Orange feedback on Mobile Connect SDK for iOS, Android and HTML 5 implementation

1. Introduction

GSMA provides a set of SDKs on the Mobile Connect developer website (mobileconnect.io). These SDKs are aiming to ease the work of a developer from a Service Provider (Web company, Bank, Government,…) willing to implement Mobile Connect on his service (Web or mobile device application).
Offering SDKs has become an industry standard and is expected by the developers to respect a set of coherent rules: provide a detailed yet easy to read documentation provide sample code and easy integration to the development flow.
The SDK currently available on mobileconnect.io has been reviewed by Orange expert's teams in order to assess its quality in comparison to other similar SDKs recognized by the industry for their high quality of implementation.
We here under expose, for each SDK (**HTML5**, **iOS, Android**), the weaknesses and ways of improvement that have been identified.

2. HTML5 SDK

2.1. Raised issues

The HTML 5 SDK seems to be provided for demonstration purposes as the functioning of the SDK is tightly linked to the HTML5 demonstration application: https://github.com/Mobile-Connect/sdk-test-app-html5-v1

This application is based on inter window communication using *window.postMessage()*. The main purpose of the demo application is to show how the REST APIs of the Discovery and OAuth 2.0 are used in a web page but it cannot be used as an industrial support for the Mobile Connect button integration in a partner web service.

It appears from its implementation that developers of the SDK seemed to lack strong knowledge of the OAuth 2.0 protocol and the associated security mechanisms.

The confidentiality of the authentication data of the GSMA application is not insured. In fact the secret of the application is stored in plain text in the JavaScript code. (see chapter 4. Secret Management)

The confidentiality of the authentication data of the OAuth 2.0 partner, the "client_secret" is visible in the result of the Discovery API call.

A simple stop point the debugger of the browser enables to capture this secret.
Capturing the OAuth2.0 client_secret is a critical issue as it enables a third party (the hacker) to forge OAuth 2.0/token requests.

The implementation does not respect the OAuth 2 authorization call flow by offering an API that enables to recover the access tokens (/token) at browser level.
This is a major issue in terms of security as the token API must only be called by the partner backend in order to assure the confidentiality of the access_token

The software developer kit currently offered by the GSMA may be too complex for a non-experimented web developer that may not be fully aware of the authorization and authentication subjects. The SDK lacks a developer integration guide that would be at industry level (ie Facebook, Google).

2.2. Ways of improvement

In order to provide an industry grade HTML5 SDK, Orange recommends:

- The SDK must offer a detailed integration guide presenting, the architecture, call flows and security recommendations for a proper use of the Discovery and OAuth 2.0 APIS

- The SDK must offer a demonstration web site correctly implementing the Mobile Connect button and respecting the security guidelines. (see chapter 5. Secret Management)

- The major part of the processing must be done on back-end side for security reasons, consequently both client-side and server-side SDKs should be provided accordingly

- There must be no direct JavaScript exchange in the browser between the partner site and the GSMA discovery pop-ups. These pop-ups must only establish communication with the partner's backend systems based on http redirect of which the redirect URIs which must be verified by comparing these to the ones declared by the developer for his application on the GSMA portal. No sensitive information (client_secret) must be exchanged during these redirections. (see chapter 5. Secret Management)

- The Discovery process could be made totally asynchronous towards the behavior of the partner web site. The notification of end of discovery and which process is then done on HTML side should only depend on the Framework used by the partner developer and not imposed by the process.

- A better support of the focus (in Chrome browser) by the GSMA pop-up. Some issues have been detected during the tests.

3. iOS SDK

3.1. Raised issues

The iOS SDK is delivered in a binary library or as a source code but not in a dependencies manager which has become an industry standard when offering an SDK or a library.

Documentation is currently not very clear as it mixes, in the iOS SDK, documentation related to other implementations: HTML, Android and iOS.

The test program delivered with the SDK compiles correctly but only offers a white screen. This is not explicit to a developer discovering Mobile Connect.

The APIs provided in the SDK are too « protocol oriented » : authorize, refreshToken, revokeToken. This may not be an issue for developers that are familiar with Mobile Connect and the OAuth protocol. But developers that are not familiar with OAuth may have difficulties to implement these APIs. The Facebook SDK is a great example and a good source de inspiration for an SDK that hides the protocol complexity and offers the developer useful functionalities such as the reuse of an existing client session.

The implementation of the SDK offers implementations of methods that are very close to the standard and do not provide enough abstraction for the developer.

For example:

```
- (void)authorize:(NSString *)url clientID:(NSString *)clientID
                             clientSecret:(NSString *)clientSecret
                                    scope:(NSString *)scope
                              redirectUri:(NSString *)redirectUri
                             responseType:(NSString *)responseType
                                    state:(NSString *)state
                                    nonce:(NSString *)nonce
                                   prompt:(promptEnumType)prompt
                                   maxAge:(int)maxAge
                                acrValues:(NSString *)acrValues
                       authorizationOptions:(AuthorizationOptions *)authorizationOptions
                                  webview:(UIWebView *)webview
```

or

```
- (void)refreshToken:(NSString *)url refreshToken:(NSString *)refreshToken
                                              scope:(NSString *)scope
                                           clientID:(NSString *)clientID
                                       clientSecret:(NSString *)clientSecret
```

The reference documentation detailing the API does not explain much more than what is already described in the standard.

The provided tutorial is much more useful as it presents examples of code. Nevertheless, the following issues have been raised:

- the SDK contains the cliendID and the clientSecret. This is a major issue and cannot be accepted as it represents a major security risk. The SDK must be changed in order to provide a technical solution that not contains the clientSecret.
- The SDK is based on a webview for the user authentication step. This implies the following :
    - The user experience is not as good as a native UI. OS interactions are also more limited.

It is impossible to share the user's session between multiple applications, imposing the user to re-authenticate in each application that uses the SDK.

As a conclusion, this SDK seems to be more like a reference implementation than a finalized SDK that can be offered to developers so they can integrate it in their application. Locally storing the clientSecret is a major issue in this regard. (see chapter 4. Secret Management)

3.2. Ways of improvement

- Provide an abstraction layer on the API that would be more « user authentication » oriented than simply implementing the OIDC fall flows.
- Adapt the implementation to the mobile context as it seems that this SDK is a limited adaptation of the web backend version.
- The use of a webview for the user authentication is not recommended. Adapting the SDK to native UI is highly recommended.
- The SDK should be provided via a dependencies manager such as Cocoapod or Carthage

- The SDK should be written in swift (a more recent language), rather than objective C.

4. Android SDK

4.1. Integration of the Mobile Connect SDK to an Android app

The SDK does not offer an example of integration using Gradle which is a standard tool when it comes to integrating third party libraries in an android environment.
The SDK is not compatible with android M and N security policies concerning dangerous permissions.

```
<uses-permission android:name="android.permission.READ PHONE STATE" />

<uses-permission android:name="android.permission.ACCESS COARSE LOCATION"
/>

<uses-permission android:name="android.permission.ACCESS FINE LOCATION" />

<uses-permission android:name="android.permission.WRITE EXTERNAL STORAGE"
/>

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

The SDK uses a http apache library that is deprecated since version M of the android sdk.

There are no version numbers in the libraries

- gsma-discovery.jar
- gsma-logo.jar
- gsma-mobileconnect.jar

4.2. Discovery

Hosting the client ID and secret of the developer/partner within the application is a major issue (same as for the iOS sdk). In fact, from the binary of the application it is possible to retrieve this info and so usurp the account of the developer.

4.3. Open ID Connect

The parameters of the discovery response which are the credentials for the OpenID Connect requests are transmitted in clear in the test application. This makes it possible to retrieve the URL, clientID and the secret that is needed for the OIDC request that is sent after the discovery phase.

5. Secret management

The current SDK exposed on Mobile Connect's Github asks the SP to put into his webpage a Javascript (DiscoverySDK.js) that handles all discovery operations. The javascript is executed from the end user browser. Internally, this javascript call the API Discovery (/v2/discovery/).

The problem is that current API Discovery uses BasicAuth, and require to put in clear the Discovery "Production key" and "Production secret" of the Service Provider. Anybody clicking on a Mobile Connect button of the SP can retrieve the SP's Discovery credentials.

Then, the discovery process developed in DiscoverSDK.js continues, and the Discovery returns the MNO endpoints, in clear to the browser. So anyone clicking on a MC button of the SP can retrieve the MNO API credentials (client_id, client_secret).

Risks :
By getting the Discovery secret and the client_secret of the SP, a pirate can develop a website with a Mobile Connect button, using Discovery resources and MNO resources for free (the true SP will be paying for him). The pirate can also develop a phishing website: the user thinks he is secure because of the Mobile Connect authentication process (that will do the real MC authentication displaying the Name of the real SP) and may give personal data to the pirate website.

Extract Facebook login online Documentation : https://developers.facebook.com/docs/facebook-login/security#appsecret

"*Therefore the App Secret or an App Access token should never be included in any code that could be accessed by anyone other than a developer of the app. This applies to all methods of code that are not secured like client-side code (such as HTML or Javascript) or native apps (such as iOS, Android or Windows desktop apps) that could be decompiled.*"

5.1. Recommendations

- Remove the current SDK from the MC github. The current SDK does not meet basic security principles, and endanger Mobile Connect and MNO reputation toward SP.
- Develop a new one that triggers API Discovery and API MC from server only. This SDK must be split in two part :
    - A front SDK : HTML (javascript and Ajax) → display the MC button, triggers the server side SDK, and redirect user to discovery
    - A server side SDK : node.js, PHP, java (already exists) → manage Discovery API calls and MC API calls
    - Update online documentation to explain the usage of the 2 parts of the SDK

6. Discovery API specific limitations

The API discovery let the MNC MCC information in clear in the redirection to the SP return URL. This information transits via the user browser.

A pirate can build an equivalent to Pathfinder service, but for free, using the Mobile Connect button of a valid SP. The pirate trigger authentication with multiples numbers. The MNC MCC of those numbers transit via the pirate browser. At the end, MNO will pay the Discovery service for these fake requests.

6.1. Recommendation

This recommendation is more a long term correction of the problem:

- Develop a new version of the API discovery , using oAuth, that rely on a authorization_code, rather that exposing in clear the MNC MCC in the browser
- Other recommendation is to implement the Discovery API in a server to server mode to limit attacks (SDK and document must be updated accordingly).

7. Conclusion

The current SDK implementation provides a solid ground for building an industry grade SDK that will fully answer the needs of developers around the word. Gaining developer engagement and support is critical for the success of Mobile Connect. This can only be achieved **by offering easy to use and secure SDKs**.

Orange strongly recommends GSMA to take into consideration the above stated remarks.

The GSMA and other MNOs are kindly requested to **provide their feedback** on this proposal.

---------- End of document----------