Linkey

A review into the uses of OAuth in higher education

Alex Bilbie, University of Lincoln

May 2013

[1] http://creativecommons.org/licenses/by/3.0/

Table of Contents

Executive Summary

The Linkey Project was funded by JISC between June 2012 and May 2013 as part of the Access and Identity Management Programme[2].

The mandate of the project was to investigate how the OAuth protocol can be used as part of an access and identity environment in higher education that improves the end-user (staff, students and guests) experience by enabling:

- A consistent and user-centric sign-in experience.
- A richer exchange of user information between applications.
- Easier development and implementation of new products.

The project team worked alongside staff in the ICT Services department at the University of Lincoln to implement a 'single sign-on' service using open source code[3] that was developed as part of the project.

The project website can be found at http://linkey.blogs.lincoln.ac.uk/.

Intended Target Audience

This report is intended for use by developers working in higher education institutions. It will also have some relevance for managers by providing them with a better understanding about what protocols the Internet industry are developing and how their services can be bridged with those already in-place in institutions.

Thanks

The author would like to thank the following people for their kind contributions towards this report: Joss Winn, David Flanders, Nick Jackson, Alex Dutton, Mathew Mannion, and Chris Brown.

[2] http://www.jisc.ac.uk/whatwedo/programmes/aim.aspx
[3] https://github.com/php-loep/oauth2-server

Introduction

Open APIs are one of the driving forces behind the modern web. Entire eco-systems of third party applications have grown around social media sites like Facebook and Twitter, content services such as YouTube and SoundCloud and shopping websites like Amazon and eBay, resulting in more engaging and connected web experiences. In return, services with open APIs have benefitted by increases in the number of users and creating new revenue streams.

In December 2012 JISC published a report entitled "The Advantage of APIs" [4] which looked at how APIs *"facilitate not only the dissemination of knowledge but also the design of innovative online services for students and staff"* and demonstrated that *"by removing the need to repeat laborious data entry tasks, they save staff time and increase organisational efficiency"*. The report highlighted examples of higher education institutions (including the University of Lincoln) who are making use of APIs to develop innovative applications and services.

Developing and implementing best practice security solutions is not straightforward for the developers who make services and applications and numerous attempts have been tried to help users and services securely share data. API tokens, SAML and OpenID are some of the well known solutions, however there is another that has seen wide-scale deployment over recent years: OAuth.

What is OAuth?

The OAuth website[5] describes OAuth as:

> *An open protocol to allow secure API authorisation in a simple and standard method from desktop and web applications.*

Essentially, OAuth is a security protocol that enables users to grant third-party access to their web resources (e.g. profile information) without sharing their authentication credentials with the third-party. Applications (called "clients" in the OAuth specification) have their own credentials and communicate with an OAuth authorisation server to acquire an "access token" which is then used to authenticate requests to an API server.

OAuth grew out of discussions between developers from Twitter[6] and Ma.gnolia[7] who wanted to authorise desktop applications to access their services. A working group was formed in 2007 to

[4] http://www.jisc.ac.uk/publications/reports/2012/advantages-of-api
[5] http://oauth.org/
[6] http://twitter.com/
[7] http://en.wikipedia.org/wiki/Ma.gnolia

draft a proposal for an open standard. Developers from both Google and Yahoo also contributed to this work.

The first OAuth Core 1.0 draft was released in late 2007. In 2008 it was decided that the Internet Engineering Task Force (IETF) would adopt the specification to allow wider discussion and further standardisation work.

A minor revision (OAuth 1.0 Revision A[8]) was published in June 2008 to fix a security hole. The OAuth 1.0 Protocol was published by the IETF OAuth Working Group in April 2010 as RFC 5849[9].

A number of Internet companies and services adopted OAuth 1.0 but many developers found it complicated to understand and implement because it involved generating cryptographic signatures[10] for every request made between the OAuth authorisation endpoint and the API endpoint. Additionally the protocol was considered "chatty" owing to the number of requests between the client and endpoints in order to acquire an access token and actually get an API response.

In May 2010 work began on version 2.0 of the OAuth protocol. Version 2.0 is **not** backwards compatible with OAuth 1.0a and focuses on implementation simplicity by removing the cryptographic signatures and requiring all requests are sent over TLS/SSL. It also details additional methods to allow OAuth to work in more situations, as well as extensions to the core protocol to enable interoperability with assertion based protocols such as SAML.

OAuth has been adopted by many billion dollar valued Internet services and companies. Here are some of those organisations who already integrated the standard into their online offerings (the version of the OAuth specification they have currently implemented is in brackets):

- Google (v2.0)[11]
- Microsoft (v2.0)[12]
- Salesforce (v2.0)[13]
- Facebook (v2.0)[14]
- Yahoo (v1.0a)[15]
- Twitter (v1.0a)[16]

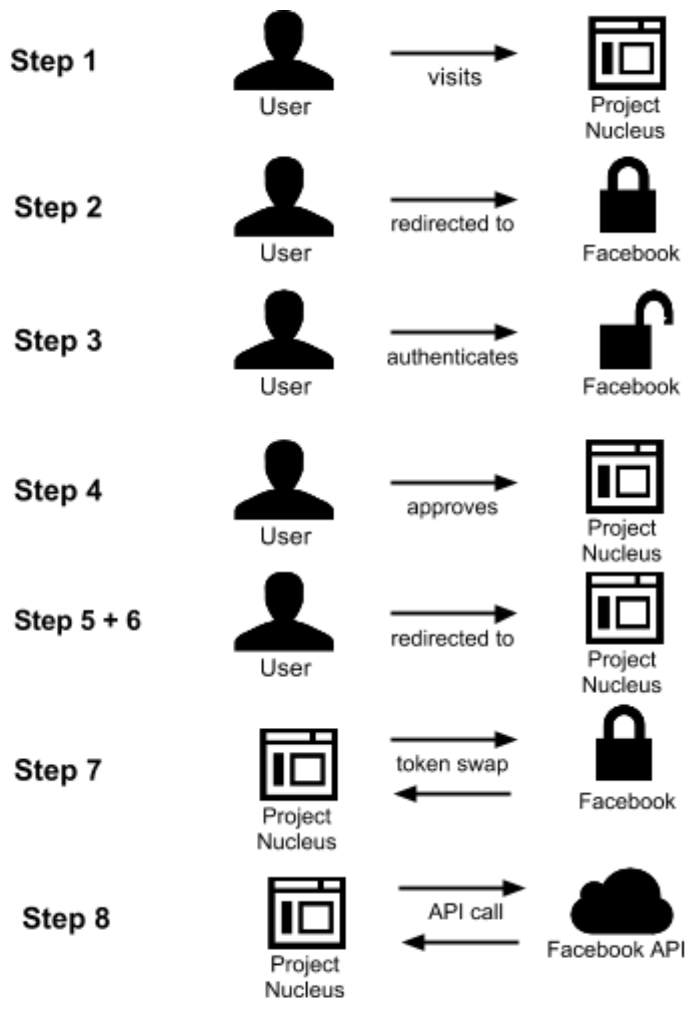
[8] http://oauth.net/core/1.0a/
[9] http://tools.ietf.org/html/rfc5849
[10] Mathematically generated strings based on the parameters of the request
[11] https://developers.google.com/accounts/docs/OAuth2
[12] http://msdn.microsoft.com/en-us/library/hh243647.aspx
[13] http://www.salesforce.com/us/developer/docs/api_rest/Content/quickstart_oauth.htm
[14] https://developers.facebook.com/docs/reference/dialogs/oauth/
[15] http://developer.yahoo.com/oauth/
[16] https://dev.twitter.com/docs/auth/oauth

In May 2013, the OAuth 2.0 specification won the 2013 European Identity Award for Best Innovation/New Standard[17].

A Typical OAuth Experience

The OAuth protocol defines a number of “grants”, which are essentially different ways that a client application can get an access token to use in requests to an API server. The following sequence describes how the “authorisation code” grant (known as the “web-server flow” in OAuth 1.0a) works. The authorisation code grant is the most commonly implemented OAuth flow.

To explain the authorisation code grant the following scenario describes an online game that allows users to sign-in to with their Facebook account.



[17] http://self-issued.info/?p=1026

1. First the user will visit the Project Nucleus website and will click a “sign in with Facebook” link.
2. The user is redirected to a Facebook sign-in screen. In the URL there are a number of query string components that identify the Project Nucleus application, the personal data it is requesting the user allows it to access (their name and email address) and the URL that the user should be redirected back to once they have approved the request.
3. If the user is not already signed into Facebook they will be asked to sign-in.
4. The user will then be presented with details of the application and list the data it is requesting. There will be two buttons that say “approve” and “deny”.
5. If the user clicks the “deny” button, the user is immediately redirected back to the application which in turn will not be able to access the user's data.
6. If the user clicks the “approve” button then they are redirected back to the application with an “authorisation code” in the query string.
7. The application takes this authorisation code and exchanges it in the background for an “access token”. This access token represents permission given by a specific user for a specific application to access a specific set of data.
8. Project Nucleus can then use this access token to authenticate against Facebook’s API to get the user’s private data.

What is important about this experience is that **the user is given clear information about exactly what data is being requested by the application**. This requirement is defined in the OAuth specification and results in the users being made very aware of exactly what personal data they are sharing as well as any additional actions that may be taken on the user’s behalf (such as post to Twitter).

How OAuth compares to SAML, Shibboleth and ADFS

SAML, and associated technologies such as Shibboleth and ADFS, are already well established in higher education institutions in the UK thanks to support and promotion by JISC[18], JANET[19] and organisations who work closely with institutions such as EduServ[20].

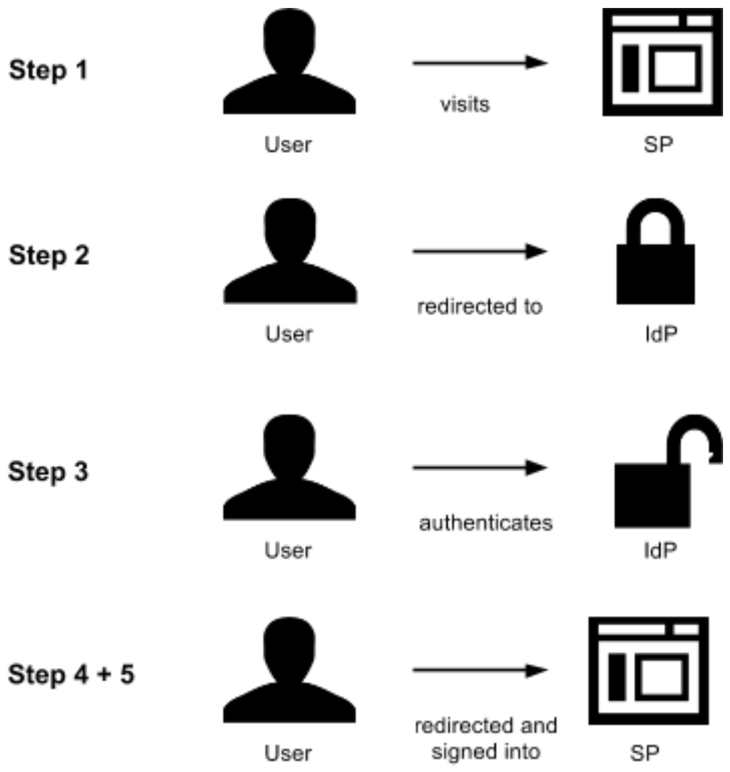
SAML, or Security Assertion Markup Language, is an XML-based framework that allows identity and security information to be shared across security domains – for example between two websites.

Information is exchanged through the concept of assertion tokens, which are XML documents that state facts about the subject of the assertion - for example “this is user John Smith, his identifier is 12345 and he can access the finance and HR systems”.

[18] http://jisc.ac.uk/
[19] http://ja.net/
[20] http://eduserv.org/

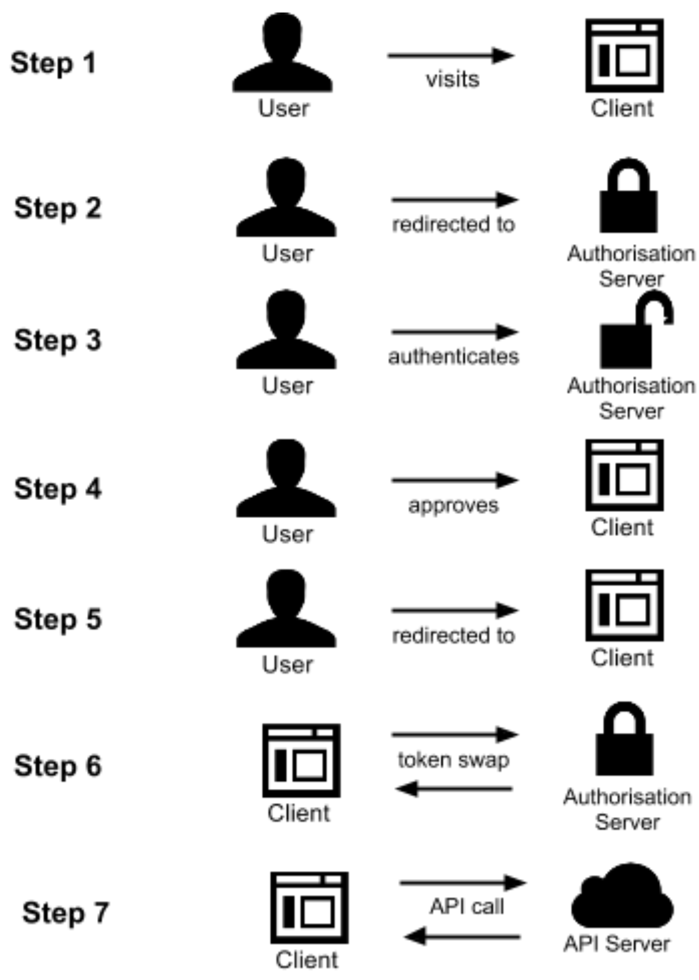
Generally the assertion will be provided by an “identity provider” (IdP) and consumed by a “service provider” (SP).

SAML is often used in “single sign-on” environments. Let’s consider a student trying to sign in to a student union website using SAML:



1. The student (the user) would visit student union website (the SP).
2. The SP would realise the user isn’t yet signed in and so redirects the user to the single sign-on endpoint (the IdP).
3. The user authenticates with the IdP.
4. The IdP redirects the user back to the SP with an assertion.
5. The SP consumes the assertion, validates it and signs the user in.

For reference, this is how OAuth can be used in a single sign on environment:



1. The student (the user) would visit student union website (the client).
2. The client would realise the user isn't yet signed in and so redirects the user to the single sign-on endpoint (the OAuth authorisation server).
3. The user authenticates with the authentication server.
4. The user grants permission for the client to access their private data.
5. The authentication server redirects the user back to the client with an authorisation code in the query string.
6. The client exchanges the authorisation code for an access token by authenticating itself (with the client's own credentials) with the authorisation server.
7. The client then sends a request to the API resource server to request the user's private data. The access token represents permission granted by the user for the client to access the data it requested.

Whilst the OAuth implementation looks longer it contains one important stage (which is a requirement of the specification), specifically step 4, which is where the user is explicitly asked if they want to allow the client to access their data and also what data will be accessed. You could emulate this in the SAML implementation but the upfront clarity of what is being asked of the user in the OAuth flow is a significant benefit benefit for users.

In addition most OAuth authorisation servers allow users to revoke access to clients as well - any access tokens that have been granted to that client on behalf of the user are destroyed which immediately results in requests using those access tokens to the API endpoint will fail.

In both flows, the client/SP itself is verified. In SAML this is done through certificate exchanges, in OAuth the client’s identifier is validated at step 3 and in step 6 both the client identifier and the client's private secret are validated.

OAuth at the University of Lincoln

The teaching and learning strategy at the University of Lincoln is called “Student as Producer”[21] which will make research-engaged teaching an institutional priority, across all faculties and subject areas. Students are supported by student services and professional staff so they can take greater responsibility not only for their own teaching and learning, but for the way in which they manage the experience of being a student.

Total ReCal[22] was a “student as producer” focused JISC funded project which created centralised APIs on top of institutional event data. We made the decision that students should be able to access “their” events as raw data so if an inspired student wanted to build their own visualisations of their academic day (should they find that what the university provided for them to manage their diary didn’t quite work for them) then they could.

We investigated a number of different methods for securing the APIs which included HTTP basic authentication and using API tokens. Neither of these fitted our requirements:

- HTTP basic authentication requires an application to have knowledge of the user’s username and password in order to talk to the APIs. As both students and outside companies could potentially be creating applications for staff and students then it would be irresponsible to recommend users give out their credentials to third parties.
- API tokens given to registered applications wouldn’t require users to give out their credentials but wouldn’t allow us to implement fine grained permissions as the tokens would be application specific instead of user specific.

Early during the TotalRecal project Facebook announced their Open Graph APIs[23] which were secured by an early draft (version 5) of the recently announced OAuth 2.0 specification. Their developer documentation clearly detailed how the OAuth 2.0 protocol worked and how it differed from OAuth 1.0a[24], which at the time many developers had contempt for due to a lack of easy to use client libraries.

This inspired the project team to look into the OAuth 2.0 protocol and resulted in the release of the first implementation of the OAuth 2.0 protocol as a PHP server library for the CodeIgniter framework[25].

[21] http://studentasproducer.lincoln.ac.uk/
[22] http://totalrecal.blogs.lincoln.ac.uk/
[23] http://developers.facebook.com/docs/opengraph/
[24] This documentation has since changed and no longer discusses the differences between the different OAuth versions
[25] https://github.com/alexbilbie/CodeIgniter-OAuth-2.0-Server

OAuth grants each user an application specific access token which allows the application to access their private resources (event data in the case of the Total ReCal project) without requiring users to give out their credentials.

We have since developed on our OAuth endpoint and over 15 internal and external applications now use the service to securely allow students and staff to share their data.

The benefits and impact of OAuth for us

In the past three years we have developed a platform, dubbed ‘Nucleus’, which is a warehouse of institutional data ranging from student and staff information, timetable events, research projects, estate data and bibliographic resources. This data is then exposed by over 100 RESTful[26] API endpoints, which when requested with OAuth access tokens, allow developers inside and outside the university to easily create personalised and secure applications.

We have encouraged computing students to use our API platform to develop their own services which they can use during their time at the university and a number of students each year have developed applications for their final projects by collaborating with different departments across the university to try and solve some of their unique problems they each have.

Example services developed by students include:

- Making use of wall mounted television screens in a new academic building to visualise free rooms which can be used by study groups for private study sessions.
- Replacing paper attendance forms with a mobile application which students can sign-in to mark their attendance resulting in real time statistics and considerable staff labour savings.
- Developing an application for smart phones which automatically silences phones when a student is present in a timetabled session.

Other interesting examples of applications developed using our APIs include:

- A staff directory[27] which uses many different datasets exposed by our APIs to build profiles for members of staff which they can customise after they’ve signed in.
- A researcher dashboard which gives academics the ability to see an overview of all their research. It also gives a single place to perform everyday administrative tasks, and gives improved feedback on the state of bids and projects.
- A prototype mobile application which allows the ICT Services server administrators to track each other’s location (between the office and the different server rooms) by

[26] http://en.wikipedia.org/wiki/REST
[27] http://staff.lincoln.ac.uk/

communicating with the Cisco wifi controllers to discover which access point each of their phones most recently communicated with.
- A short URL generator http://lncn.eu/ which converts long URLs into much smaller ones. This service is heavily used by the marketing department for promotional material. OAuth is used to sign a user into the service.
- An application embedded in our institutional virtual learning environment which allows students to post and rate resources they have discovered which others might find useful.

The mobile application for the server administrators is a good example of where OAuth has a benefit over other authorisation protocols because each time a user signs-in the authorisation server explicitly details what data the application the user is authorising is requesting. As this particular application has some interesting privacy considerations the users are told exactly how they are tracked and re-assured that it only works when their mobile phones are connected to the university network and the data is only shared with other users of the application.

The Student Union have made use of our platform too with their own online services. Previously students had to register manually with their website but now they can sign in with their university credentials and once they've approved that the website may access their basic data it is automatically shared.

The annual student union council election voting application now no longer requires students to manually create an account instead allowing sign-in through our OAuth authorisation server. By reducing the friction required to vote the student union increased the number of students taking part in the elections because union representatives can base themselves around campus with tablet computers and students can vote then and there without having to go out of their way in their own time to register then vote.

As OAuth access tokens are linked to both a user and an application we can easily manage access. As students and staff leave the university we can automatically revoke their tokens, and applications which are either sunset or found to be acting nefariously can have their own credentials revoked as well as all access tokens issued by its users, which immediately suspends the ability to access the entire API and authentication platform.

We can also grant private scopes (OAuth's term for permissions) to our own administrative applications which give them access to additional data or the ability to perform actions not permitted by other applications.

Other institutions using OAuth

Although the use of OAuth within the higher education sector appears to be relatively low, we have become aware of other institutions that are using OAuth as part of their online services.

University of Warwick

The web-team in the IT Services department at the University of Warwick have developed a number of APIs that allow applications developed in house to create and amend content on their online publishing platforms. The primary users of these APIs are the web-team themselves who develop applications in-house. They use the APIs to provide a service oriented architecture across departmental supported applications. By doing this, they can expose applications in a personalised institutional portal (Start.Warwick[28]) and also provide rich client side functionality within web pages and mobile applications.

Additional users of these APIs include technical teams within IT Services, technical users outside of IT Services (both internal and external to the university) who wish to develop personalised applications programmatically, and the APIs allow them scope to do this.

The APIs themselves use OAuth 1.0a to authorise requests. Mathew Mannion the developer and service owner of Warwick's web publishing platform explain why they choose OAuth:

> *"[OAuth] was a standard that fit very well with our needs at the time, which were to provide a gadget container (Start.Warwick) that fit with OpenSocial standards (á la iGoogle). As iGoogle was the reference implementation of such a container, and that used OAuth, OAuth was an obvious choice. The widespread availability of client libraries and documentation was also a factor."*

Their OAuth solution was built on top of a freely available Java library provided by the OAuth foundation[29].

In order to help developers work with their APIs the web-team have provided documentation on their website for accessing Warwick APIs with OAuth[30]. In addition the team provide a Node.js[31] package for the the authentication endpoint, designed to make it easy for new users[32].

The team anticipate implementing OAuth 2.0 as more of their applications support it.

[28] https://start.warwick.ac.uk/
[29] http://oauth.net/code/
[30] http://warwick.ac.uk/oauth
[31] http://nodejs.org/
[32] https://github.com/UniversityofWarwick/passport-warwick-sso-oauth

University of Oxford

The University of Oxford use the open source Sakai software[33] for their virtual learning environment[34]. The Sakai software includes an OAuth 2.0 service provider that allows approved clients to access in-house developed APIs. These provide limited access for users to sign up for events and complete surveys. This functionality is currently used by the University's mobile site, Mobile Oxford[35] to provide a mobile-friendly interface as part of the JISC-funded Erewhon project[36].

Oxford has also been investigating the use of OAuth to mediate access to non-public data hosted by its Open Data Service. By also providing support for Cross-Origin Resource Sharing (CORS[37]), it's possible to develop pure JavaScript clients for exploring institutional data.

[33] http://www.sakaiproject.org/
[34] http://weblearn.ox.ac.uk
[35] http://m.ox.ac.uk/
[36] http://erewhon.oucs.ox.ac.uk/
[37] http://enable-cors.org/

The use of OAuth in enterprise software

Google Apps for Education

Many educational institutions around the world use Google Apps for Education to provide their email, calendar and document solutions for staff and students. Google have developed a large number of OAuth protected APIs[38] available for all of their services which allow developers to add functionality to Google's services and make use of the data in each service for other purposes.

Examples of potential integration between institutional data and Google Apps include:

- Pushing student timetable events to Google Calendar.
- Automatically creating a new Google Apps account when a new student is added to the student management system.
- A research or study group application that allows users to create project websites (using Blogger), manage meetings, deadlines and events (using Google Calendar), allows users to message each other (using Gmail) and store shared documents (using Google Drive).

Different OAuth grants (or flows) are being used in the examples above. User facing applications such as the research group application will use the traditional OAuth flow as described previously. Administrative applications (that may run as a scheduled commands) will generate access tokens each time they run and will have permission (scopes) to access all users' data not just one individual's data.

Salesforce

Salesforce develop online customer relationship management applications and are used by some higher educational institutions[39] (principally in the US). Salesforce were among some of the first online services to have open APIs for their services back in the 1990s.

Like Google, Salesforce's APIs are authenticated with OAuth access tokens allowing new personalised services to be created with data held on the Salesforce platform.

[38] https://developers.google.com/
[39] http://www.salesforcefoundation.org/highered

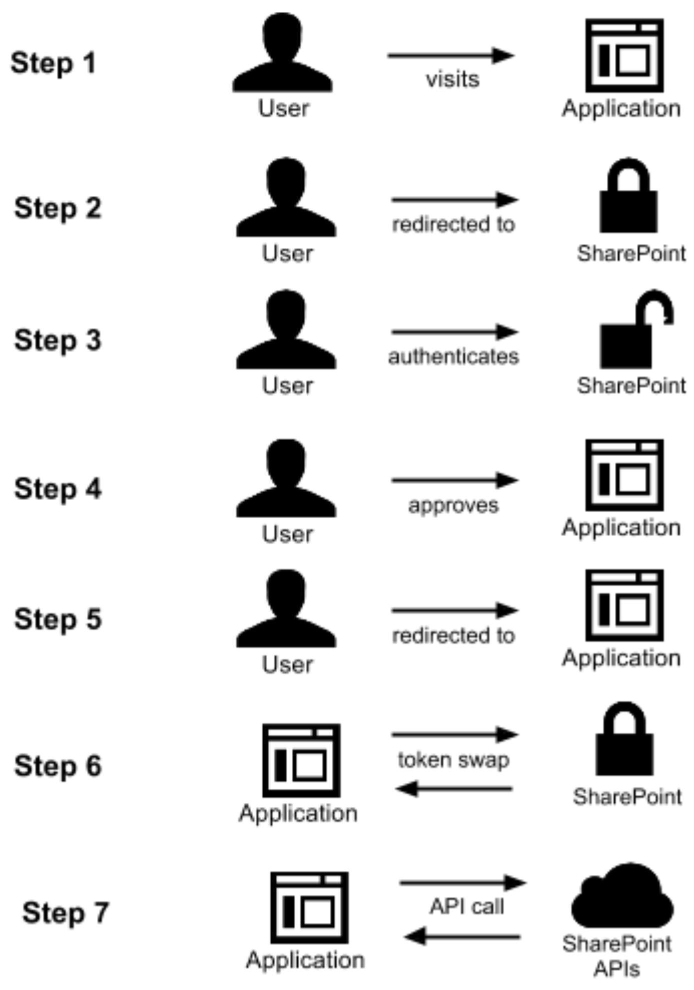
A good example of an application that could be created using their protected APIs is one which automatically sends data to Salesforce about prospective students when they register to attend open days on an institution's website. Academics might then be able to add additional information about prospective students (such as specific courses they're interested in) after speaking to them at the open day using a custom application which has been developed on top of the APIs.

Microsoft SharePoint 2013

Microsoft's SharePoint is a very popular system that is implemented by many institutions to develop internal intranets and Internet-facing websites. In SharePoint 2013, Microsoft have brought some technology from their new cloud services, Office365 and Azure, into Sharepoint[40]. As a result Sharepoint installations can allow 3rd party applications to access and change content that it stores using REST APIs which are protected with OAuth 2.0. In addition, new integrations can be created meaning access to other APIs and data can be delegated through SharePoint.

This is very significant because as traditional desktop applications are brought into the browser (including Microsoft Office with Office365), Microsoft are embracing existing Internet standards instead of developing new proprietary standards. By using OAuth, developers can make use of existing code and libraries that they may already be using to integrate with other OAuth authenticated endpoints such as Google or Saleforce's APIs. Additionally Sharepoint 2013 contains an implementation of an OAuth 2.0 authorisation server, allowing the easy management of applications, scopes, users, and access tokens through an interface in the administration panel or using integrations with Windows PowerShell.

[40] http://msdn.microsoft.com/en-gb/library/fp142384.aspx



An example extension (as illustrated above) to the built-in OAuth functionality in SharePoint 2013 would be creating new APIs in SharePoint that expose data from Active Directory which would then allow a single sign-on system to be developed. A user would be redirected from a website which wishes to sign the user in to the SharePoint installation, approve the website to access their data and finally the website itself would validate the request in return for an access token. With this access token the website can then call the API linked to Active Directory to access information about the user.

Technical challenges and considerations

OAuth 1.0a *vs* OAuth 2.0

OAuth 1.0 is the original version of the OAuth specification. It was developed in 2007 following discussions between developers from a few companies including Yahoo and Twitter. OAuth 1.0a is a very small modification which fixed a security concern.

The principal difference between OAuth 1.0a and OAuth 2.0 is the use of “signed signatures” in 1.0a whereby every single request to the authorisation server and the APIs contains a string that is calculated from all of the parameters in the request and is then signed with the application’s private key. Due to this requirement OAuth 1.0a had very little adoption due to few easy to use implementations available for developers. In addition the specification itself was difficult to understand due to very technical language and few examples. At the time of writing the only major Internet service that still uses OAuth 1.0a to secure their APIs are Twitter.

OAuth 2.0’s main feature was to remove the necessity for signatures by moving the security to the transport layer and requiring that all communication with authentication servers and APIs is over TLS. The specification uses much clearer language and features many more examples as well as grants to implement OAuth for different devices and situations.

In addition to the main specification[41] there are a number of additional specifications (the status of each document at the time of writing is in brackets):

- OAuth use cases (draft 3)[42] - provides a number of use cases where OAuth is a useful solution to a problem
- OAuth mac tokens (draft 3)[43] - details how to request access tokens using signed signatures (similar to OAuth 1.0a).
- OAuth SAML bearer tokens (draft 16)[44] - details how to request access tokens and authenticate applications using SAML assertions.
- OAuth threats model (draft 8)[45] - goes into extensive detail about potential security risks and implications which should be considered.

[41] http://tools.ietf.org/html/rfc6749
[42] http://tools.ietf.org/html/draft-ietf-oauth-use-cases-03
[43] http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-03
[44] http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-16
[45] http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-08

Different Implementations

Many different companies from all sectors have influenced the OAuth 2.0 specification and this has resulted in a very extensible framework. In addition some companies started implementing OAuth 2.0 during it's early stages - for example when Facebook announced their Open Graph APIs they implemented draft 5 of the specification; this has benefitted them but also been a thorn in their side. They benefitted by having an easy to manage and easy to implement security layer for their APIs, but because the specification has evolved over the two years of the standardisation process they had to manage multiple versions which resulted in several[46] public security flaws being published.

Different vendors have implemented slightly different variations in their OAuth endpoints which causes pain for developers integrating with many services:

- Most vendors respond with JSON[47] but some such as Facebook and Foursquare respond with url-encoded plain text.
- In the URL parameters some vendors require the delimiter for the *scope* and *response* parameters to be a comma but the specification says the delimiter should be a url-encoded space.
- Some vendors require access tokens to be sent in the Authorization header, others require it to be passed in the query string.
- No major vendor requires the *state* parameter despite the specification strongly recommending requiring it to mitigate cross site request forgery attacks.
- Some vendors don't require all parameters to be sent in requests which can cause confusion when creating generic client libraries.

One of the other outcomes of the Linkey project is a number of PHP client libraries that enable developers to easily integrate with major OAuth 2.0 vendors[48] and create a standard-compliant OAuth 2.0 authorisation server[49]. Detailed documentation for both can be found on the project wikis.

Skills and Training

All of the large Internet-based companies such as Google have thoroughly documented their APIs and OAuth implementations and in some cases included code libraries in different programming languages to help implement integrations with their services.

[46] http://www.breaksec.com/?p=5753
[47] JavaScript object notation
[48] https://github.com/php-loep/oauth2-client
[49] https://github.com/php-loep/oauth2-server

Developers who are used to SOAP and WSDL web-services may find working with APIs and OAuth very different at first but there are plenty of examples and tutorials available online. It is recommended that developers working with APIs and OAuth make use of vendor developed code libraries (for example the Facebook PHP SDK[50] or the Google .NET client[51]) where they have been provided and they familiarise themselves with the concepts of REST[52] and Hypermedia[53].

OAuth 2.0 Grants

OAuth 2.0 by its nature is a very flexible standard and can be adapted to work in many different scenarios. The core specification includes four authorisation grants which describe how an end application can request an access token that can then be used as authorisation against an API:

- Authorisation code grant
- Implicit grant
- Resource owner credentials grant
- Client credentials grant

The specification also details another grant called the refresh token grant.

As a refresher here is a quick glossary of OAuth terms (taken from the core spec):

- Resource owner (a.k.a. the User) - An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- Resource server (a.k.a. the API server) - The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
- Client - An application making protected resource requests on behalf of the resource owner and with its authorisation. The term client does not imply any particular implementation characteristics (e.g. whether the application executes on a server, a desktop, or other devices).
- Authorisation server - The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorisation.

Authorisation code grant

The authorisation code grant is the grant that most people think of when OAuth is described.

[50] https://developers.facebook.com/docs/reference/php/
[51] https://code.google.com/p/google-api-dotnet-client/
[52] https://en.wikipedia.org/wiki/Representational_state_transfer
[53] http://en.wikipedia.org/wiki/Hypermedia

If you’ve ever signed into a website or application with your Twitter/Facebook/Google/(insert major Internet company here) account then you’ll have experienced using this grant.

Essentially, a user will click on a “sign in with Facebook” (or other IdP) and then be redirected from the application/website (the “client”) to the IdP authorisation server. The user will then sign in to the IdP with their credentials, and then - if they haven’t already - authorise the client to allow it to use the user’s data (such as their name, email address, etc). If they authorise the request the user will be redirected back to the client with a token (called the authorisation code) in the query string (e.g. http://client.com/redirect?code=XYZ123) which the client will capture and exchange for an access token in the background.

This grant is suitable where the resource owner is a user and they are using a client which allows a user to interact with a website in a browser. An obvious example is the client being another website, but desktop applications such as Spotify or Reeder use embedded browsers.

Some mobile applications use this flow and again use an embedded browser (or redirect the user to the native browser and then are redirected back to the app using a custom protocol).

In this grant the access token is kept private from the resource owner.

If an organisation has a mobile application that is for its own service (such as the official Spotify or Facebook apps on iOS), it isn’t appropriate to use this grant as the app itself should already be trusted by its authorisation server and so the resource owner credentials grant would be more appropriate.

Implicit grant

The implicit grant is similar to the authentication code grant described above. The user will be redirected in a browser to the IdP authorisation server, sign in, authorise the request but instead of being returned to the client with an authentication code they are redirected with an access token straight away.

The purpose of the implicit grant is for use by clients which are not capable of keeping the client’s own credentials secret; for example a JavaScript only application.

If you decide to implement this grant then you must be aware that the access token should be treated as “public knowledge” (like a public RSA key) and therefore it must have a very limited permissions when interacting with the API server. For example an access token that was granted using the authentication code grant could have permission to be used to delete resources owned by the user, however an access token granted through the implicit flow should only be able to “read” resources and never perform any destructive operations (i.e. a non-idempotent method[54]).

Resource owner credentials grant

When this grant is implemented the client itself will ask the user for their username and password (as opposed to being redirected to an IdP authorisation server to authenticate) and then send these to the authorisation server along with the client’s own credentials. If the authentication is successful then the client will be issued with an access token.

This grant is suitable for trusted clients such as a service’s own mobile client (for example Spotify’s iOS app). You could also use this in software where it’s not easy to implement the authorisation code - for example we implemented this grant so that our OwnCloud[55] service could retrieve details about a user that aren’t available over LDAP from the university’s Active Directory server which OwnCloud authenticates against.

Client credentials grant

This grant is similar to the resource owner credentials grant except only the client’s credentials are used to authenticate a request for an access token. Again, the use of this grant should only be allowed by trusted clients.

This grant is suitable for machine-to-machine authentication, for example in a cron job which is performing maintenance tasks over an API. Another example would be a client making requests to an API that don’t require user’s permission.

When someone visits a member of staff’s page on the University of Lincoln staff directory the website uses its own access token (that was generated using this grant) to authenticate a request to the API server to get the data about the member of staff that is used to build the page. When a member of staff signs in to update their profile however, their own access token is used to retrieve and update their data. Therefore there is a good separation of concerns and we can easily restrict permissions that each type of access token has.

Refresh token grant

The OAuth 2.0 specification details a fifth grant which can be used to “refresh” (i.e. renew) an access token which has expired.

[54] A method which can be called repeatedly and will always return the same result.
[55] http://owncloud.org/

Authorisation servers which support this grant will also issue a “refresh token” when it returns an access token to a client. When the access token expires, instead of sending the user back through the authorisation code grant the client can use the refresh token to retrieve a new access token with the same permissions as the old one.

A problem with this grant is that the client has to implement additional logic to deal with expired tokens. The client can either request new tokens only when requests fail or ensure all the tokens it posses are valid by requesting new ones in a background schedule.

Developing your own grants

The core OAuth specification defines four grants as described above, however not all of these will suit all situations. Therefore the specification gives some detail on how to implement your own[56]. With all grants the end goal is for a client to receive an access token which it can then use in requests to an API resource server. Two examples of custom grants are below:

Google Glass Mirror grant

Google Glass[57] is a wearable computer with a head-mounted display that is being developed by Google with the mission of producing a mass-market, ever-present and highly integrated (with the real world) computer.

Glass has no keyboard or mouse interface, only a simple touch pad along the right hand frame which allows users to scroll through menus. Therefore Google have developed their own OAuth grant for Glass so that third party applications can be used with Glass. Applications using the Mirror API[58] aren't installed on Glass as they are with a smartphone. Instead applications “push” data to a Google service which then forwards it on to the device. Likewise any interaction from the device itself is proxied back through Google to the application. The grant that Google have implemented is similar to the authorisation code grant but there is no interaction with the device itself as part of the user experience. The emphasis here is that the user is delegating permission for integration via Google.

Cross Client Identity grant

[56] http://tools.ietf.org/html/rfc6749#section-8.3
[57] http://www.google.com/glass/
[58] https://developers.google.com/glass/about

A disadvantage from the user experience perspective of OAuth is that if your application is available on multiple platforms (for example a mobile application and a web application) and they both require integration with a user's Google account then you would have to approve access for each platform to what is (from the user's point of view) the same application.

Google have implemented a grant called “cross client identity”[59] which allows different platform implementations to automatically acquire an access token if they have used one of the other platforms' implementation of an application.

To the user this reduces the friction of using an application as they won't necessarily have to struggle to enter their password on their mobile device if they've already used the web application for example and Google think this will “impose the minimum necessary pain on the user, ideally zero, and lets [them] get to work”[60].

[59] https://developers.google.com/accounts/docs/CrossClientAuth
[60] http://googledevelopers.blogspot.co.uk/2013/05/cross-platform-sso-technology.html

Conclusion

The OAuth 2.0 main specification was ratified in October 2012 along with the bearer token specification[61]. In January 2013 a threat model document was completed and published[62] which helps implementors avoid potential attacks against an OAuth 2.0 authorisation server.

It is expected during 2013 that 7 additional specifications will be finalised. These specifications include methods to integrate SAML assertions with OAuth 2.0, encrypting requests with cryptographic signatures and allowing clients to automatically register and configure themselves with an authorisation server.

In addition, all of the new OpenID Connect standards[63] are based on the OAuth 2.0 specification resulting in interoperability and social extensions that can easily be added to any existing implementation. The OpenID Connect specifications are currently at "implementors draft review" stage and are anticipated to be published in 2013 as well.

In May 2013 Google unveiled a draft of their "Stronger Consumer Authentication - 5 year report."[64] In the document they commit themselves to the OAuth standard and a number of other upcoming standards such as ChannelID[65] to further improve security and consumer confidence.

> *"Google also makes heavy use of OAuth via bearer tokens. We will similarly be working with the standards community to make it easier for developers to leverage the OAuth standards."*

Microsoft has implemented OAuth 2.0 in the 2013 release of SharePoint which will result in enterprise organisations - included higher education institutions - being able to easily make use of the standard for their internally developed applications. When integrated with their Azure[66] cloud services, OAuth will be a key component in enterprise architectures along with existing highly deployed protocols such as LDAP proxy and ADFS.

Finally in May 2013, the OAuth 2.0 specification won the 2013 European Identity Award for Best Innovation/New Standard[67]. The judging panel concluded that "*OAuth 2.0 is the most important*

[61] http://datatracker.ietf.org/doc/draft-ietf-oauth-v2-bearer/
[62] http://datatracker.ietf.org/doc/draft-ietf-oauth-v2-threatmodel/
[63] http://openid.net/connect/
[64] https://docs.google.com/document/d/1r9qnZUehCbtkQR86Wp-sJR2Zu6sHx47queuqmegW2PY/
[65] http://tools.ietf.org/html/draft-balfanz-tls-channelid
[66] http://www.windowsazure.com/en-us/
[67] http://self-issued.info/?p=1026

new standard in the identity and access management field, which already enjoys broad recognition and support by the leading vendors on the market”.

With over 20 standards already published or almost complete, implementation by many of the largest companies on the Internet, and tens of code libraries (including those developed for the Linkey project) available for the developers to use, the OAuth ecosystem is looking very healthy.