Blog

API Security: Deep Dive into OAuth and OpenID Connect (https://nordicapis.com/api-security-oauth-openid-connect-depth/)

POSTED BY TRAVIS SPENCER (HTTPS://NORDICAPIS.COM/AUTHOR/TRAVISSPENCER/) | DECEMBER 5, 2014 SECURITY (HTTPS://NORDICAPIS.COM/API-INSIGHTS/SECURITY/)

Updated on March 19th, 2018

5

Facebook (https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/api-security-oauth-openid-connect-depth/&t=API Security: Deep Dive into OAuth and OpenID Connect)

33 Twitter

13

Google+ (https://plus.google.com/share?url=https://nordicapis.com/api-security-oauth-openid-connect-depth/)

257

LinkedIn (https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=API Security: Deep Dive into OAuth and OpenID Connect&url=https://nordicapis.com/api-security-oauth-openid-connect-depth/)

Reddit (http://reddit.com/submit?url=https://nordicapis.com/api-security-oauth-openid-connect-depth/&title=API Security: Deep Dive into OAuth and OpenID Connect)

Y HackerNews (https://news.ycombinator.com/submitlink?u=https://nordicapis.com/api-security-oauth-openid-connect-depth/&t=API Security: Deep Dive into OAuth and OpenID Connect)

Total: 309

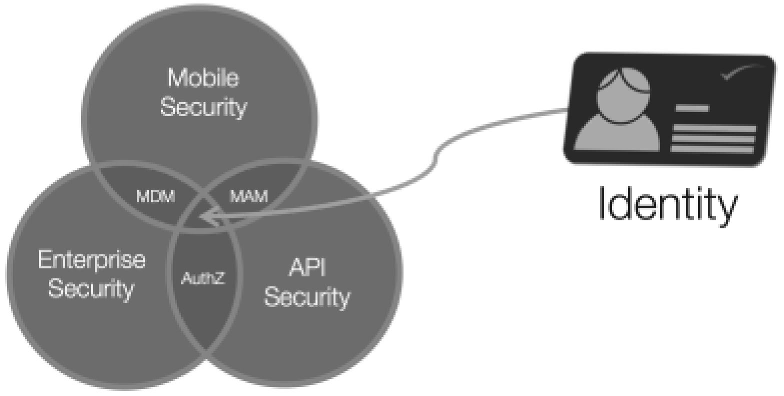
OAuth 2 and **OpenID Connect** are fundamental to securing your APIs. To protect the data that your services expose, you must use them. They are complicated though, so we wanted to go into some depth about these standards to help you deploy them correctly.

OAuth and OpenID Connect in Context

Always be aware that OAuth and OpenID Connect are part of a larger information security problem. You need to take additional measures to protect your servers and the mobiles that run your apps in addition to the steps taken to secure your API. Without a holistic approach, your API may be incredibly secure, your OAuth server locked down, and your OpenID Connect Provider tucked away in a safe enclave. Your firewalls, network, cloud infrastructure, or the mobile platform may open you up to attack if you don’t also strive to make them as secure as your API.

To account for all three of these security concerns, you have to know who someone is and what they are allowed to do. To authenticate and authorize someone on your servers, mobile devices, and in your API, you need a complete Identity Management System. At the head of

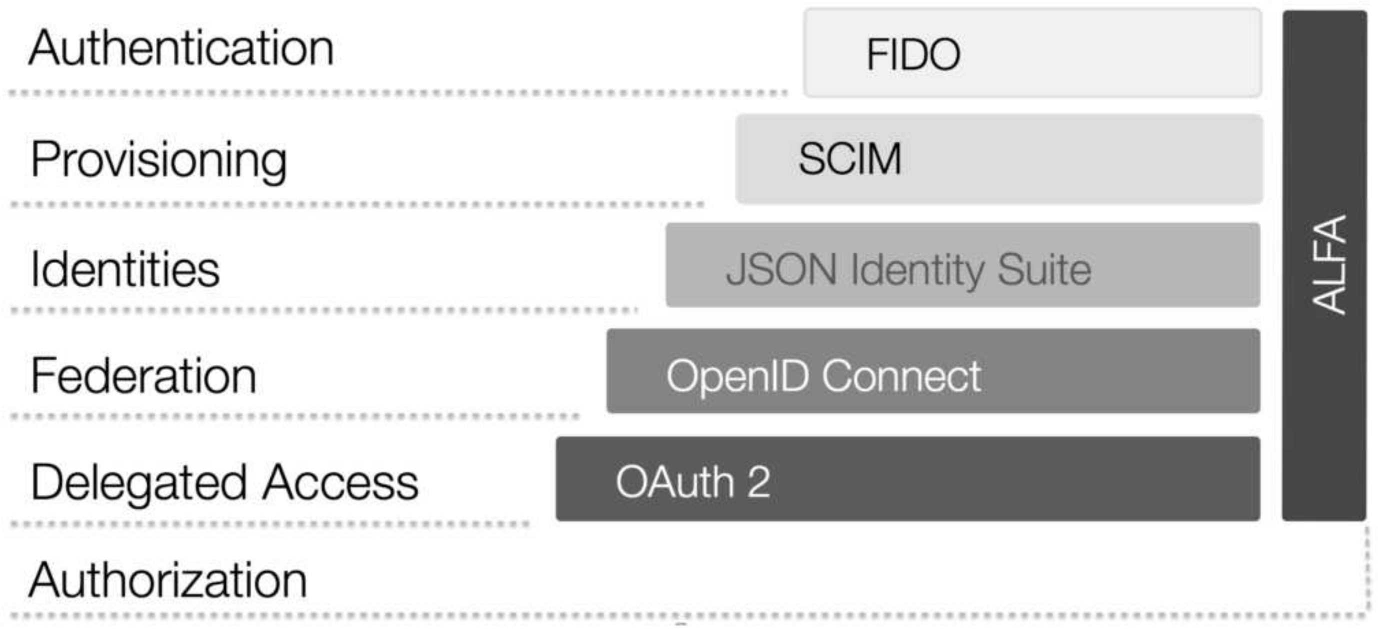
API security, enterprise security and mobile security is identity!



Only after you know who someone (or something) is can you determine if they should be allowed to access your data. We won't go into the other two concerns, but don't forget these as we delve deeper into API security.

Start with a Secure Foundation

To address the need for Identity Management in your API, you have to build on a solid base. You need to establish your API security infrastructure on protocols and standards that have been peer-reviewed and are seeing market adoption. For a long time, lack of such standards has been the main impediment for large organizations wanting to adopt RESTful APIs in earnest. This is no longer the case since the advent of the Neo-security Stack:



This protocol suite gives us all the capabilities we need to build a secure API platform. The base of this, OAuth and OpenID Connect, is what we want to go into in this blog post.

Overview of OAuth

OAuth is a sort of "protocol of protocols" or "meta protocol," meaning that it provides a useful starting point for other protocols (e.g., OpenID Connect (http://openid.net/specs/openid-connect-core-1_0.html), NAPS (http://openid.net/wg/napps/), and UMA (https://docs.kantarainitiative.org/uma/draft-uma-core.html)). This is similar to the way WS-Trust was used as the basis for WS-Federation, WS-SecureConversation, etc., if you have that frame of reference.

Beginning with OAuth is important because it solves a number of important needs that most API providers have, including:

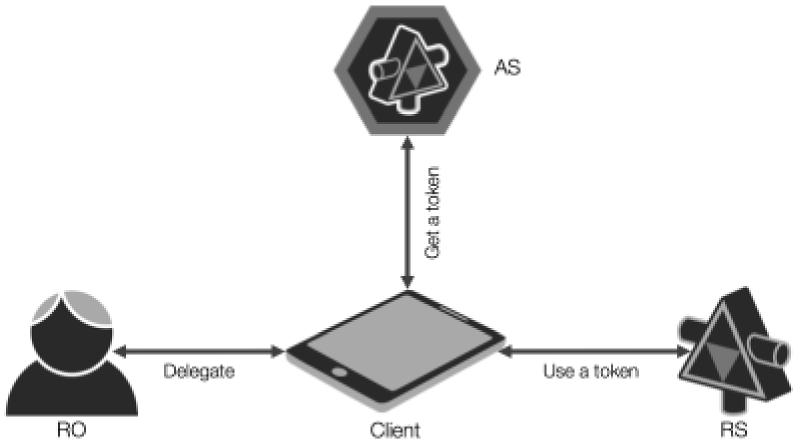
- Delegated access
- Reduction of password sharing between users and third-parties (the so called "password anti-pattern")

- Revocation of access

When the password anti-pattern is followed and users share their credentials with a third-party app, the only way to revoke access to that app is for the user to change their password. Consequently, all other delegated access is revoked as well. With OAuth, users can revoke access to specific applications without breaking other apps that should be allowed to continue to act on their behalf.

Actors in OAuth

There are four primary actors in OAuth:



1. **Resource Owner (RO)**: The entity that is in control of the data exposed by the API, typically an end user
2. **Client**: The mobile app, web site, etc. that wants to access data on behalf of the Resource Owner
3. **Authorization Server (AS)**: The Security Token Service (STS) or, colloquially, the OAuth server that issues tokens
4. **Resource Server (RS)**: The service that exposes the data, i.e., the API

Scopes

OAuth defines something called “scopes.” These are like permissions or delegated rights that the Resource Owner wishes the client to be able to do on their behalf. The client may request certain rights, but the user may only grant some of them or allow others that aren’t even requested. The rights that the client is requested are often shown in some sort of UI screen. Such a page may not be presented to the user, however. If the user has already granted the client such rights (e.g., in the EULA, employment contract, etc.), this page will be skipped.

What is in the scopes, how you use them, how they are displayed or not displayed, and pretty much everything else to do with scopes are not defined by the OAuth spec. OpenID Connect does define a few, but we'll get to that in a bit.

Kinds of Tokens

In OAuth, there are two kinds of tokens:

1. **Access Tokens**: These are tokens that are presented to the API
2. **Refresh Tokens**: These are used by the client to get a new access token from the AS

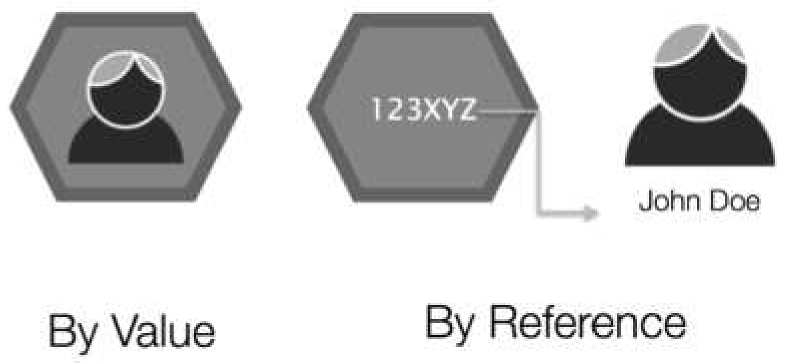
(Another kind of token that OpenID Connect defines is the ID token. We'll get to that in a bit.)

Think of access tokens like a session that is created for you when you login into a web site. As long as that session is valid, you can continue to interact with the web site without having to login again. Once that session is expired, you can get a new one by logging in again with your password. Refresh tokens are like passwords in this comparison. Also, just like passwords, the client needs to keep refresh tokens safe. It should persist these in a secure credential store. Loss of these tokens will require the revocation of all consents that users have performed.

NOTE: The AS may or may not issue a refresh token to a particular client. Issuing such a token is ultimately a trust decision. If you have doubts about a client's ability to keep these privileged tokens safe, don't issue it one!

Passing Tokens

As you start implementing OAuth, you'll find that you have more tokens than you ever knew what to do with! How you pass these around your system will certainly affect your overall security. There are two distinct ways in which they are passed:



1. By value
2. By reference

These are analogous to the way programming language pass data identified by variables. The run-time will either copy the data onto the stack as it invokes the function being called (by value) or it will push a pointer to the data (by reference). In a similar way, tokens will either contain all the identity data in them as they are passed around or they will be a reference to that data.

TIP: Pass by reference when tokens have to leave your network, and then convert them to by-value tokens as they enters your space. Do this conversion in your API gateway. See Jacob Ideskog's Microservices presentation (https://www.youtube.com/watch?v=BdKmZ7mPNns) for the Platform Summit for details.

If you pass your tokens by reference, keep in mind that you will need a way to dereference the token. This is typically done by the API calling a non-standard endpoint exposed by your OAuth server.

Profiles of Tokens

There are different profiles of tokens as well. The two that you need to be aware of are these:

1. Bearer tokens

2. Holder of Key (HoK) tokens

You can think of bearer tokens like cash. If you find a dollar bill on the ground and present it at a shop, the merchant will happily accept it. She looks at the issuer of the bill, and trusts that authority. The saleswomen doesn't care that you found it somewhere. Bearer tokens are the same. The API gets the bearer token and accepts the contents of the token because it trusts the issuer (the OAuth server). The API does not know if the client presenting the token really is the one who originally obtained it. This may or may not be a bad thing. Bearer tokens are helpful in some cases, but risky in others. Where some sort of proof that the client is the one to who the token was issued for, HoK tokens should be used.

HoK tokens are like a credit card. If you find my credit card on the street and try to use it at a shop, the merchant will (hopefully) ask for some form of ID or a PIN that unlocks the card. This extra credential assures the merchant that the one presenting the credit card is the one to whom it was issued. If your API requires this sort of proof, you will need HoK key tokens. This profile is still a draft (https://tools.ietf.org/html/draft-ietf-oauth-proof-of-possession), but you should follow this before doing your own thing.

NOTE: You may have heard of MAC tokens (https://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac) from an early OAuth 2 draft. This proposal was never finalized, and this profile of tokens are never used in practice. Avoid this unless you have a very good reason. Vet that rational on the OAuth mailing list (http://www.ietf.org/mail-archive/web/oauth/current/maillist.html) before investing time going down this rabbit trail.

Types of Tokens

We also have different types of tokens. The OAuth specification doesn't stipulate any particular type of tokens. This was originally seen by many as a negative thing. In practice, however, it's turned out to be a very good thing. It gives immense flexibility. Granted, this comes with reduced interoperability, but a uniform token type isn't one area where interop has been an issue. Quite the contrary! In practice, you'll often find tokens of various types and being able to switch them around enables interop. Example types include:

- WS-Security tokens, especially SAML tokens
- JWT tokens (which I'll get to next)
- Legacy tokens (e.g., those issued by a Web Access Management system)
- Custom tokens

Custom tokens are the most prevalent when passing them around by reference. In this case, they are randomly generated strings (https://www.owasp.org/index.php/Session_Management_Cheat_Sheet#Session_ID_Properties). When passing by val, you'll typically be using JWTs.

JSON Web Tokens

JSON Web Tokens or JWTs (pronounced like the English word "jot") are a type of token that is a JSON data structure that contains information, including:

- The issuer
- The subject or authenticated uses (typically the Resource Owner)
- How the user authenticated and when
- Who the token is intended for (i.e., the audience)

These tokens are very flexible, allowing you to add your own claims (i.e., attributes or name/value pairs) that represent the subject. JWTs were designed to be light-weight and to be snuggly passed around in HTTP headers and query strings. To this end, the JSON is split into different parts (header, body, signature) and base-64 encoded.

If it helps, you can compare JWTs to SAML tokens. They are less expressive, however, and you cannot do everything that you can do with SAML tokens. Also, unlike SAML they do not use XML, XML name spaces, or XML Schema. This is a good thing as JSON imposes a much lower technical barrier on the processors of these types of tokens.

JWTs are part of the JSON Identity Suite (http://www.slideshare.net/2botech/the-jsonbased-identity-protocol-suite), a critical layer in the Neo-security Stack. Other things in this suite include JWA for expressing algorithms, JWK for representing keys, JWE for encryption, JWS for signatures, etc. These together with JWT are used by both OAuth (typically) and OpenID Connect. How exactly is specified in the core OpenID Connect spec (http://openid.net/specs/openid-connect-core-1_0.html) and various ancillary specs, in the case of OAuth, including the Bearer Token spec (https://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer).

OAuth Flow

OAuth defines different “flows” or message exchange patterns. These interaction types include:

- The code flow (or web server flow)
- Client credential flow
- Resource owner credential flow
- Implicit flow

The code flow is by far the most common; it is probably what you are most familiar with if you’ve looked into OAuth much. It’s where the client is (typically) a web server, and that web site wants to access an API on behalf of a user. You’ve probably used it as a Resource Owner many times, for example, when you login to a site using certain social network identities. Even when the social network isn’t using OAuth 2 per se, the user experience is the same. Checkout this YouTube video at time 12:19 to see how this flow goes and what the end user experiences:

We'll go into the other flows another time. If you have questions on them in the meantime, ask in a comment below.

Improper and Proper Uses of OAuth

After all this, your head may be spinning. Mine was when I first learned these things. It's normally. To help you you orient yourself, I want to stress one really important high-level point:

- **OAuth is not used for authorization**. You might think it is from it's name, but it's not.
- **OAuth is also not for authentication**. If you use it for this, expect a breach if your data is of any value.
- **OAuth is also not for federation**.

So what is it for?

It's for delegation, and delegation only!

This is your plumb line. As you architect your OAuth deployment, ask yourself: In this scenario, am I using OAuth for anything other than delegation? If so, go back to the drawing board.

Consent vs. Authorization

How can it ***not*** be for authorization, you may be wondering. The "authorization" of the client by the Resource Owner is really consent. This consent may be enough for the user, but not enough for the API. The API is the one that's actually authorizing the request. It probably takes into account the rights granted to the client by the Resource Owner, but that consent, in and of its self, is not authorization.

To see how this nuance makes a very big difference, imagine you're a business owner. Suppose you hire an assistant to help you manage the finances. You ***consent*** to this assistant withdrawing money from the business' bank account. Imagine further that the assistant goes down to the bank to use these newly delegated rights to extract some of the company's capital. The banker would refuse the transaction because the assistant is not authorized – certain paperwork hasn't been filed, for example. So, your act of delegating your rights to the assistant doesn't mean squat. It's up to the banker to decide if the assistant gets to pull money out or not. In case it's not clear, in this analogy, the business owner is the Resource Owner, the assistant is the client, and the banker is the API.

Building OpenID Connect Atop OAuth

As I mentioned above, OpenID Connect builds on OAuth. Using everything we just talked about, OpenID Connect constrains the protocol, turning many of the specification's SHOULDs to MUSTs. This profile also adds new endpoints, flows, kinds of tokens, scopes, and more. OpenID Connect (which is often abbreviated OIDC) was made with mobile in mind. For the new kind of tokens that it defines, the spec says that they must be JWTs, which were also designed for low-bandwidth scenarios. By building on OAuth, you will gain both delegated access and federation capabilities with (typically) one product. This means less moving parts and reduced complexity.

OpenID Connect is a modern federation specification. It is a passive profile, meaning it is bound to a passive user agent that does not take an active part in the message exchange (though the client does). This exchange flows over HTTP, and is analogous to the SAML artifact flow (if that helps). OpenID Connect is a replacement for SAML and WS-Federation. While it is still relatively new, you should prefer it over those unless you have good reason not to (e.g., regulatory constraints).

As I've mentioned a few times, OpenID Connect defines a new kind of token: ID tokens. These are intended for the client. Unlike access tokens and refresh tokens that are opaque to the client, ID tokens allow the client to know, among other things:

- How the user authenticated (i.e., what type of credential was used)
- When the user authenticated
- Various properties about the authenticated user (e.g., first name, last name, shoe size, etc.)

This is useful when your client needs a bit of info to customize the user experience. Many times I've seen people use by value access tokens that contain this info, and they let the client take the values out of the API's token. This means they're stuck if the API needs to change the contents of the access token or switch to using by ref for security reasons. If your client needs data about the user, give it an ID token and avoid the trouble down the road.

The User Info Endpoint and OpenID Connect Scopes

Another important innovation of OpenID Connect is what's called the "User Info Endpoint." It's kinda a mouthful, but it's an ***extremely*** useful addition. The spec defines a few specific scopes that the client can pass to the OpenID Connect Provider or OP (which is another name for an AS that supports OIDC):

- openid (required)
- profile
- email
- address
- phone

You can also (and usually will) define others. The first is required and switches the OAuth server into OpenID Connect mode. The others are used to inform the user about what type of data the OP will release to the client. If the user authorizes the client to access these scopes, the OpenID Connect provider will release the respective data (e.g., email) to the client when the client calls the user info endpoint. This endpoint is protected by the access token that the client obtains using the code flow discussed above.

NOTE: An OAuth client that supports OpenID Connect is also called a Relying Party (RP). It gets this name from the fact that it ***relies*** on the OpenID Connect Provider to assert the user's identity.

Not Backward Compatible with v. 2

It's important to be aware that OpenID Connect ***is not*** backward compatible with OpenID 2 (or 1 for that matter). OpenID Connect is effectively version 3 of the OpenID specification. As a major update, it is not interoperable with previous versions. Updating from v. 2 to Connect will require a bit of work. If you've properly architected your API infrastructure to separate the concerns of federation with token issuance and authentication, this change will probably not disrupt much. If that's not the case however, you may need to update *each and every* app that used OpenID 2.

Conclusion

In this post, I dove into the fundamentals of OAuth and OpenID Connect and pointed out their place in the Neo-security Stack. I said it would be in depth, but honestly I've only skimmed the surface. Anyone providing an API that is protected by OAuth 2 (which should be all of them that need secure data access), this basic knowledge is a prerequisite for pretty much everyone on your dev team. Others, including product management, ops, and even project management should know some of the basics described above.

If you have questions beyond what I was able to cover here, checkout this video recording of a talk I gave on this topic (http://youtu.be/XGmUlyggXVo), this Slideshare presentation (http://www.slideshare.net/nordicapis/incorporating-oauth-how-to-integrate-oauth-into-your-mobile-app), or drop a comment below.

The Mobile/Enterprise/API Security Venn diagram was created by Gunnar Peterson (http://1raindrop.typepad.com/) and also used by permission.]*

5

Facebook (https://www.facebook.com/sharer/sharer.php?u=https://nordicapis.com/api-security-oauth-openid-connect-depth/&t=API Security: Deep Dive into OAuth and OpenID Connect)

33 Twitter

13

Google+ (https://plus.google.com/share?url=https://nordicapis.com/api-security-oauth-openid-connect-depth/)

257

LinkedIn (https://www.linkedin.com/shareArticle?mini=true&ro=true&trk=EasySocialShareButtons&title=API Security: Deep Dive into OAuth and OpenID Connect&url=https://nordicapis.com/api-security-oauth-openid-connect-depth/)

Reddit (http://reddit.com/submit?url=https://nordicapis.com/api-security-oauth-openid-connect-depth/&title=API Security: Deep Dive into OAuth and OpenID Connect)

HackerNews (https://news.ycombinator.com/submitlink?u=https://nordicapis.com/api-security-oauth-openid-connect-depth/&t=API Security: Deep Dive into OAuth and OpenID Connect)

Total: 309

ALFA (https://nordicapis.com/tag/alfa/), API security (https://nordicapis.com/tag/api-security/), Curity (https://nordicapis.com/tag/curity/), Curity.io (https://nordicapis.com/tag/curity-io/), FIDO (https://nordicapis.com/tag/fido/), JWT (https://nordicapis.com/tag/jwt/), Neo-security (https://nordicapis.com/tag/neo-security/), OAuth (https://nordicapis.com/tag/oauth/), OpenID Connect (https://nordicapis.com/tag/openid-connect/), scim (https://nordicapis.com/tag/scim/), Security (https://nordicapis.com/tag/security/), XACML (https://nordicapis.com/tag/xacml/)

33 Comments (https://nordicapis.com/api-security-oauth-openid-connect-depth/#disqus_thread)

About Travis Spencer

Founder & CEO of Curity (https://curity.io/) and a co-founder of Nordic APIs. An American living in Sweden and specializing in API security.

(https://nordicapis.com/author/travisspencer/)

(http://twitter.com/travisspencer)
(https://www.facebook.com/travislspencer)
(https://www.linkedin.com/in/travisspencer)
(https://plus.google.com/109819247814151405081)
(https://www.curity.io/)

Designing APIs for Machines (https://nordicapis.com/designing-apis-machines/)

Open Data: How to Make it... (https://nordicapis.com/open-data-how-to-make-it-work-for-your-business/)

33 Comments **Nordic APIs** 1 **Login**

Recommend 6 **Share** **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Paul Van Bladel • a year ago
A wonderful article, such a difficult subject brought on such an intuitive level !
Thanks.
2 • Reply • Share ›

Mario Tamaguchi • 7 months ago

Mario Yamaguchi • 7 months ago

What if in the scenario the RO does not exist per say.

The Client is calling the RS? Strictly a non-human event.

Client needs access to APIs on the RS.

The Client will be registered on the AS - but where does the authorization to the APIs reside. Does the AS list in JWT what APIs the Client has access on the RS? RS subsequently validates agaibst AS to comfy?

^ | ˅ • Reply • Share ›

Fede Lists • 9 months ago

"OAuth is also not for authentication. If you use it for this, expect a breach if your data is of any value." I definitely agree on the first sentence. I do not quite understand why you are referring to breaches in this context. What scenario do you have in mind ?

^ | ˅ • Reply • Share ›

travisspencer Mod ➦ Fede Lists • 8 months ago

It's a bit of hyperbole to sell newspapers, honestly, but I had this Facebook case in mind. To get the token like John was blogging about there, I was thinking about an old issue where FB's Android app wrote the access token to the logcat, thus allowing other apps to read it and impersonate the user. In general though, my point is this: if you mix up these fundamentals, then there will likely be many security issues with your architecture and implementation. Consequently, the likelihood of a breach will increase especially if your data is worth mounting a persistent attack.

^ | ˅ • Reply • Share ›

Fede Lists ➦ travisspencer • 8 months ago

Brilliant! Thanks. Just few days before you replied I had come across the very same article and went like "ah ah now I understand what he meant"

^ | ˅ • Reply • Share ›

Ranvijay Sachan • 2 years ago

How to use OpenID connect SSO with Django rest-framework?

How to use OpenID connect SSO with Django rest-framework?
In my current flow, We are using Django rest-framework login, now we need to integrate OpenID connect SSO in our project so guide me how we can integrate OpenID connect SSO with Django-rest framework.

^ | ˅ • Reply • Share ›

Daniel Lindau ➔ Ranvijay Sachan • 2 years ago

We have created an OpenID Connect example application using Python Flask, and you might find inspiration there. Most of the Flask code is factored out, so it should be fairly easy to create a Django.

The repo is here, and the code is under Apache 2 license.
https://github.com/curityio...

The client part of it:
https://github.com/curityio...

Hope it helps!

Regards,
Daniel Lindau, Twobo Technologies

^ | ˅ • Reply • Share ›

Ranvijay Sachan ➔ Daniel Lindau • 2 years ago

Thanks Daniel Lindau I have also created a sample code please go through the link
http://stackoverflow.com/qu...

^ | ˅ • Reply • Share ›

Preben Nilsson • 2 years ago

Hi Travis.

I have read your blog and watched the video. But as a newcomer to OpenId Connect and Oauth2, I have a hard time figuring out the "full picture", where both OpenId connect is in place to authenticate a user (ie making sure that I am who I say I am) and OAuth2 to allow me to access my data at a resource server.

Do you have a combined picture showing both technologies in play at the same time?

I think I get the autorization code flow, where I can do redirects to an OpenId provider (eg. Facebook) to identify myself and then getting redirected back with a token.

But imagine that I am building a stock trader application, that needs information from eg. Bloomberg. Both my application and Bloomberg trust OpenId Connect from Facebook to identify users. So when my users log in to my stock trader application, I will redirect them to Facebook, where they can log in and eventually get redirected back to my site.

When I/the user need to access the Bloomberg services, how do I do that? Do I pass an OpenId token to them, or will they need to have an OAuth2 token endpoint, so that I can authorize the user to the service?

Best regards
Preben

• Reply • Share ›

Jaap Francke • 2 years ago

I very much like this overview of OAuth and OIDC. Specifically the explanation around token types, kinds and profiles gave me better understanding.

I need to think a bit more about your statement on consent versus authorisation. In my opinion the common definition of authorisation is the process of specifying or granting a right to someone. See https://en.wikipedia.org/wi.... Quote: "The access control process can be divided into the following two phases: 1) policy definition phase where access is authorized, and 2) policy enforcement phase where access requests are approved or disapproved. Authorization is thus the function of the policy definition phase which precedes the policy enforcement phase "

In your example, I think the business owner is authorizing the assistent to do financial transactions. The banker is doing the enforcement of rights (authorisations) on financial transactions. So OAuth (to me) is a means of informing the ResourcesServer about an right/authorisation so the RS can enforce them properly. (I think XACML has a refined model for access policies, but that's a topic of it's own).

I'm curious to see your response.

• Reply • Share ›

travisspencer Mod Jaap Francke • 2 years ago

Thanks, **@Jaap Francke**, for the comment. You aren't along in your thinking, so, as you noodle on this, feel free to disagree with me, but I hold tightly to my opinion.

Firstly, I completely agree with that definition of what authorization is. What and who though is defining the policy and what and who is enforcing it? Those are the questions here. IMO, it's the API (i.e., the Resource Server or analogously, the banker) that's both defining and enforcing policy. The banker sets the policy of who can transfer corporate funds and checks that the person requesting funds is allowed to do so for that corporate account. This policy is not up to the business owner, though the business owner can provide inputs used by the banker to arrive at its decision. If it was up to the owner, I'd hire you, "authorize" you to take $1M out of the corp. account, and we'd run away to the Bahamas, skewing all the other owners despite their objection to my "authorization!" It doesn't work like that in the real-world, and it doesn't work like that with OAuth. The OAuth server can't dictate authorization policy for a resource that isn't under it's control. The API controls/governs the resource and so it sets the policy for access; it also makes the decision about releasing that data. (The API could actually be relived of this role, and IMO it should; instead, using centrally authored policies which it enforces using an embedded PDP, but let's take that up in a separate blog post or a subsequent comment.)

see more

• Reply • Share ›

Michael Pöttgen travisspencer • a year ago

What you can say however is that even if the resource server may incorporate a more sophisticated mechanism to decide what to do, if a client presents an access token with a scope of "withdraw_money", is that the resource server (banker) should not give out any money, if such an access token is not presented. What the scope "withdraw_money" actually means is totally outside of the scope of OAuth 2.0. It could mean that the client is authorized to withdraw $10,000 a month, it could mean that the bank also needs written permission by the bank account owner with a set limit. But without that token, the client would get nothing, unless he is using a different protocol. Therefore OAuth 2.0 does contribute to the authorization process. In OpenID Connect it contributes to the question, whether a client should get access to particular information about the user. And even if you use OpenID Connect to provide the Identity of the user only and nothing else, then part of your authorization process is that you require a user to be identifiable.

^ | ˅ • Reply • Share ›

DJ • 2 years ago

I believe the Authorization Code flow should *not* be used with in a browser-based AngularJS application, because it cannot securely store a client secret. That leaves us with the Implicit Flow, where there is no refresh token, I believe. How does one implement an Implicit flow and still allow user interaction in the browser to extend the session (to avoid redirecting a user to the AS after a fixed time)?

^ | ˅ • Reply • Share ›

travisspencer Mod ➦ DJ • 2 years ago

There are 3 ways:

1. Embed the client secret in the HTML/JavaScript of the SPA, so you can use the code flow and get a refresh token. On the back-end, in the APIs, treat this client's access tokens with skepticism. Restrict access of what these tokens can do based on scopes. Don't let this client get tokens with powerful scopes. In other words, treat this client as a public, non-confidential client even though it has a "secret".

2. Issue many tokens each with different scopes. Make sure tokens with powerful scopes expire quickly and weak ones do not. In the SPA's back-end, store all these tokens and associate them with some sort of session cookie that the AngularJS front end will maintain. When the front-end makes calls via the back-end, find the right token from the session cache and forward the call with the proper weak/powerful token. If a powerful token is gone and the user is trying to do some high-value action, prompt them to login. If they are trying to do something of low-value, it'll be fine to use the weak token and so the user won't have to login again.

3. In the OAuth server, issue tokens who's power degrades over time. This is the ideal solution because the SPA stays simple, there's 1 token in the mix, the SPA doesn't need a

solution because the SPA stays simple, there's 1 token in the mix, the SPA doesn't need a back-end to map a bunch of tokens, etc. It does requires explicit support in the AS though, and the only product that I know that has support for this OOTB is Curity.

If you have followup questions, please feel free to write back here or email me. Will try to reply quicker next time ;-)

^ | ˅ • Reply • Share ›

Kourosh Vorell • 2 years ago

Is it safe to provide an OAuth Access Token inside the URI? Can an attacker use the Access Token and establish a session or call the APIs prior to the genuine client using the Access Token? Should the Access Token be only passed in the body of an HTTPS request/response?

^ | ˅ • Reply • Share ›

travisspencer Mod ➦ Kourosh Vorell • 2 years ago

> Is it safe to provide an OAuth Access Token inside the URI?

Maybe, maybe not. If there are no intermediaries between the Client and the RS (the API), then there is not much risk of including it in the query string. This usually isn't the case, however, and there tends to be *many* proxies between the two. These look at the headers of an HTTP request, but not the body (for performance sake). Therefore, the token won't accidentally get logged by these intermediaries (e.g., reverse proxies, load balancers, SSL terminators, etc.) if you send it in the body. That's why the general best practice is to the POST the token around or include it in an Authorize HTTP header.

> Can an attacker use the Access Token and establish a session or call the APIs prior to the genuine client using the Access Token?

Prior doesn't matter so much because the AT can be used repeatedly. The AC is a nonce, so that can only be used once. In either case, obtaining the AT or AC will allow the attacker to assume the identity of the victim, the RO, so keep 'em safe!

> Should the Access Token be only passed in the body of an HTTPS request/response?

Use the Authorize header or a FORM POST. Refer to RFC 6750 for more details or post a reply here.

^ | v • Reply • Share ›

Kourosh Vorell → travisspencer • 2 years ago

Thanks for the reply. Yes, RFC 6750 is very clear on this point, The Access Token (bearer token) SHOULD NOT be passed in page URL's and it SHOULD be passed in HTTPS header line or body (sections 3.2 and 5.3 of the RFC). It is rather surprising that various implementations are still passing the Access Token on the URI. Do we know which implementations (OAUTH libraries) are safe and are passing the Access Token in the Authorization header or body and not in the URI? Thanks

^ | v • Reply • Share ›

travisspencer Mod → Kourosh Vorell • 2 years ago

There are too many libs to say. Even ones that support sending ATs in the URI aren't necessarily insecure (e.g., when there are no intermediaries between the client and API). So, it's up to the user to know what they're doing with the lib. Good that you know now ;-)

^ | v • Reply • Share ›

Shaun Forgie • 3 years ago

The section on ID Tokens needs clarification. You mention that OpenID-connect allows the client to read some personal information out of the token. But are you recommending the use of these or not?

^ | v • Reply • Share ›

travisspencer Mod → Shaun Forgie • 3 years ago

> If your client needs data about the user, give it an ID token

The use of ID tokens comes down to this. If the client (i.e., Web or mobile app) needs to know _any_ information about the user (e.g., how they logged in, their first name, a customer ID, etc.), you should issue an ID token. Also, if the client needs different (typically less) information about the user than the API, then ID tokens are needed and the access token (presented to the API) should contain (or refer to) additional user-related data.

HTH but if not reply with more questions.

^ | ˅ • Reply • Share ›

Shaun Forgie ➦ travisspencer • 3 years ago

Which spec makes the distinction between ID Tokens and Access Tokens ? In the case of the typical code flow scenario I thought the design intent was for the API (relying party) and not the client to be able to access Identity information via the User Info endpoint by sending a request that contained the access token. But I do understand that, if requested, bits of a users identity can be embedded into the token, if the OpenID provider is asked by the Relying Party to do so. But this is still referred to as an Access Token correct?

^ | ˅ • Reply • Share ›

travisspencer Mod ➦ Shaun Forgie • 3 years ago

> Which spec makes the distinction between ID Tokens and Access Tokens ?
ID tokens are defined in OpenID Connect. Access Tokens are defined in OAuth 2.

> ... the API (relying party) ...
The API isn't the Relying Party, the client is. Client in OAuth == RP in OpenID Connect.

While we're on definitions, I didn't mention this one in the article:

OP == OpenID Connect Provider

Also, Access Token is normally abbreviated AT and ID Token is shortened to IDT (as I do below).

> I thought the design intent was for the API [...] and not the
client to be able to access Identity information via the User Info
endpoint by sending a request that contained the access token.

Either the client or API can use the AT for this purpose

see more

^ | v • Reply • Share ›

Ricardo Villanueva • 3 years ago

Really good explanation, I'm starting to build Neo-Security Stack and this has been an excellent reference. Thank you.

^ | v • Reply • Share ›

travisspencer Mod → Ricardo Villanueva • 3 years ago

Excellent, **@Ricardo Villanueva**. Glad it was helpful. BTW, Neo-sec is all we do @2botech. Contact me if you're looking to make it your full-time gig.

^ | v • Reply • Share ›

Filippos Vasilakis • 4 years ago

Excellent introduction. Looking more into neo-stack security these days. But it turns out, if you want to build such a GOOD api you need to spend time (=money) since you will actually build the stack on your own (which might lead into security holes). Any frameworks/libs that ease the development of such stack ?

^ | v • Reply • Share ›

travisspencer Mod → Filippos Vasilakis • 4 years ago

The Neo-security Stack is a few hundred pages of PDFs. When you start, you have the option of using non-standards, other standards than these, or the ones I mentioned. My point is that you should use the Neo-security Stack of specifications and not any competing alternative.

From these, you can assemble a half-dozen (or more) "candidate architectures" (i.e., you can assemble them together in all sorts of ways). The one I'd suggest you follow is what we (at Twobo) call the "Neo-security Platform." I talked about this architecture at another Nordic APIs event:

see more

^ | v • Reply • Share ›

Filippos Vasilakis ➦ travisspencer • 4 years ago

Interesting! Thanks!

^ | v • Reply • Share ›

Niall Mcloughlin • a year ago

The piece I'm missing is the trust between the API provide and the OP. At 14.27 the OpenID Connect flow has completed and the client holds the access token to forward it to the API. Is there a SAML like out of bound configuration swap between the OP and the API provider such that when the access token arrives, the API service trusts the OP ? Does the API service have to have a local representation of the unique identifier for the user presented the access token before authentication is successful (again, SAML like) meaning there's an account management requirement on the API service.

^ | v • Reply • Share ›

travisspencer Mod ➦ Niall Mcloughlin • a year ago

Usually, an API will only trust its own OAuth Authorization Server (AS) or OpenID Connect Provider (OP). Those will require authentication. Authentication may be done by some upstream authentication provider, but that is unbeknownst to the API. The API simply trusts its own AS; it won't accept any token that isn't issued by its own Identity Management System (IMS), of which the AS/OP is a part. This keeps the systems decoupled. In fact, the AS/OP shouldn't do the authentication either. That should be handled by a dedicated Authentication Service. This is because login logic, self-service, account linking, etc. are very different concerns then token issuance, which the AS/OP are focused on. In this way, you end up with an architecture like this:

Does this clarify things, Niall?

1 ^ | ˅ • Reply • Share ›

Niall Mcloughlin ➦ travisspencer • a year ago

Hi Travis. Thank you for taking the time to respond. That certainly helps. The use case I'm trying to work through is a native app or single page app that leverages an IDaaS (such as Okta) for authentication and entitlement to the app. That app leverages two or more resource providers that expose APIs that support OIDC. I'm trying to put together the OAuth2/OpenID connect authentication flows for that model but I'm coming up with a knowledge gap. Twi follow up questions if I may; Is the trust establishment in your diagram above between the Authentication Provider and the Identity Management system through dynamic discovery (trust anyone) or exchange of configuration data similar to SAML between IdP and SP ? And lastly does the API service have to have a local representation of an authenticating account (sub) before succesful authentication meaning account management is required at the API service or is a Just In Time approach support (again, similar to SAML).

^ | ˅ • Reply • Share ›

Emanoel Xavier • 3 years ago

Great Article! I am in the process of designing a simple Rest API that will be used by some various clients (native, mobile, web). The high level functionality of the API will be allowing users to sign up (create an account), create some data associated with that account and retrieve it. I am planning to leverage OIDC implemented by any OP out there. So basically the clients will do the OIDC handshake with some identity provider (Google, for instance) then call my API (to create account, create data or retrieve data) providing the id token issued by the OIDC. My service would then validate the token, make sure the client is a known client (I believe this info is in the token), take the identity of the user from the token and from that point on do what it must (create an account associated with that identity, create data associated with that account - inferred from the identity, or retrieve the data if belong to that user). In this case the id token is purely used to identify who that user is (within the identity provider) and map that identity to my own account. The id token would be passed from the client (again mobile, native or web) to my API.
I have not seen many uses of OIDC like that out there so I was wondering if this would be a valid flow or if I am violating the usage of the protocol. Thoughts?

• Reply • Share ›

Shawn ODonnell ➦ Emanoel Xavier • 3 years ago

That sounds like a good approach. Were you successful? Did you use open source

(https://nordicapis.com/events/the-2018-platform-summit/)

(https://nordicapis.com/best-public-api-of-2018/)

(https://nordicapis.com/events/livecast-the-role-of-identity-in-api-security/)

SMARTER TECH DECISIONS USING APIS

Subscribe to our mailing list

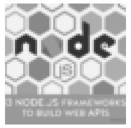
Subscribe

POPULAR POSTS

(https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/) 7 Frameworks To Build A REST API In Go (https://nordicapis.com/7-frameworks-to-build-a-rest-api-in-go/)
by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on July 4, 2017

(https://nordicapis.com/5-lightweight-php-frameworks-build-rest-apis/) 5 Lightweight PHP Frameworks to Build REST APIs (https://nordicapis.com/5-lightweight-php-frameworks-build-rest-apis/)
by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on May 4, 2017

(https://nordicapis.com/best-practices-api-error-handling/) Best Practices for API Error Handling (https://nordicapis.com/best-practices-api-error-handling/)
by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on June 15, 2017


(https://nordicapis.com/13-node-js-frameworks-to-build-web-apis/) 13 Node.js Frameworks to Build Web APIs (https://nordicapis.com/13-node-js-frameworks-to-build-web-apis/)
by Kristopher Sandoval (https://nordicapis.com/author/sandovaleffect/) | posted on October 26, 2017

(https://nordicapis.com/a-pragmatic-take-on-rest-anti-patterns/) A Pragmatic Take On REST Anti Patterns (https://nordicapis.com/a-pragmatic-take-on-rest-anti-patterns/)
by Chris Wood (https://nordicapis.com/author/chriswood/) | posted on July 24, 2018

Search

Search

RECENT POSTS

How APIs Make Marketing Data Analysis and Reporting Infinitely Easier (https://nordicapis.com/how-apis-make-marketing-data-analysis-and-reporting-infinitely-easier/)

7 Steps For Building Successful API Products (https://nordicapis.com/7-steps-for-building-successful-api-products/)

Public Awareness for APIs Sucks. Here's What We Can Do. (https://nordicapis.com/public-awareness-for-apis-sucks-heres-what-we-can-do/)

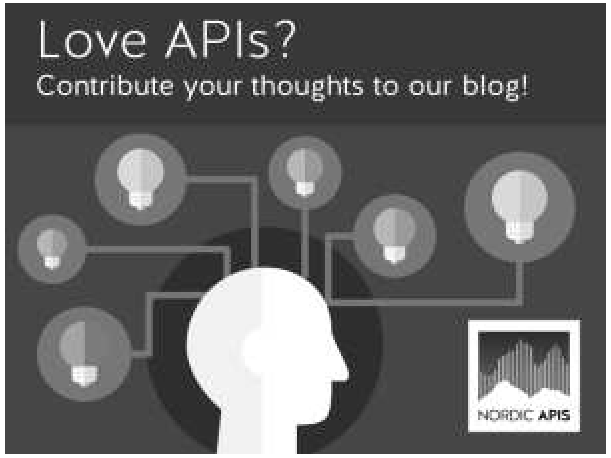
A Pragmatic Take On REST Anti Patterns (https://nordicapis.com/a-pragmatic-take-on-rest-anti-patterns/)

Case Study: How Rakuten RapidAPI Is Globalizing The API Marketplace (https://nordicapis.com/case-study-rakuten-rapidapi-is-globalizing-the-api-marketplace/)

SUBSCRIBE TO OUR FEED

Nordic APIs RSS (http://nordicapis.com/feed/)

CREATE WITH US



(https://docs.google.com/a/twobotechnologies.com/forms/d/12Ng9A_QKUjmAHDgv8Pxb4uLIkECGJawV3vwAWJ4WxTs/viewform)

TWITTER (HTTPS://TWITTER.COM/NORDICAPIS)

FACEBOOK (HTTPS://WWW.FACEBOOK.COM/NORDICAPIS)

YOUTUBE (HTTPS://WWW.YOUTUBE.COM/USER/NORDICAPIS)

SLIDESHARE (HTTP://WWW.SLIDESHARE.NET/NORDICAPIS)

INSTAGRAM (HTTPS://WWW.INSTAGRAM.COM/NORDICAPIS/)

RSS (HTTP://NORDICAPIS.COM/FEED/)