SII-12-18 – PoC apps SURFconext on mobile devices

Project	: SURFworks
Project Year	: 2012
Project Manager	: Remco Poortinga – van Wijnen
Author(s)	: François Kooman
Completion Date	: October 17, 2012
Version	: 0.4

Summary

This document reports on the results that have been achieved as part of the SURFworks 'PoC Apps SURFconext on mobile devices' activity.

Colophon

Programme line	: SURFworks
Part	: SI-Infra
Activity	: SII-12-18 PoC apps SURFconext on mobile devices
Deliverable	: SII-12-18 PoC apps SURFconext on mobile devices
Access rights	: Public
External party	: none

This project was made possible by the support of SURF, the collaborative organisation for higher education institutes and research institutes aimed at breakthrough innovations in ICT. More information on SURF is available on the website www.surf.nl.

6 Matters one should know about mobile apps for SURFconext.

Scenario	Accessing SURFconext services on mobile devices using a browser can be challenging since they tend to be designed for desktop based browsers with more screen estate and other modes of interaction (mouse/keyboard) than mobile browsers (touch). Building native apps for mobile devices is an attractive alternative, at least if a solution can be found for the need to do web-based Single Sign On (Web SSO) every time the app is started. OAuth2 can provide that solution.
What is it?	OAuth2 is a token based solution. Using a token, apps can get access to relevant services or data without requiring Web-SSO every time access is needed. The SSO is done at the time the token is requested from the Authorization Server (AS) associated with the service. To demonstrate how OAuth2 can be used by iOS/Android apps for accessing SURFconext services two template apps and an OAuth2 AS were developed.
Who is it for?	Service developers wanting to build mobile apps that work with their SURFconext connected service.
How does it work?	The Authorization Server can be used to make a service or resource OAuth2 enabled. The template apps can then be used to kick-start development of mobile apps that can access the service or resource.
What can one do with it?	Build OAuth2 enabled services/resources and mobile apps.
More information	SURFconext team

Introduction

Services connected to a federated identity management are typically web-based and targeted at access by desktop (based) browsers. With the sharp rise in availability and use of internet connected mobile devices such as smart phones and tablets, the wish to be able to access these services on those devices is increasing as well. For the larger devices such as the iPad or other tablets, the built-in browser can be used; especially if attention is paid in the service to dealing with the touch based mode of interaction.

For smart phones these measures can also help; but given the smaller screen sizes, native apps can provide a better user experience. OAuth 2.0 can then be used to integrate an app in a federated environment such as SURFconext.

OAuth for Mobile Applications

This document is written for developers wanting to integrate OAuth 2.0 in their native mobile applications. It contains a short introduction to "The OAuth 2.0 Authorization Framework", RFC 6749. After this introduction it describes the generic architecture of how OAuth 2.0 works on mobile devices. The OAuth specification [1] itself is well written and relatively easy to comprehend.

This document was written without any specific mobile operating system in mind, and this introduction should be useful for all operating systems, including "legacy" desktop operating systems. Some screenshots in Appendix A show the application flow and user experience.

What is OAuth?

From the specification [1]:

> "The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf."

The OAuth 1.0 protocol [2] was initially intended for resource sharing between web applications running on web servers. Suppose John, the resource owner (RO), has an account at Facebook and Google. Now Facebook wants to access John's address book at Google. Instead of John giving his Google password to Facebook, a mechanism was developed to work with a temporary, revokable and scoped access token. With this token Facebook can, on behalf of John, access John's address book at Google. The advantages are clear:

1. The token can have an expiry time, e.g.: only valid for 30 minutes;
2. The token can have a limited scope, e.g.: can only be used to *read* the address book, not add addresses, or read emails;
3. John can *revoke* the token making it impossible for Facebook to access the address book, even if the token did not expire yet.

For more “long term” relationships between websites the automatic expiry is less important, but the ability to scope the token and revoke it whenever the user wants *is*.

In OAuth 2.0 the uses cases have been extended to also include applications written in HTML, CSS and JavaScript, the so called “user-agent-based applications”, running in the users' web browser, or on mobile devices as native applications. These application types required some modifications to the protocol. The involvement of enterprise businesses in the IETF specification process also made sure the protocol is extendable for (future) enterprise use cases, e.g.: by adding support for SAML assertions as tokens and the ability to have expiring tokens.

OAuth Protocol

As shown in Diagram 1, the client is the native mobile application running on a (mobile) device. In order to retrieve an access token to access the resources at the resource server the client needs to get permission from the resource owner first. It does this by launching the web browser installed on the device - e.g.: a smart phone - and pointing it to the authorization server's “authorize” endpoint (A). The authorization server will ask the user to authenticate, possibly - if some single sign on mechanism like SAML is used – via its IdP login page. After authentication the resource owner is asked to give permission to, or authorize, the specific client to access the resource (B). After successful authentication, and authorization, the web browser is redirected back to a special URL registered by the application using an “intent” in Android terminology (C). The URL can for example contain a custom “scheme” and look like this: oauth-mobile-app://callback. The web browser will know to open the application that registered this URL scheme and make the authorization code available to the mobile application by adding it as a query parameter. The mobile application then uses a backchannel to exchange this authorization code for an access token (D, E).

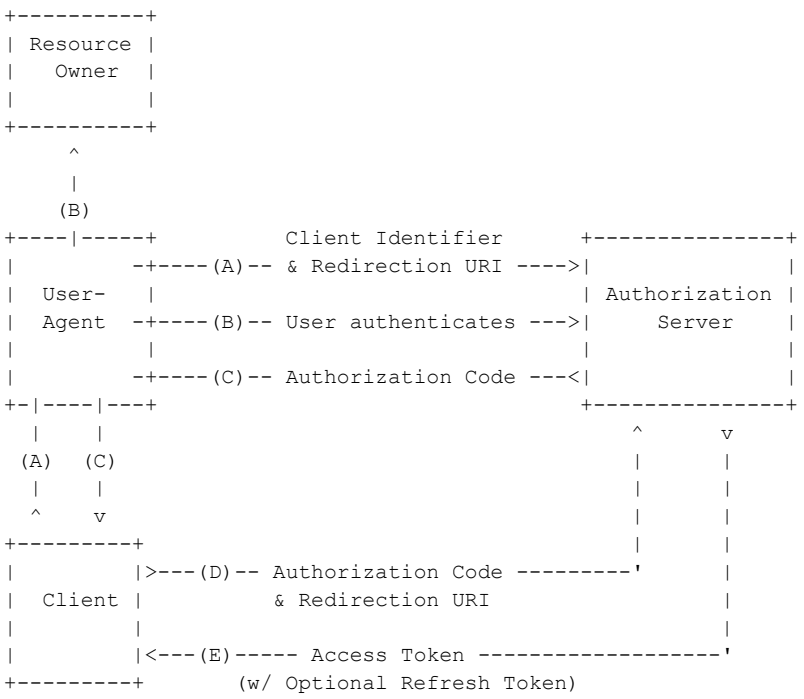

Diagram 1: OAuth 2.0 Authorization Code Flow

Using this access token (E), it is now possible to retrieve the protected resource at the resource server [3]. This protocol is described in "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750. The requested data is returned if the access token is valid for the requested resource (F).

Usually the authorization server and the resource server are located on the same server, but for enterprise deployments those two services can be separated and even run on different machines. In that case an additional backchannel is needed between the resource server and the authorization server for access token verification. Unfortunately, this verification protocol is not (yet) part of the specification.

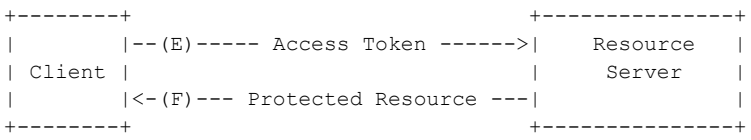

Diagram 2: OAuth 2.0 Accessing the Resource Server

The OAuth 2.0 authorization framework defines two client types, the *confidential* client and the *public* client. By definition a native application is a public client. This means that no secret is needed, but that the application needs to "claim" a redirect URI specifically for this client. Something to note is that every application you install on your device can claim to handle the scheme you use for a redirect URI and thus access the data from the service. This is a good thing, as it allows the end-user to choose their own applications. For auditing purposes this can however be a drawback.

Mobile Application

As part of this project, two template applications [4, 5] were created that implement the OAuth 2.0 authorization framework and interact with an authorization and resource server, while developing the applications, two independent OAuth 2.0 server implementations were used and tested. To get started follow the README file contained in the project.

The first environment is PHP-based [6] containing a sample resource server with exam results of fictional students.

The second environment is Java-based and part of SURFconext, providing group membership information. The OAuth server code is available in the "apis" project [7].

Appendix A shows a flow of the mobile application from application launch to receiving the actual data back in the application.

References

[1] RFC 6749 "The OAuth 2.0 Authorization Framework"

[2] RFC 5849 "The OAuth 1.0 Protocol"

[3] RFC 6750 "The OAuth 2.0 Authorization Framework: Bearer Token Usage"

[4] https://github.com/OpenConextApps/android-oauth-app

[5] https://github.com/OpenConextApps/ios-oauth-app

[6] https://github.com/fkooman/oauth-install-all

[7] https://github.com/OpenConextApps/apis

[8] http://www.surf-academy.nl/archief/event/?id=473

[9] http://www.surf-academy.nl/media/onderwijsdata_121016/121016_onderwijsdata_kooman.pdf

Appendix A

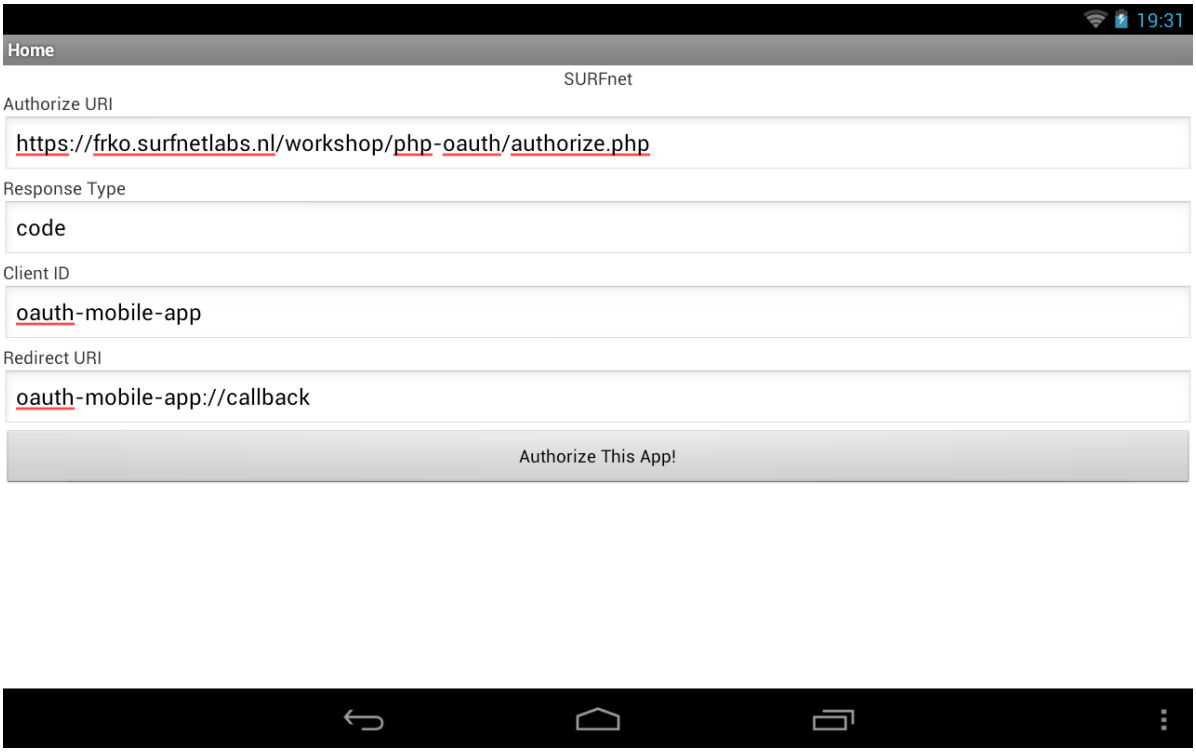

Figure 1: The initial application launch screen, ready to request authorization...

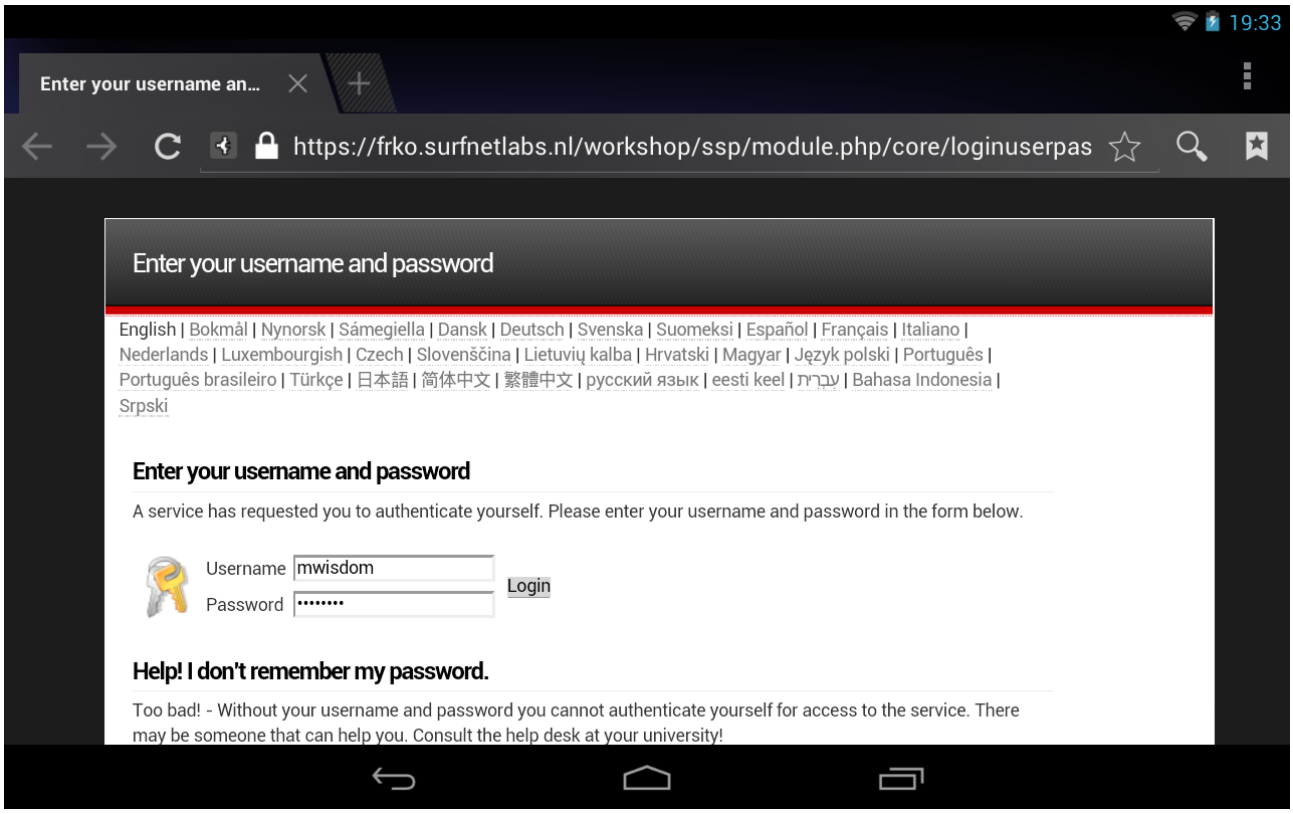

Figure 2: Authentication at protected resource's authorization server...

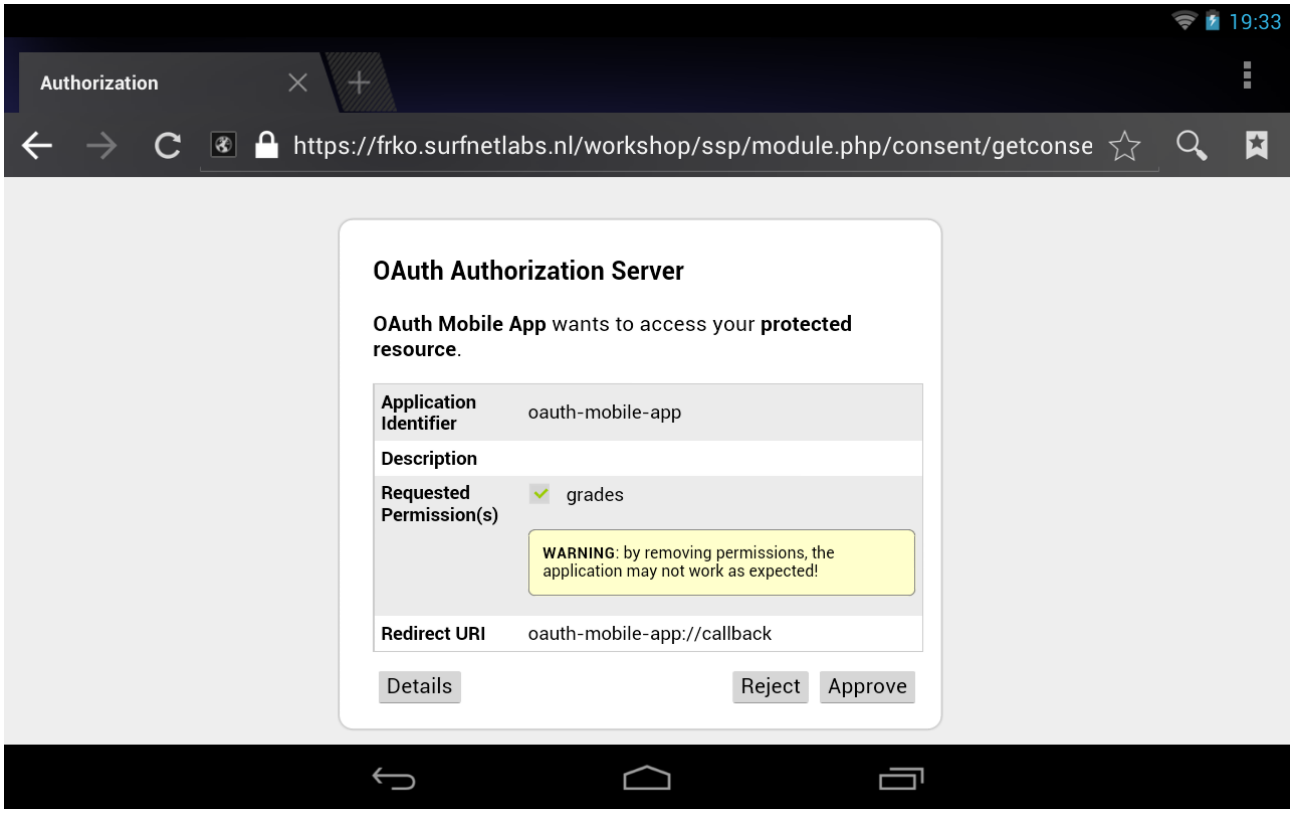

Figure 3: Authorization of the application requesting the data, note the Redirect URI...

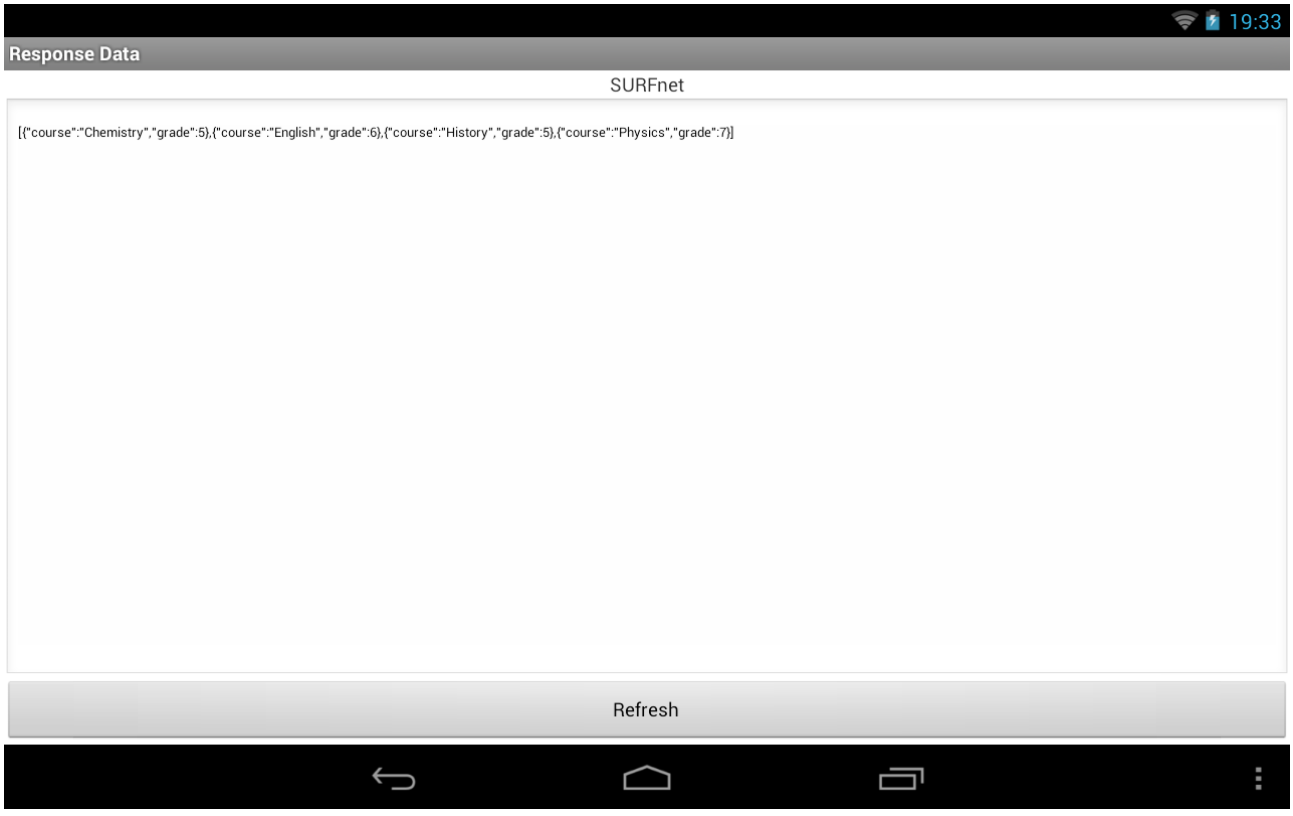

Figure 4: The application can now retrieve the data using the obtained access token...