource provider’s native app to be an installed on the smart phone. It is assumed that the provider’s app is trusted.

- The client app sends the user to the provider’s native app to perform the authentication and authorization stages.
Type 3: Using the Resource Provider’s App

• This approach requires the resource provider’s native app to be an installed on the smart phone. It is assumed that that the provider’s app is trusted.

• The client app sends the user to the provider’s native app to perform the authentication and authorization stages.
Type 3: Using the Resource Provider’s App
Type 3: Using the Resource Provider’s App

- Isolation is provided through the system message passing system.
Type 3: Using the Resource Provider’s App

• Isolation is provided through the system message passing system.

• The main weakness of this approach is that:
  • The user has to install the provider’s app for each service provider they want to use this approach for.
  • Each provider app has a different OAuth flow which makes it difficult for the user to comprehend the OAuth stages.
  • In some cases a malicious app can impersonate the provider’s app and in such case it can retrieve the user’s username/password and can completely control the OAuth flow.
SDKs and Apps Study
SDKs and Apps Study

• Major services providers offer software development kits (SDKs) that can be included in the mobile apps to seamlessly integrate them with their services.
SDKs and Apps Study

- Major services providers offer software development kits (SDKs) that can be included in the mobile apps to seamlessly integrate them with their services.

- We conducted an empirical study on the current OAuth implementation trends followed by different service providers and by the OAuth development choices made by application developers.
SDKs and Apps Study

- Major services providers offer software development kits (SDKs) that can be included in the mobile apps to seamlessly integrate them with their services.

- We conducted an empirical study on the current OAuth implementation trends followed by different service providers and by the OAuth development choices made by application developers.

- In this study:
  - We used 9 resource providers’ SDKs.
  - We investigated the two most popular platforms (iOS and Android).
  - We downloaded, decompiled and analyzed:
    - 231 Facebook integrated apps.
    - 202 Dropbox integrated apps.
## OAuth SDK Implementations

<table>
<thead>
<tr>
<th>Platform</th>
<th>Resource Provider SDK</th>
<th>Embedded Web Component</th>
<th>Native Browser</th>
<th>Installable App</th>
<th>OS Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td>Facebook [5]</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Twitter [6]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dropbox [7]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Microsoft Live [8]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Box [9]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Google Plus [10]</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Instagram [11]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Linkedin [12]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flickr [13]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS</td>
<td>Facebook [14]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Twitter [15]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dropbox [7]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Microsoft Live [16]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Box [17]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Google Plus [18]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Instagram [19]</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Linkedin [20]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flickr [21]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OAuth SDKs and Authentication Models
OAuth Implementation Stats

- We downloaded and analyzed 231 Facebook integrated apps from Google Play.

![Bar chart showing percentage of apps using Facebook SDK and not using Facebook SDK.]

- Forced WebView: 22%
- Forced FB App: 3%
- Default SDK Logic: 56%
- Not Using Facebook SDK: 68%
OAuth Implementation Stats

- We downloaded and analyzed 202 Dropbox integrated apps from Google Play.
Proposed Approach (*OAuth Manager*)

- We propose to use the privilege separation concept to ensure that the client application has no control over the user-agent.

- Based on privilege separation, we removed the critical OAuth components and implemented it in a separate application (secure sandbox).

![Diagram showing the Proposed Approach](Diagram.png)
Proposed Approach (OAuth Manager)

• We conducted performance analysis on our prototype, we measured the response time and the memory usage.

• We performed our experiments on a standard Android device, the Nexus S, that has android version 4.1.2, 1007.89 MB internal memory, 13624.34 MB SDCard, 343 MB RAM, system browser version 4.1.2-485486.

• We also analyzed the security of our framework:
  • Detects impersonation attack.
  • Prevents from stealing and modification attacks.
OAuth Manager Memory Analysis

• We used the Android Debug Bridge (adb) to measure memory overhead.

• We ran out test application multiple times and each time we used different authentication method. We recorded the memory consumption for each method (proportional set size).

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Browser</td>
<td>41386</td>
</tr>
<tr>
<td>Embedded WebView</td>
<td>5525</td>
</tr>
<tr>
<td>Facebook App</td>
<td>22114</td>
</tr>
<tr>
<td>OAuth Manager</td>
<td>13518</td>
</tr>
</tbody>
</table>
OAuth Manager Response Time Analysis

• We performed benchmarking to estimate the overhead of OAuth Manager on displaying pages.

• We used Android Logging System, we added hooks to the code to record the time samples immediately after the user clicks the login button and promptly after successfully loading the authentication page.

<table>
<thead>
<tr>
<th>Method</th>
<th>Response (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Browser</td>
<td>3429</td>
</tr>
<tr>
<td>Embedded WebView</td>
<td>8077</td>
</tr>
<tr>
<td>Facebook App</td>
<td>1879</td>
</tr>
<tr>
<td>OAuth Manager</td>
<td>1892</td>
</tr>
</tbody>
</table>
OAuth Manager Security Analysis

• The OAuth flow based on OAuth Manager is more secure than the other flows, it provides the measures to prevent from the aforementioned attacks.

• It isolates the user-agent and the client apps. It provides a secure WebView that is not accessible to the client app.

• It detects impersonation attacks by scanning the installed packages and detecting possible malicious registered handlers.
Conclusion and Future Work

• Conclusion:
  • We described the design and security assumptions of each of the main OAuth implementations in smart phone apps.
  • We demonstrated the attacks that can be performed on the different implementations and discussed their effects.
  • We conducted an empirical study on the current OAuth implementation trends followed by the service providers and by the OAuth development choices made by app developers.
  • We proposed an application-based OAuth Manager framework, that provides a secure, light, and fast OAuth flow.

• Future Work:
  • Investigate OAuth management at the OS or Core library levels.
  • Investigate methods to enhance the awareness of secure OAuth implementation and usage.
Thank You.

mshehab@uncc.edu