AuthN is "Who are you?"

AuthZ is "What can you do?"

OAuth is for AuthZ only by design.

OAuth is for AuthZ without sharing AuthN.

OAuth: Better Than Passwords

- Never had to share our password
- Revoke the access at any time
- Third-party website can only do what we granted
- We're training users to do silly things
- Never had to share our password

OAuth Vocabulary

- The exchange is called a **flow** or **grant type**
- The permissions are called **scopes**
- Access is granted via **tokens**

OAuth solves a major problem.

We can grant access without giving up control.

Some Challenges in OAuth

- OAuth is a framework—it doesn't solve every problem
- Specific solutions come through extensions
- Extensions vary from vendor to vendor
- Lots of things are open to interpretation

Why Not SAML?

- Defined in 2002–03
- SAML was designed primarily for SSO (AuthN)
- XML-based, so heavy payloads
- Not designed for delegated access
- Defined before mobile apps, and early in API history

So, how do we AuthN+AuthZ for APIs?

Approaches to API AuthN+AuthZ

- API keys
  - Simple, common, easy to implement and use
  - Hard to manage and rotate
- Account Identifier+Secret
  - Very similar to API keys approach
  - False sense of security

Approaches to API AuthN+AuthZ

- OAuth
  - Easy to delegate selective access
  - Easy to revoke access
  - Common, well-supported in most tech stacks
  - Not easy to implement

OAuth is for AuthZ.

OAuth doesn't specify payloads.

Though we often use JSON Web Tokens (JWTs).

OAuth doesn't specify AuthN.

That's up to the Identity Provider.

OAuth doesn't specify Identity.

We know someone AuthN+AuthZ, but not who.

OpenID Connect (OIDC)

- A simple identity extension on top of OAuth 2.0
- Defines specific fields (content and structure) for sharing profile information, such as address, phone number, email, and other fields
- This is done through an ID token
- Not part of the previous OpenID standard

OIDC is not for AuthZ.

But it builds on OAuth, which is for AuthZ.

OAuth 2.0	OpenID Connect
• Designed for AuthZ	• Designed for identity
• Extensible	• Structure+extension
• User or devices/services	• Only for users

OIDC is just a special case of OAuth.

RFC 6749: OAuth Core

- /authorize

  The endpoint which the end user (resource owner) interacts with to grant permission to the resource for the application

  Could return an authorization code or an access token

- /token

  The endpoint which the application uses to trade an authorization code or refresh token for an access token

Everything else is an extension.

Good: Because there are some gaps to cover
Bad: They're all optional

RFC 7009: Token Revocation

- /revoke

  The endpoint which applications use to deactivate (invalidate) a token

  Valid for access or refresh tokens

RFC 7662: Token Introspection

- /introspect

  The endpoint which applications use to learn more about a token

  Whether it is active or not (not revoked, within expiration)

  (Optional) Additional information such as expiration time, scopes included, the issued to client_id, etc.

RFC 7591: Dynamic Client Registration

- /register

  The endpoint which applications use to create new OAuth clients (client_id and client_secret) for provisioning new applications or users

  Introduces the concept of "metadata discovery documents"

OpenID Connect Core

- /userinfo

  The endpoint that applications use to retrieve profile information about the authenticated user

  This returns a spec-defined set of fields, depending on the permissions (scope) requested.

Working Draft: Server Discovery

- /.well-known/openid-configuration

  The endpoint which applications use to retrieve the configuration information for the OIDC server

  This returns a spec-defined set of fields.

There are lots more RFCs.

We'll cover some more.

RFC 6749: OAuth Core

- Access Token

  The token given to the application to access the protected resource on the user or application's behalf

  No formal requirements to format, contents, etc.

- Refresh Token

  The token given to the application to request a new access token on its expiration

I don't want to log in repeatedly…Can we have long lived access tokens? Days? Months? Years?

Short-Lived Access Tokens

Force an application to return to the server and exchange the Refresh Token, checking permissions

OpenID Connect Core

- ID Token

  It is specifically designed for user profile information.

  The specification details the main structures, how to name them, and when to use them.

They have fundamentally different purposes.

Token Management

RFC 6749: Access Tokens

"Access tokens are credentials used to access protected resources. An access token is a string representing an authorization issued to the client. The string is usually opaque to the client."

-Internet Engineering Task Force

Tokens are strings and might be opaque.

RFC 6749: Access Tokens

"Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server."

-Internet Engineering Task Force

Tokens represent scopes
and expiration/duration aspects.

RFC 6749: Access Tokens

"The access token provides an abstraction layer, replacing different authorization constructs (e.g., username and password) with a single token understood by the resource server."

-Internet Engineering Task Force

Token is not a username and password.

RFC 6749: Access Tokens

"Access tokens can have different formats, structures, and methods of utilization (e.g., cryptographic properties) based on the resource server security requirements."

-Internet Engineering Task Force

What We Actually Know

- Access Tokens might include information, or reference it
- If an Access Token has data, it can be any data
- The data can be in any format
- But it will never be the username and password
- But it might be opaque anyway
- Not very helpful…

RFC 7519 JSON Web Token (JWT)

Finally, something that can help!

RFC 7519 Defines

- "iss" – the issuer of the token, an entity we trust
- "sub" – the subject (user) of the token
- "aud" – the audience or intended recipient of the token
- "exp" – the expiration time of the token
- Additional fields that aren't relevant right now…

RFC 7009: Token Revocation

- /revoke

  The endpoint which applications use to deactivate (invalidate) a token

  Valid for access or refresh tokens

  This revocation is not broadcast to applications or APIs.

We need to be careful revoking a token.

Using /revoke is not foolproof in all scenarios with all applications.

Scopes and Claims

Scope

Also known as: "What am I allowed to do?"

A group of permissions that a user can grant to an application to take action on their behalf

In OAuth, they're completely undefined.

Except when you…

Let's come back to that.

Claim

A key/value pair within the token that gives the client application information.

That's it! There's nothing else to them!

In OAuth, they're completely undefined.

RFC 7519 Defines

- "iss" – the issuer of the token, an entity we trust
- "sub"– the subject (user) of the token
- "aud" – the audience or intended recipient of the token
- "exp" – the expiration time of the token

Getting some more structure would be great!

Efforts to Standardize Scopes and Claims

- Healthcare

  Heart Working Group – http://openid.net/wg/heart

  FHIR – http://www.hl7.org/implement/standards/fhir

- Finance

  Financial API – http://openid.net/wg/fapi

OIDC Scopes

- OpenId (required)
- Profile
- Email
- Address
- Phone

OIDC Claims

- (Varies depending on the scopes)
- Sub
- Name (many pieces)
- Address (street address+pieces)
- Email
- And many others…

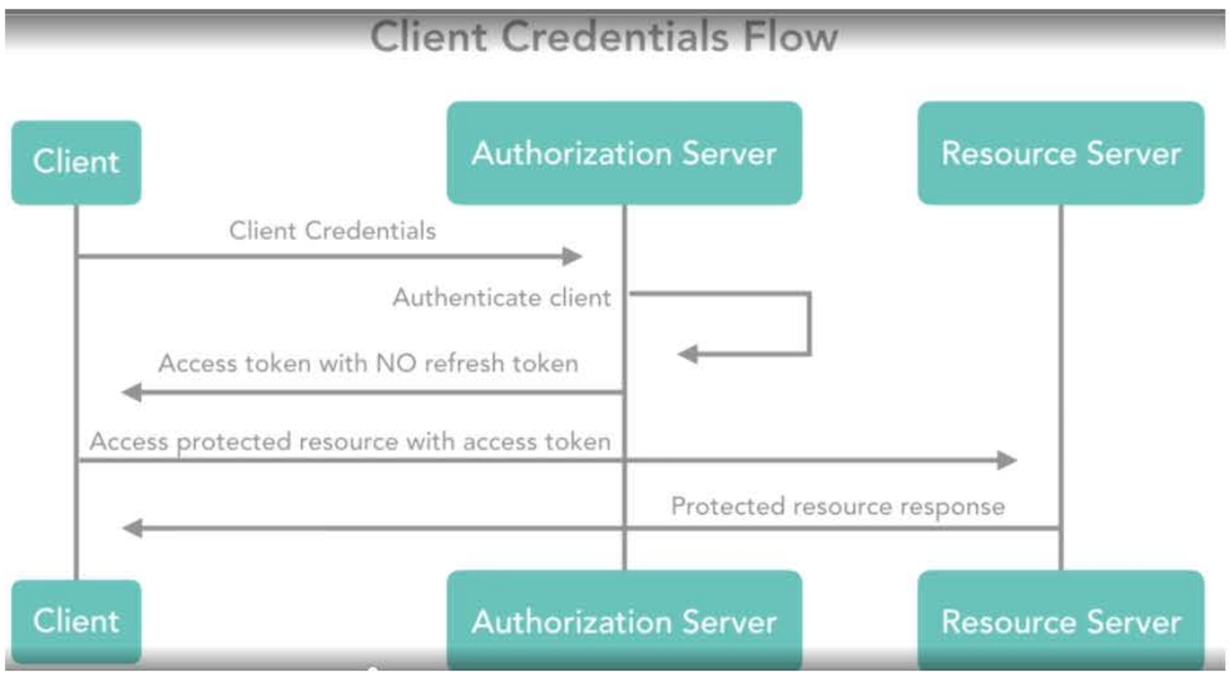
"Log in with…" uses OpenID Connect.

OpenID Connect is a huge interoperability win for OAuth.

Client Credential Grant Type

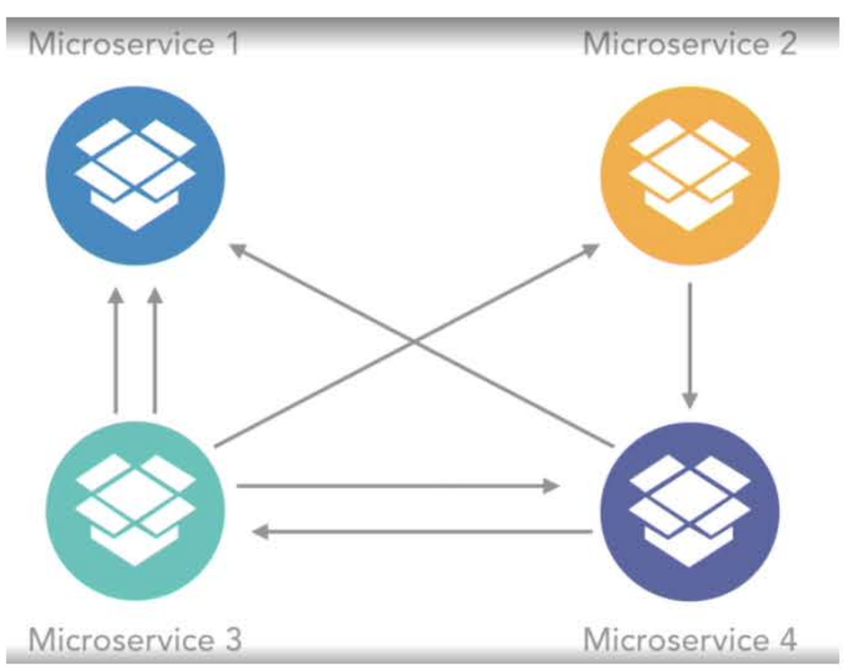
Designed specifically for server-to-server interactions, similar to "service accounts" from Active Directory and other backend services

No user involved (therefore not compatible with OIDC)



How the Client Credentials Flow Works

- I use task tracking system Trello
- I upload a file to a Trello task
- Trello uploads the file to Amazon S3
- I can download or view the file as I need



Why Client Credentials Flow?

- All the benefits of OAuth
- Easy onboarding

  I didn't need an S3 account to use Trello.

- Pluggable interfaces

  They can switch backend services as needed without my knowledge or involvement.

Important

The Client Credentials Grant Type is only for "private" clients where the interaction is handled by backed code.

Client Credential Flow Security Considerations

The Good News

Having no users means they can't compromise it!

Security Considerations

- Private clients only

  Secrets must be in backend code.

  Not appropriate for mobile apps or single page apps.

- Must use secure communications
- We don't have a user for logging or audit purposes

Security Consideration

- Validating our Access Tokens

  We could use the /introspect endpoint (if supported).

  We could use the token validation rules (RFC 7519).

  Either way, we'll likely cache the results—be careful.

Let's talk mobile.

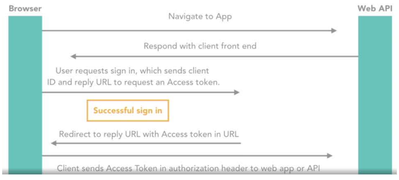
Implicit or Hybrid Grant Type

Designed specifically for untrusted or "public" clients, where a malicious user could get access to the source code

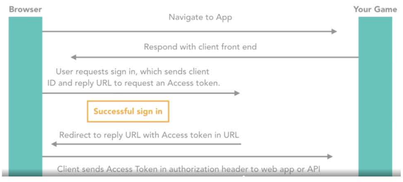
In other words, we can't count on it to keep our client_secret safe.

This is only for users who explicitly grant authorization via an authentication and user-consent flow.

Authorize Endpoint



Facebook Authorize Endpoint



Implementing It

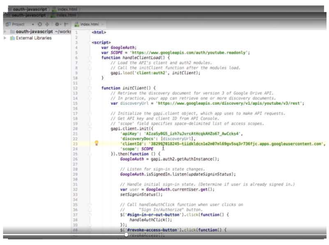
- For Android or iOS/Swift

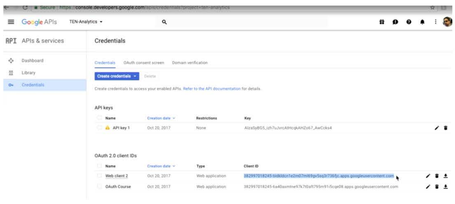
  Use AppAuth, formally backed by OpenID Foundation, etc.

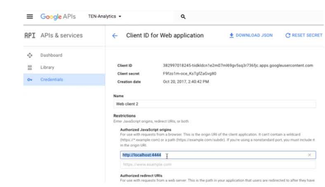
  Respects the OAuth flows, cookie/session storage, etc.

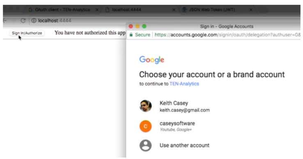
- For single-page apps

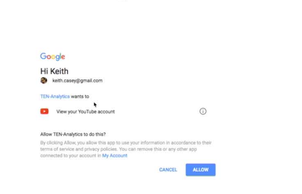
  Use Passport for a modular/pluggable approach to authentication and authorization, regardless of your provider.

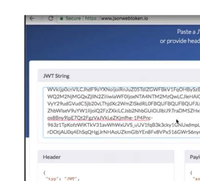


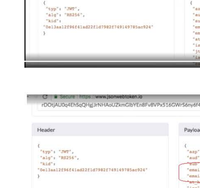












Hybrid or Implicit Flow Security Considerations

AppAuth and Passport

Evaluate these first. Don't create your own.

Security Considerations

- Remember what an Access Token is

  It's permission to act as the user.

  We have to store and use it securely at all times.

- Must use secure communications

Security Considerations

- Test your redirect_uri whitelist

  This is where the server sends the resulting access token.

  Confirm that your server doesn't accept any arbitrary redirect_uri.

  Probably fine with a mainstream server, but check anyway.

- All resource servers must validate the access tokens

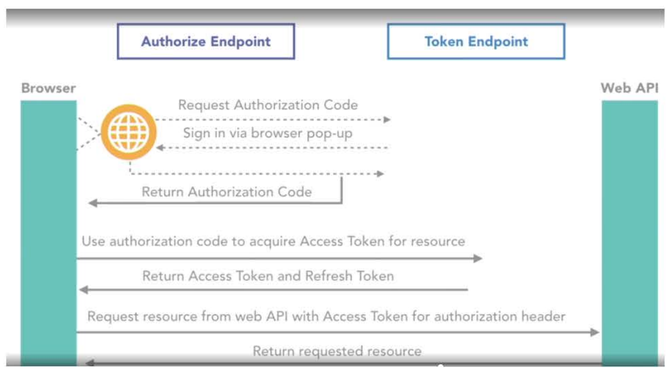
  You don't have to love it, but you have to do it.

Authorization Code for Web Applicatin

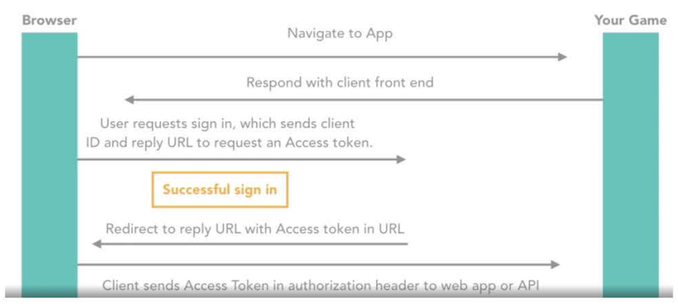
Let's move server side.

The Basics

- It's not simple, but it is reliable
- It's not mobile friendly, but we can trust it
- Works the same in any back-end programing language
- The attack/threat vectors are small and short-lived
- Large and growing ecosystem including libraries, Postman, etc.

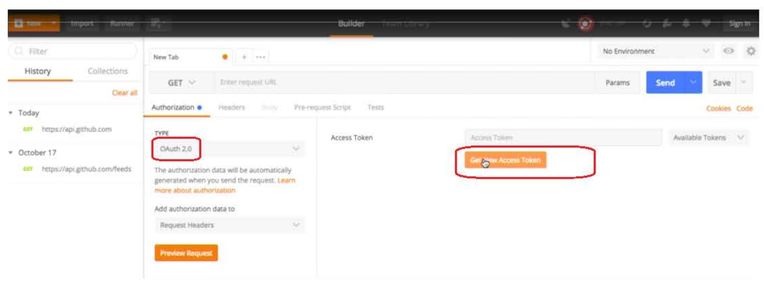


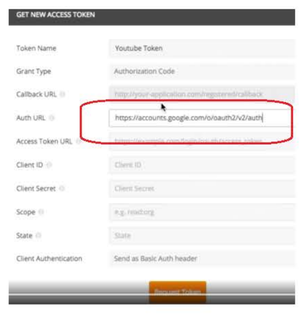
Facebook Authorize Endpoint

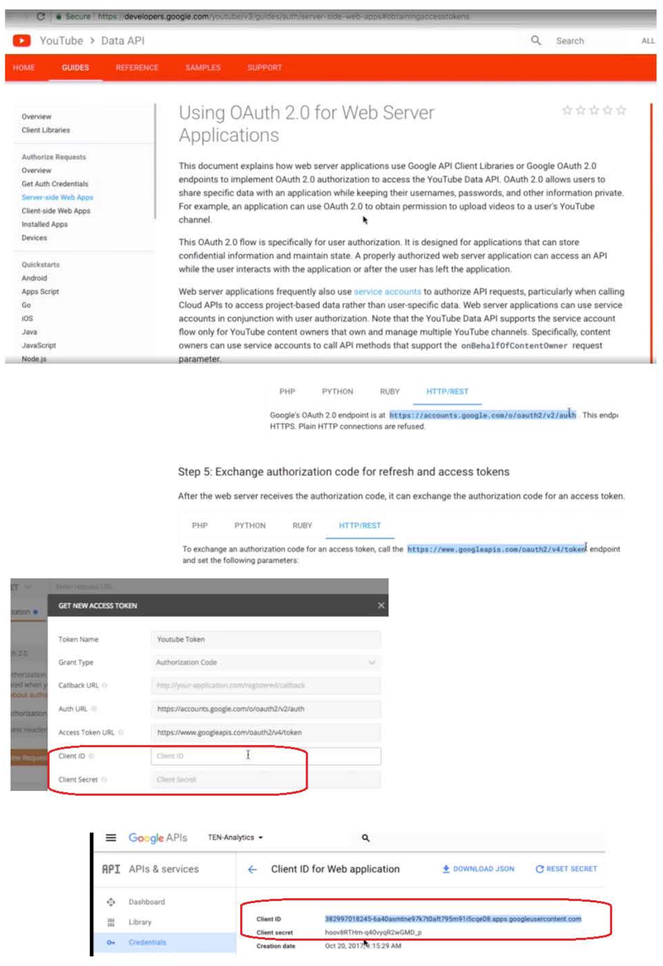


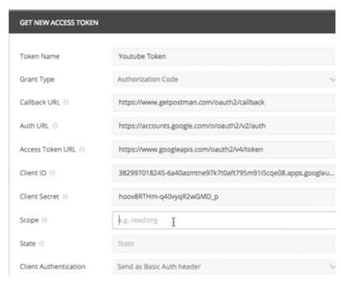
Creating the Authorization Code URL is complex.
Use a well-established library. Your life will be easier.

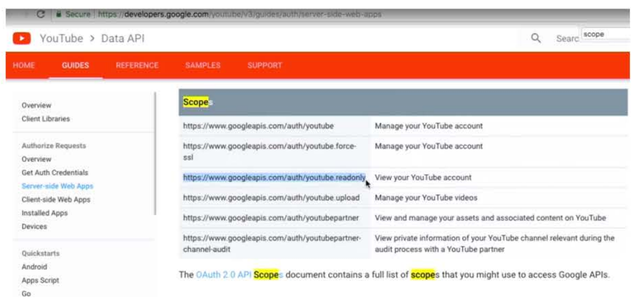
Build an example with JavaScript

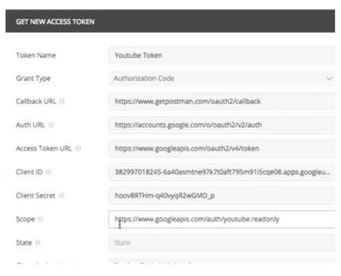


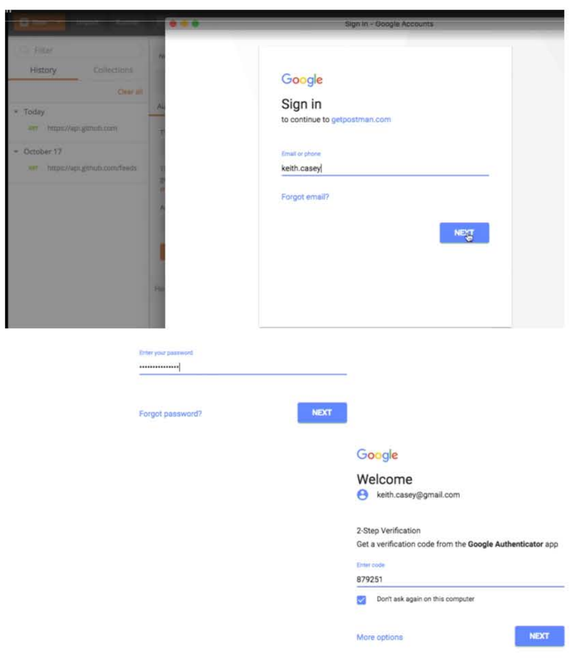


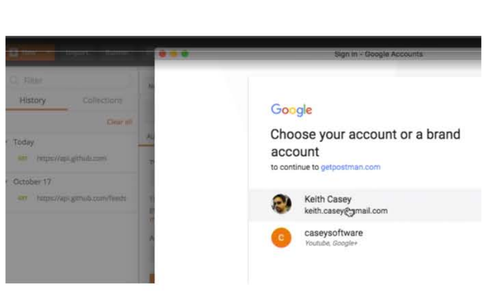


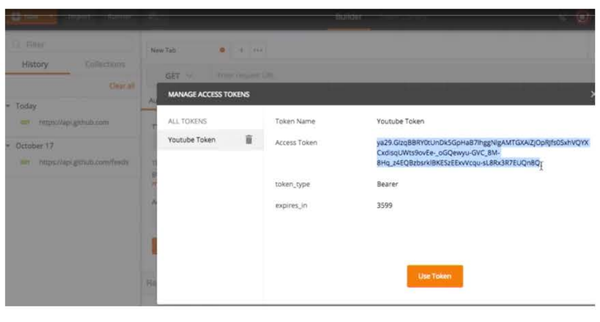


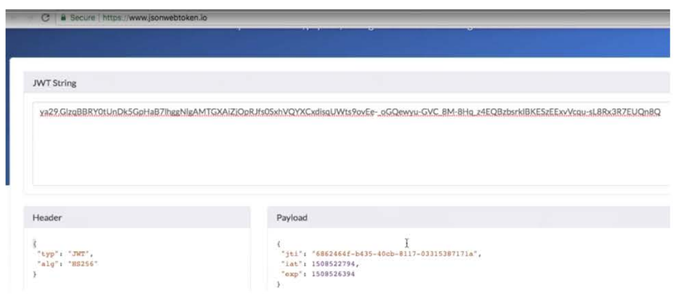












Authorization Code Flow Security Considerations

This is usually the most secure, except for...

Security Considerations

- Protect the auth code
  - It's the one-time code that gives us an access token.
  - (We never see the resulting access token.)
- Be careful of active browser sessions
  - Generally, user-consent screens will mitigate this risk.
- Must use secure communications

Security Considerations

- Test your redirect_uri whitelist
  - This is where the server sends the resulting access token.
  - Confirm that your server doesn't accept any arbitrary redirect_uri.
- All resource servers must validate the access tokens
  - No, don't argue—just do it.

Security Considerations

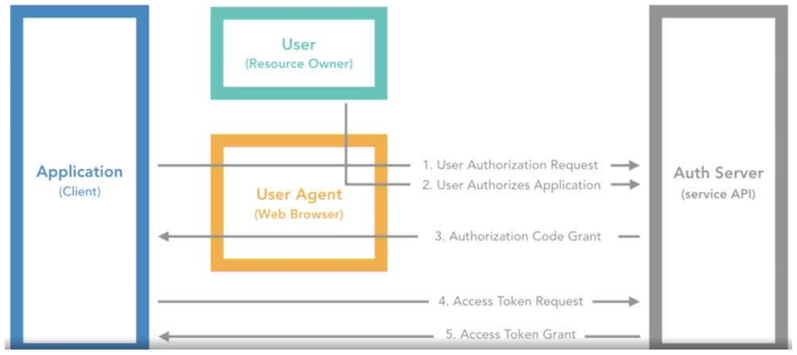
- Impersonation
  - The May 2017 "Google Docs Worm"
  - Disguised itself as Google and asked for permissions via the normal OAuth authorization flow
  - It hit millions of people—no one is sure of the impact or goal.
  - Google responded quickly and shut it down.
  - This was 100% human behavior, not a flaw in OAuth.

What about legacy applications?

Resource Owner Password Flow

This is not the flow you're looking for.

Authorization Code Flow



The Basics

- The user never saw the access token
- The application had the user's credentials
- Do we trust the application and its developers?

This defeats the purpose of OAuth.

There is one valid use case

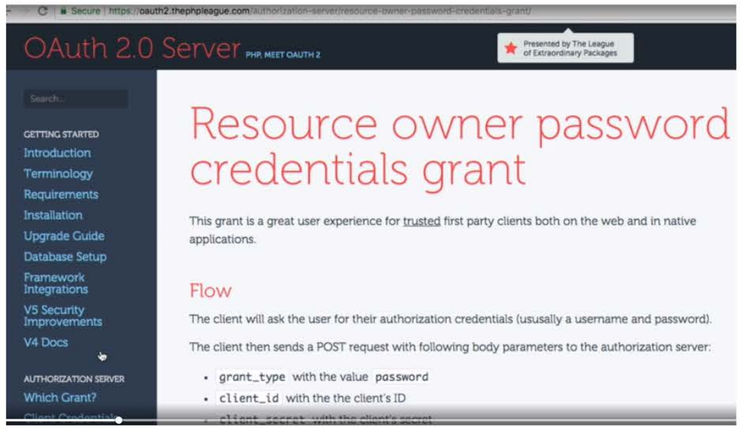
Wrapping legacy systems in an OAuth interface to bring them into the same security practices

This should be a *temporary* state.

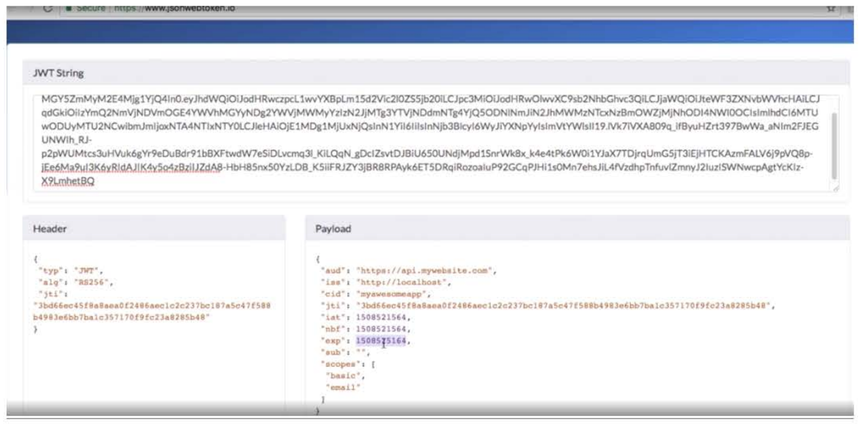
New applications should never use Resource Owner Password Flow.

Time for some code

(that hopefully you never need).




```
curl -X POST "http://localhost:4444/password.php/access_token" -d
curl: option -d: requires parameter
curl: try 'curl --help' or 'curl --manual' for more information
~
curl -X POST "http://localhost:4444/password.php/access_token" -d "grant_type=password&client_id=m
yawesomeapp&client_secret=abc123&username=keith&password=keithisawesome&scope=basic%20email"
{"token_type":"Bearer","expires_in":3600,"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6I
jgxZjdiNjhmYWI3ZGEyMjU0NGFlZGMzNzJkNTM1Y2Q3YjhkNWM1ZTZjMjQ1N2FjNGUzNDE5MTRkODliYjE5OTkyZmRiNTViNWU5N2Z
mYTY0In0.eyJhdWQiOiJodHRwczpcL1wvYXBpLm15d2Vic2l0ZS5jb20iLCJpc3MiOiJodHRwOlwvXC9sb2NhbGhvc3QiLCJjaWQiO
iJteWF3ZXNvbWVhcHAiLCJqdGkiOiI4MWY3YjY4ZmFiN2RhMjI1NDRhZWRjMzcyZDUzNWNkN2I4ZDVjNWU2YzI0NTdhYzRlMzQxOTE
0ZDg5YmIxOTk5MmZkYjU1YjVlOTdmZmE2NCIsImlhdCI6MTUwODUyMTU1NCwibmJmIjoxNTA4NTIxNTU0LCJleHAiOjE1MDg1MjUxN
TQsInN1YiI6IiIsInNjb3BlcyI6WyJiYXNpYyIsImVtYWlsIl19.L86ussEfD5KavX5Esbk4lJecXnQTb9QpXk5GBfRLi50kAnnCAm
dCSYiVGpC6b-gbAWNUwt8DXENJSj-PRzJvtSr7PBU9j8PwJ3hJpkBSyxk1xjUBL3agGkMHC5NAFwNQolv51UiwoBA9kIWGtWNnvTw0
3yB4q5D88ztLURkuNiTi_yg7m8hmbkgC6-pYLJSpWGfEP9oJOAuLFM9AfVLY9-TaNItLvRaoE54JuNYB1u7NjruX8MsAOCdMQ7Vs7v
3L7AyUVNJyRoVtfb6araaD9sGwaRmUTYd3EaJ9guYh80h73MSOlLW7n1XP32SrhOQPHGMGHjUSjotF5xdHFBwWIA"}~
curl -X POST "http://localhost:4444/password.php/access_token" -d "grant_type=password&client_id=m
yawesomeapp&client_secret=abc123&username=keith&password=keithisawesome&scope=basic%20email" | ~/jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1028  100   906  100   122   4620    622 --:--:-- --:--:-- --:--:--  4646
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjNiZDY2ZWM0NWY4YThhZWEwZjI0ODZhZWMxYzJ
jMjM3YmMxODdhNWM0N2Y1ODhiNDk4M2U2YmI3YmExYzM1NzE3MGY5ZmMyM2E4Mjg1YjQ4In0.eyJhdWQiOiJodHRwczpcL1wvYXBpL
m15d2Vic2l0ZS5jb20iLCJpc3MiOiJodHRwOlwvXC9sb2NhbGhvc3QiLCJjaWQiOiJteWF3ZXNvbWVhcHAiLCJqdGkiOiIzYmQ2NmV
jNDVmOGE4YWVhMGYyNDg2YWVjMWMyYzIzN2JjMTg3YTVjNDdmNTg4YjQ5ODNlNmJiN2JhMWMzNTcxNzBmOWZjMjNhODI4NWI0OCIsI
mlhdCI6MTUwODUyMTU2NCwibmJmIjoxNTA4NTIxNTY0LCJleHAiOjE1MDg1MjUxNjQsInN1YiI6IiIsInNjb3BlcyI6WyJiYXNpYyI
sImVtYWlsIl19.lVk7iVXA809q_ifByuHZrt397BwWa_aNIm2FJEGUNWIh_RJ-p2pWUMtcs3uHVuk6gYr9eDuBdr91bBXFtwdW7e5i
DLvcmq3l_KiLQqN_gDcIZsvtDJBiU650UNdjMpd1SnrWk8x_k4e4tPk6W0i1YJaX7TDjrqUmG5jT3iEjHTCKAzmFALV6j9pVQ8p-jE
e6Ma9uI3K6yRIdAJIK4y5o4zBziIJZdA8-HbH85nx50YzLDB_K5iiFRJZY3jBR8RPAyk6ET5DRqiRozoaIuP92GCqPJHi1s0Mn7ehs
JiL4fVzdhpTnfuvlZmnyJ2IuzISWNwcpAgtYcKIz-X9LmhetBQ"
}
```



Resource Owner Password Flow Security Considerations

Security Considerations

- Protect the client_secret (backend apps only)
- Must use secure communications like TLS
- Protect the access token
- All resource servers must validate the access tokens

Security Considerations

- The actual risk is much bigger

  Your user is giving their credentials to the application.

  Do we want to train users to put their credentials anywhere?

  Do we trust the application and all the developers?

- Scopes don't protect us here
- Revoking access doesn't work either

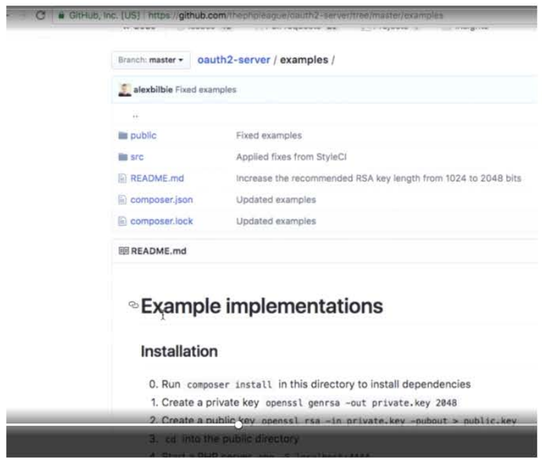
There is one valid use case.

Wrapping legacy systems in an OAuth interface to bring them into the same security practices

!

If you can, avoid the Resource Owner Password Flow.

Configuring an OAuth server in PHP

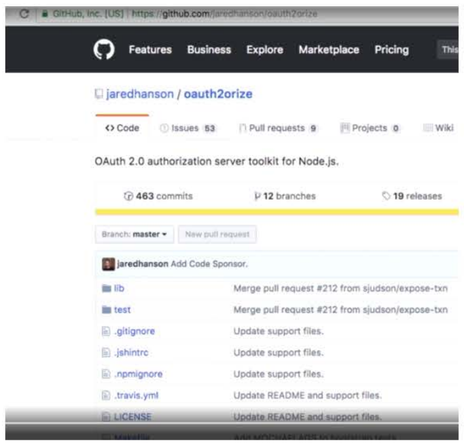


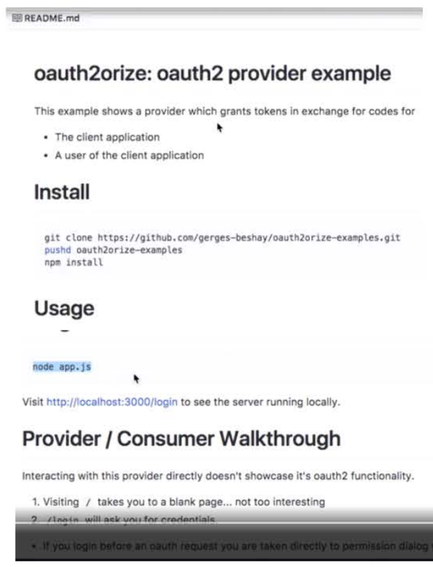
Web Security OAuth and OpenID Connect\Ch07\07_01\README.md

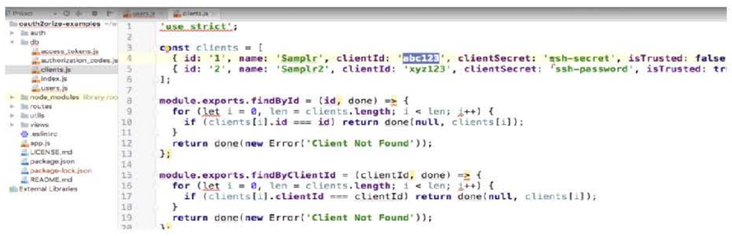
Heads Up

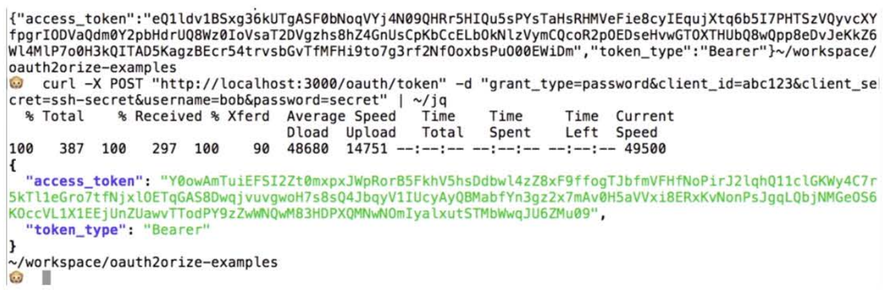
- The project may not support the extensions you need
- It's not always fun to support your projects
- If you have the time, skills, and abilities, it can work
- Make sure your project supports your requirements now and into the future

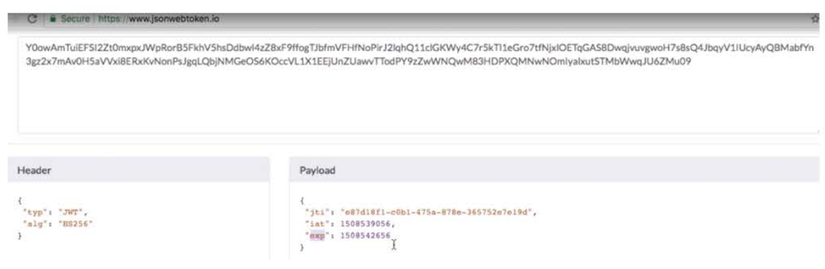
Configuring an OAuth server in Node











OAuth 2.0 as a service using Okta