How is OAuth 2 different from OAuth 1?

Asked 10 years, 3 months ago Active 8 months ago Viewed 230k times

625

In very simple terms, can someone explain the difference between OAuth 2 and OAuth 1?

Is OAuth 1 obsolete now? Should we be implementing OAuth 2? I don't see many implementations of OAuth 2; most are still using OAuth 1, which makes me doubt OAuth 2 is ready to use. Is it?

224

oauth oauth-2.0 authorization

Share Improve this question Follow

edited Apr 13 '19 at 2:51
NM Pennypacker
5,939 11 30 36

asked Nov 6 '10 at 16:18
sullivan
6,303 3 13 8

You may found your answer here OAuth 2.0 - Overview – John Joe Feb 23 '17 at 2:07

10 Answers

Active | Oldest | Votes

545

Eran Hammer-Lahav has done an excellent job in explaining the majority of the differences in his article Introducing OAuth 2.0. To summarize, here are the key differences:

More OAuth Flows to allow better support for non-browser based applications. This is a main criticism against OAuth from client applications that were not browser based. For example, in OAuth 1.0, desktop applications or mobile phone applications had to direct the user to open their browser to the desired service, authenticate with the service, and copy the token from the service back to the application. The main criticism here is against the user experience. With OAuth 2.0, there are now new ways for an application to get authorization for a user.

OAuth 2.0 no longer requires client applications to have cryptography. This hearkens back to the old Twitter Auth API, which didn't require the application to HMAC hash tokens and request strings. With OAuth 2.0, the application can make a request using

only the issued token over HTTPS.

OAuth 2.0 signatures are much less complicated. No more special parsing, sorting, or encoding.

OAuth 2.0 Access tokens are "short-lived". Typically, OAuth 1.0 Access tokens could be stored for a year or more (Twitter never let them expire). OAuth 2.0 has the notion of refresh tokens. While I'm not entirely sure what these are, my guess is that your access tokens can be short lived (i.e. session based) while your refresh tokens can be "life time". You'd use a refresh token to acquire a new access token rather than have the user re-authorize your application.

Finally, OAuth 2.0 is meant to have a clean separation of roles between the server responsible for handling OAuth requests and the server handling user authorization. More information about that is detailed in the aforementioned article.

Share Improve this answer Follow

edited Apr 24 '17 at 10:07

Riyaz Mohammed Ibrahim

9,257 5 24 33

answered Nov 8 '10 at 17:02

villecoder

13k 2 27 46

2 Could anyone clarify how callback urls are different between oauth 1 and 2? – Brian Armstrong Jun 16 '11 at 21:58

2 OAuth 2.0 will only obselete OAuth if it is approved as an RFC. Currently it is an Internet Draft, but it is planned to become an Internet Standard (as far as these things can be planned). However, this will take years, since large parts of the framework is not yet specified. We will probably see a whole family of Internet Drafts being published the coming years, each one concerning different aspects of the OAuth 2.0 authorization framework. To see why this is true, visit tools.ietf.org/html/draft-ietf-oauth-v2, and search for "beyond the scope of this specification" ;) – Håvard Geithus Jul 4 '12 at 17:01

48 The author of the article wrote a follow-up last year called "OAuth 2.0 and the Road to Hell", which can be read here: web.archive.org/web/20120731155632/http://hueniverse.com/2012/… A significant difference in the two are security - as foreshadowed by the lack of cryptography in 2.0. – kdazzle Jun 6 '13 at 20:15

4 Security of OAuth 1.0 relies on the assumption that a secret key embedded in a client application can be kept confidential, but the assumption is naive. In OAuth 2.0, such a naive client application is called *confidential client*. There is no practical difference in security level between OAuth 1.0 clients and OAuth 2.0 confidential clients. "OAuth 2.0 and the Road to Hell" misses this point. – Takahiko Kawasaki Apr 11 '16 at 16:40

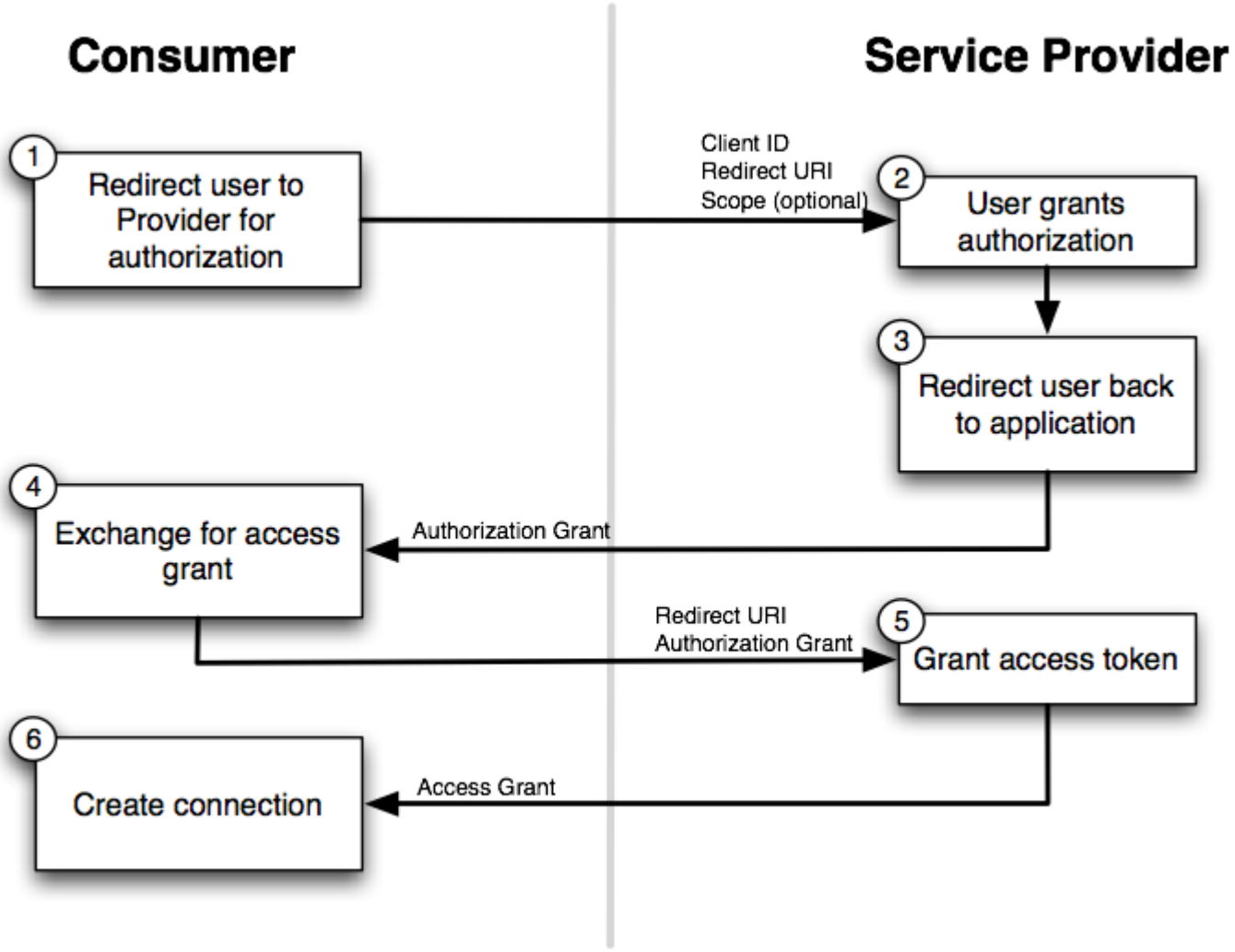
6 @kdazzle, that link has now moved to here: hueniverse.com/oauth-2-0-and-the-road-to-hell-8eec45921529 – e_i_pi May 24 '17 at 3:00

196

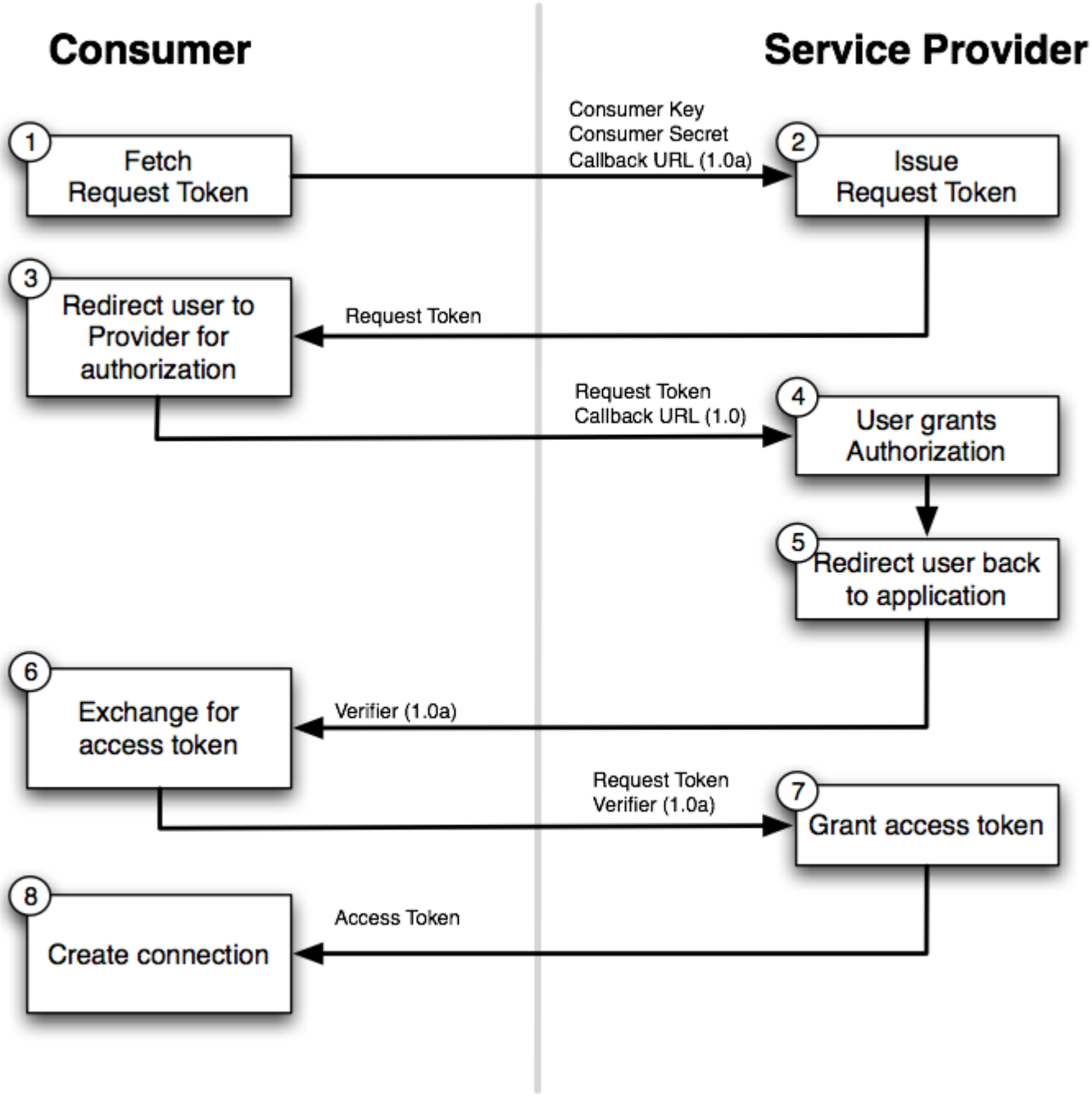
I see great answers up here but what I miss were some diagrams and since I had to work with Spring Framework I came across their explanation.

I find the following diagrams very useful. They illustrate the difference in communication between parties with OAuth2 and OAuth1.

OAuth 2



OAuth 1



Share Improve this answer Follow

answered Dec 13 '14 at 12:32

nyxz

5,750 8 43 61

4 where is "client_secret" used in this flow ?? – ashwintastic Mar 8 '16 at 6:34

3 If you mean the secret the user enters when redirected to the provider (say Facebook, Twitter, Google, etc.) then this would be step 2 for `OAuth 2` and step 4 for `OAuth 1` . – nyxz May 19 '16 at 10:53

Why do both diagrams have a Service Provider step called "User grants Authorization?" This seems backwards or wrong. Isn't the "user" the one seeking authorization? – Forbin Feb 18 '19 at 19:43

@Forbin Because this step happens on the Service Provider's side. You are at their page where you see the grants the service requires from you and you have to agree to share this information with the service you are trying to authenticate to. StackOverflow actually have the option to login using Google account. It works the same way. SO will ask Google to view your email and you have to agree on that. – nyxz Feb 19 '19 at 9:16

99

The previous explanations are all overly detailed and complicated IMO. Put simply, OAuth 2 delegates security to the HTTPS protocol. OAuth 1 did not require this and consequentially had alternative methods to deal with various attacks. These methods required the application to engage in certain security protocols which are complicated and can be difficult to implement. Therefore, it is simpler to just rely on the HTTPS for security so that application developers dont need to worry about it.

As to your other questions, the answer depends. Some services dont want to require the use of HTTPS, were developed before OAuth 2, or have some other requirement which may prevent them from using OAuth 2. Furthermore, there has been a lot of debate about the OAuth 2 protocol itself. As you can see, Facebook, Google, and a few others each have slightly varying versions of the protocols implemented. So some people stick with OAuth 1 because it is more uniform across the different platforms. Recently, the OAuth 2 protocol has been finalized but we have yet to see how its adoption will take.

Share Improve this answer Follow

answered Oct 30 '12 at 19:00

chacham15

12.2k 19 86 182

12 So basically OAuth2 works with HTTPS and therefore is simpler than OAuth1 which needs to be a bit more complex since it can work without HTTPS? – Micro Dec 28 '15 at 3:58

@MicroR This is one practical definition you got over there! ;) – EralpB Mar 7 '17 at 8:13

Note there are serious security arguments against using Oauth 2:

38

[one bleak article](#)

[and a more technical one](#)

Note these are coming from Oauth 2's lead author.

Key points:

- Oauth 2 offers no security on top of SSL while Oauth 1 is transport-independent.
- in a sense SSL isn't secure in that the server does not verify the connection and the common client libraries make it easy to ignore failures.

  > The problem with SSL/TLS, is that when you fail to verify the certificate on the client side, the connection still works. Any time ignoring an error leads to success, developers are going to do just that. The server has no way of enforcing certificate verification, and even if it could, an attacker will surely not.

- you can fat-finger away all of your security, which is much harder to do in OAuth 1.0:

  > The second common potential problem are typos. Would you consider it a proper design when omitting one character (the ‘s’ in ‘https’) voids the entire security of the token? Or perhaps sending the request (over a valid and verified SSL/TLS connection) to the wrong destination (say ‘http://gacebook.com’?). Remember, being able to use OAuth bearer tokens from the command line was clearly a use case bearer tokens advocates promoted.

Share Improve this answer Follow

edited Jan 26 '18 at 21:18

harmv

1,570 18 17

answered Mar 1 '13 at 22:04

djechlin

53.7k 28 140 261

4 the "typo" argument is not very valid - it's common practise to redirect from http to https – Oleg Mikheev Feb 20 '15 at 10:14

4 @OlegMikheev Yeah, but it takes only one http (no-s) request to allow a MITM to sniff your headers and your token is now being used by someone else! – Patrick James McDougle Mar 24 '15 at 5:50

4 if by headers you mean cookies then they are supposed to be secure. Other than that I don't see how user typo (in browser URL) can expose tokens, they are not even supposed to be in headers – Oleg Mikheev Mar 24 '15 at 9:40

4 As an additional point against the "typo" argument, a service provider can reject any OAuth 2.0 requests that are not through https and revoke the access token in that request. – skeller88 Oct 21 '15 at 20:46

36

Security of the OAuth 1.0 protocol (RFC 5849) relies on the assumption that a secret key embedded in a client application can be kept confidential. However, the assumption is naive.

In OAuth 2.0 (RFC 6749), such a naive client application is called a *confidential* client. On the other hand, a client application in an environment where it is difficult to keep a secret key confidential is called a *public* client. See 2.1. Client Types for details.

In that sense, OAuth 1.0 is a specification only for confidential clients.

"OAuth 2.0 and the Road to Hell" says that OAuth 2.0 is less secure, but there is no practical difference in security level between OAuth 1.0 clients and OAuth 2.0 confidential clients. OAuth 1.0 requires to compute signature, but it does not enhance security if it is already assured that a secret key on the client side can be kept confidential. Computing signature is just a cumbersome calculation without any practical security enhancement. I mean, compared to the simplicity that an OAuth 2.0 client connects to a server over TLS and just presents `client_id` and `client_secret`, it cannot be said that the cumbersome calculation is better in terms of security.

In addition, RFC 5849 (OAuth 1.0) does not mention anything about open redirectors while RFC 6749 (OAuth 2.0) does. That is, `oauth_callback` parameter of OAuth 1.0 can become a security hole.

Therefore, I don't think OAuth 1.0 is more secure than OAuth 2.0.

[April 14, 2016] Addition to clarify my point

OAuth 1.0 security relies on signature computation. A signature is computed using a secret key where a secret key is a shared key for HMAC-SHA1 (RFC 5849, 3.4.2) or a private key for RSA-SHA1 (RFC 5849, 3.4.3). Anyone who knows the secret key can compute the signature. So, if the secret key is compromised, complexity of signature computation is meaningless however complex it is.

This means OAuth 1.0 security relies not on the complexity and the logic of signature computation but merely on the confidentiality of a secret key. In other words, what is needed for OAuth 1.0 security is only the condition that a secret key can be kept confidential. This may sound extreme, but signature computation adds no security enhancement if the condition is already satisfied.

Likewise, OAuth 2.0 **confidential** clients rely on the same condition. If the condition is already satisfied, is there any problem in creating a secure connection using TLS and sending `client_id` and **`client_secret`** to an authorization server through the secured

connection? Is there any big difference in security level between OAuth 1.0 and OAuth 2.0 confidential clients if both rely on the same condition?

I cannot find any good reason for OAuth 1.0 to blame OAuth 2.0. The fact is simply that (1) OAuth 1.0 is just a specification only for confidential clients and (2) OAuth 2.0 has simplified the protocol for confidential clients and supported **public** clients, too. Regardless of whether it is known well or not, smartphone applications are classified as public clients (RFC 6749, 9), which benefit from OAuth 2.0.

Share Improve this answer Follow

edited Sep 24 '18 at 12:38
harmv
1,570 18 17

answered Mar 3 '16 at 14:34
Takahiko Kawasaki
15.3k 7 53 95

9 Sending secrets instead of signatures, whether through HTTP, HTTPS, etc., will always carry an implicit security risk because of MITM at the protocol level. Now there's 2 ways to find secrets instead of just 1: root the device, *or* forge root certs (has happened before, so not far-fetched). When your security model is "eh, let transport handle it," it's true that it won't be any LESS secure than the protocol. But monolithic security models == one point of entry for many services. It's "good enough" for pragmatic engineers, but it'll never be "as secure" as an alternative decentralized model. – Mark G. Oct 18 '16 at 14:05

25

OAuth 2 is apparently a waste of time (from the mouth of someone that was heavily involved in it):

https://hueniverse.com/oauth-2-0-and-the-road-to-hell-8eec45921529

He says (edited for brevity and bolded for emphasis):

> ...I can no longer be associated with the OAuth 2.0 standard. I resigned my role as lead author and editor, withdraw my name from the specification, and left the working group. Removing my name from a document I have painstakingly labored over for three years and over two dozen drafts was not easy. Deciding to move on from an effort I have led for over five years was agonizing.
>
> ...At the end, I reached the conclusion that OAuth 2.0 is a bad protocol. WS-* bad. It is bad enough that I no longer want to be associated with it. ...When compared with OAuth 1.0, the 2.0 specification is more complex, less interoperable, less useful, more incomplete, and most importantly, less secure.

To be clear, **OAuth 2.0 at the hand of a developer with deep understanding of web security will likely result is a secure implementation. However, at the hands of most developers – as has been the experience from the past two years – 2.0 is likely to produce insecure implementations.**

Share Improve this answer Follow

edited Jan 26 '18 at 14:16
YakovL
5,094 10 43 65

answered Jun 28 '13 at 11:59
Tony Knibb
468 6 6

7 Note that link-only answers are discouraged, references tend to get stale over time. Please consider adding a stand-alone synopsis here, keeping the link as a reference. – kleopatra Jun 28 '13 at 12:10

6 Security of OAuth 1.0 relies on the assumption that a secret key embedded in a client application can be kept confidential, but the assumption is naive in the case of smartphone applications. In OAuth 2.0, such a naive client application is called *confidential client*. There is no practical difference in security level between OAuth 1.0 clients and OAuth 2.0 confidential clients. "OAuth 2.0 and the Road to Hell" misses this point. – Takahiko Kawasaki Apr 11 '16 at 16:47

24

OAuth 2.0 signatures are not required for the actual API calls once the token has been generated. It has only one security token.

OAuth 1.0 requires client to send two security tokens for each API call, and use both to generate the signature. It requires the protected resources endpoints have access to the client credentials in order to validate the request.

Here describes the difference between OAuth 1.0 and 2.0 and how both work.

Share Improve this answer Follow

answered Jan 9 '12 at 6:02
Fionaa Miller
451 3 4

19

If you need some advanced explanation you need read both specifications :

https://oauth.net/core/1.0a/

https://oauth.net/2/

If you need a clear explanation of flow differences , this could be help you:

OAuth 1.0 Flow

1. Client application registers with provider, such as Twitter.
2. Twitter provides client with a “consumer secret” unique to that application.
3. Client app **signs** all OAuth requests to Twitter **with** its unique **“consumer secret.”**
4. If any of the OAuth request is malformed, missing data, or signed improperly, the request will be rejected.

OAuth 2.0 Flow

1. Client application registers with provider, such as Twitter.
2. Twitter provides client with a “client secret” unique to that application.
3. Client application **includes “client secret”** with **every** request commonly as http header.
4. If any of the OAuth request is malformed, missing data, or contains the wrong secret, the request will be rejected.

Source : https://codiscope.com/oauth-2-0-vs-oauth-1-0/

Share Improve this answer Follow

edited Mar 26 '19 at 14:28

answered Apr 10 '17 at 14:40

JRichardsz
7,019 2 35 57

3 Could you see the **signs** bold text? Maybe functional could have the same concept but, technically speaking use a simple **header** (oauth2) it's very different to **sign** the entire request. **Pay attention and improve your reading comprehension** before mark answers as **not useful** – JRichardsz Oct 22 '19 at 21:15

2 please read your own answer and try to make a sense of it. “Signs all request with secret” and “send secret with all requests”. Nobody in their right mind is going to understand the difference here unless he already used them. I know the difference but the OP doesn’t. This answer will only confuse OP further hence the downvotes. Such vague answers deserve a downvote. Please read other answers here which are far more specific and informative. – saran3h Oct 23 '19 at 4:16

12 **developers** got it. oauth1 & oauth2 have many differences. Previous answers cover them and **As I said**, you can read this oauth.net/core/1.0a or this oauth.net/2 to make your own answer. My goal is show one of the most notorious technical **difference** when a **developer** need to develop a rest client. – JRichardsz Oct 23 '19 at 14:15

OAuth 2.0 promises to simplify things in following ways:

7

1. SSL is required for all the communications required to generate the token. This is a huge decrease in complexity because those complex signatures are no longer required.
2. Signatures are not required for the actual API calls once the token has been generated -- SSL is also strongly recommended here.
3. Once the token was generated, OAuth 1.0 required that the client send two security tokens on every API call, and use both to generate the signature. OAuth 2.0 has only one security token, and no signature is required.
4. It is clearly specified which parts of the protocol are implemented by the "resource owner," which is the actual server that implements the API, and which parts may be implemented by a separate "authorization server." That will make it easier for products like Apigee to offer OAuth 2.0 support to existing APIs.

Source:http://blog.apigee.com/detail/oauth_differences

Share Improve this answer Follow

edited Jan 28 '14 at 18:08

answered Mar 14 '13 at 5:14

Abhijit Gaikwad
2,849 24 37

2

From a security point of view, I'd go for OAuth 1. See OAuth 2.0 and the road to hell

quote from that link: "If you are currently using 1.0 successfully, ignore 2.0. It offers no real value over 1.0 (I'm guessing your client developers have already figured out 1.0 signatures by now).

If you are new to this space, and consider yourself a security expert, use 2.0 after careful examination of its features. If you are not an expert, either use 1.0 or copy the 2.0 implementation of a provider you trust to get it right (Facebook's API documents are a good place to start). 2.0 is better for large scale, but if you are running a major operation, you probably have some security experts on site to figure it all out for you."

Share Improve this answer Follow

edited Apr 18 '16 at 7:44

answered Nov 28 '14 at 12:52

harmv
1,570 18 17