SECURING MICROSERVICES WITH OAUTH 2 UND OPENID CONNECT

OWASP Chapter Munich 30.4.2019

Slides: https://andifalk.github.io/owasp-chapter-munich-04-2019
Demos: https://github.com/andifalk/owasp-chapter-munich-04-2019

ANDREAS FALK

Novatec Consulting GmbH

andreas.falk@novatec-gmbh.de / @andifalk (Twitter)

https://www.novatec-gmbh.de

NOVATEC

AGENDA

Intro to OAuth 2.0 & OpenID Connect 1.0

4th OAuth Security Workshop 2019

OAuth 2 & OIDC with Spring Security (Live Demo)

Discussion

OAUTH 2.0

101

RFC 6749: The OAuth 2.0 Authorization Framework

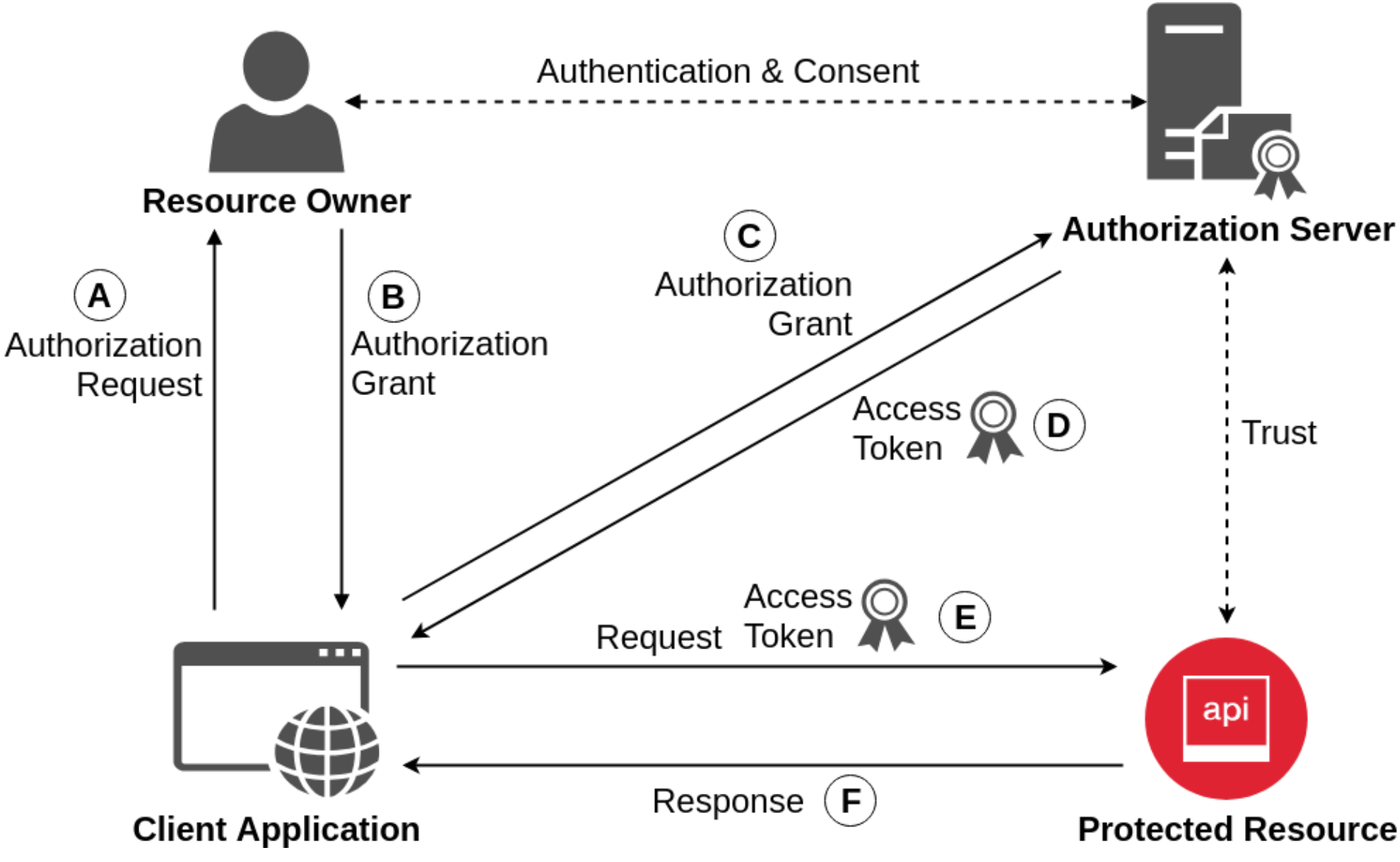
RFC 6750: OAuth 2.0 Bearer Token Usage

RFC 6819: OAuth 2.0 Threat Model and Security Considerations

WHAT IS OAUTH 2.0?

OAuth 2.0 is an authorization delegation framework

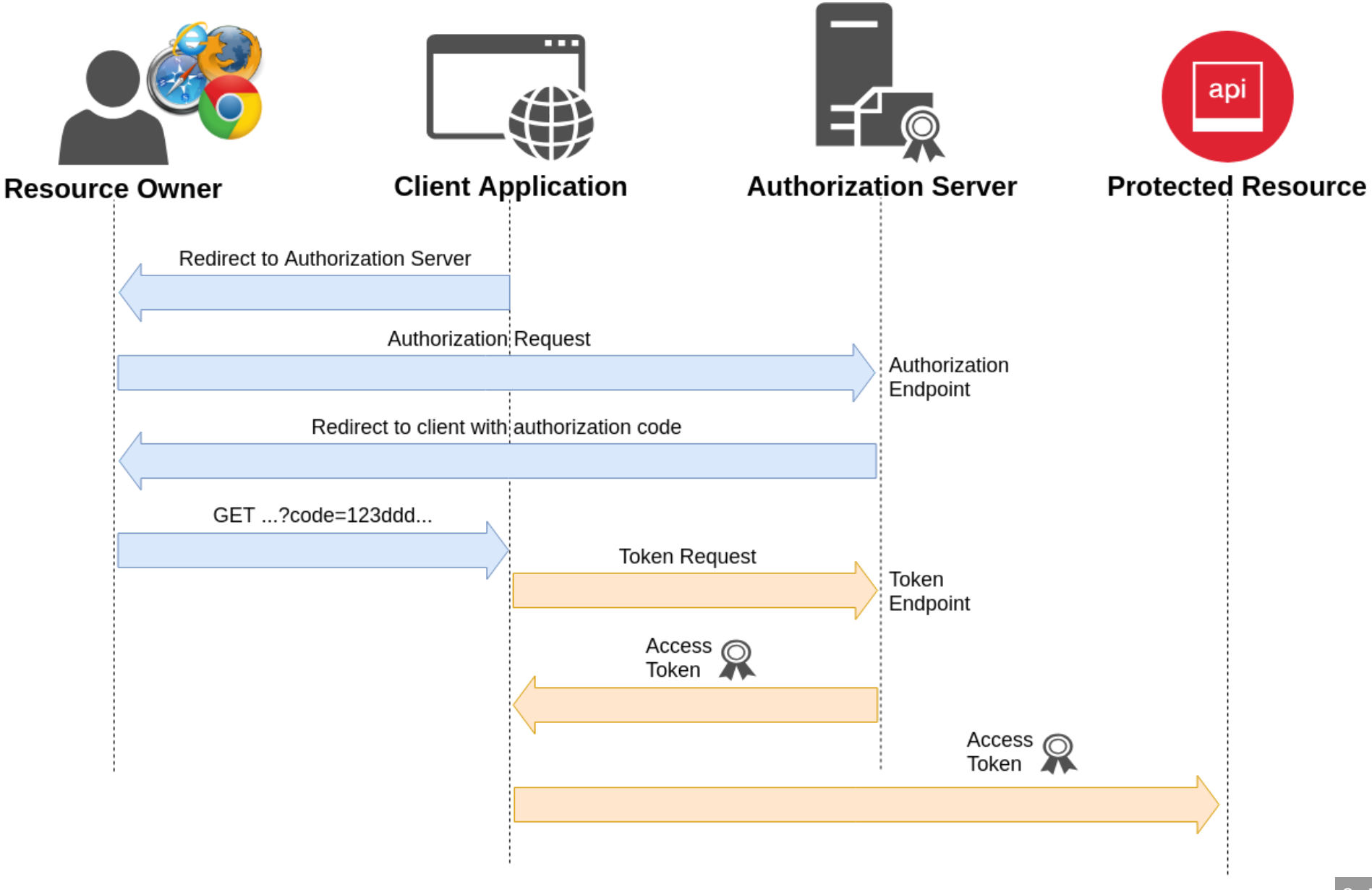
OAUTH 2.0 MODEL



OAUTH 2.0 GRANT FLOWS

Client Type	Flow	Refresh Tokens
Confidential	Authorization Code	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	Implicit	--
Trusted	RO Password Creds	X
No Resource Owner	Client Credentials	--

AUTHORIZATION CODE GRANT FLOW



AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

?code=ab23bhW56Xb

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

?code=ab23bhW56Xb

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

?code=ab23bhW56Xb

&state=xyz

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BASIC AUTH)

Client-Id=123, Client-Secret=456, Base64(123:456)="MTIzOjQ1Ng=="

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

Authorization: Basic MTIzOjQ1Ng==

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

TOKEN REQUEST (BODY)

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

&redirect_uri=https://client.abc.com/callback

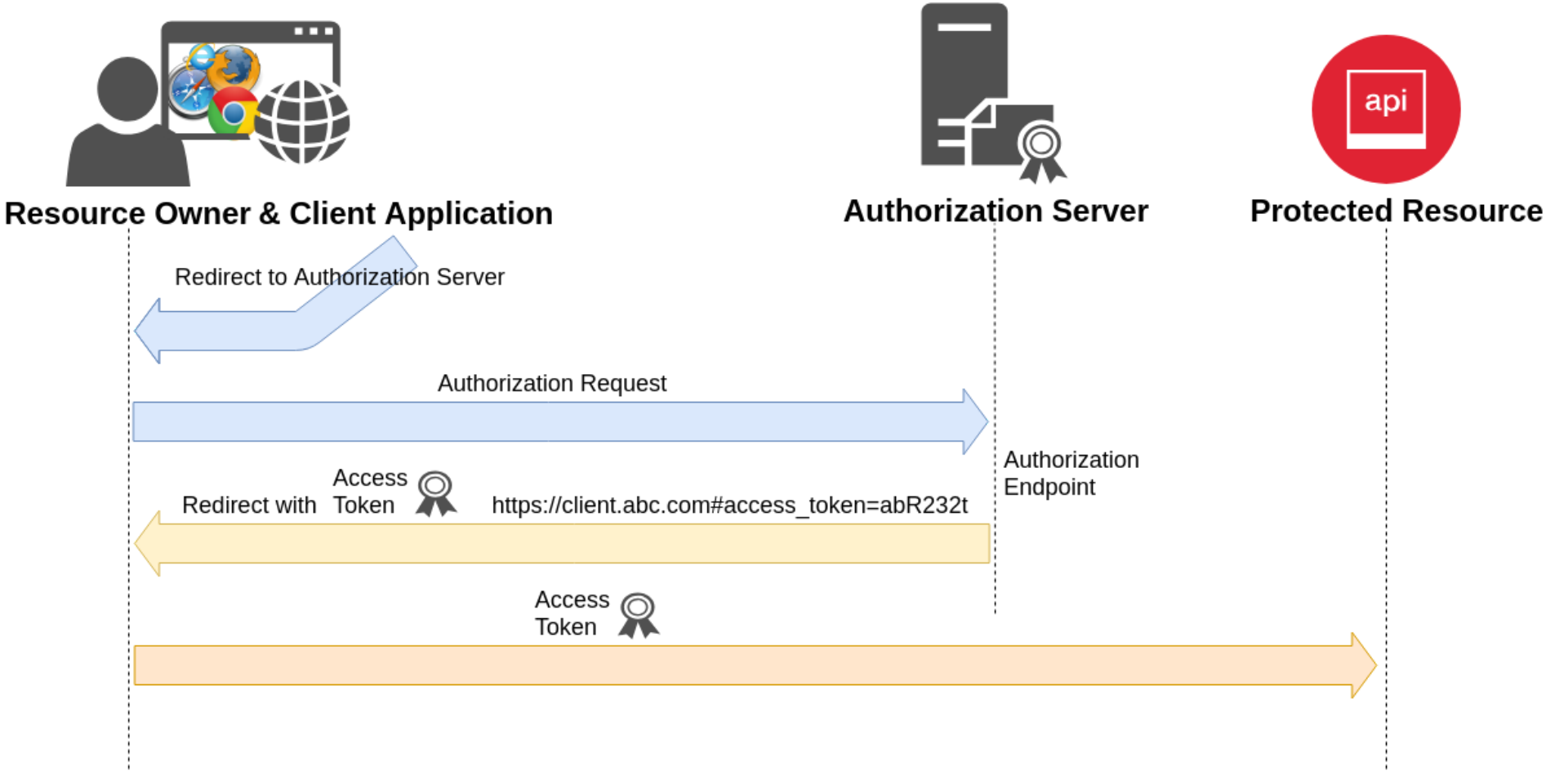
&client_id=123&client_secret=456

TOKEN RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

IMPLICIT GRANT FLOW



AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=token

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

AUTHORIZATION RESPONSE

HTTP/1.1 302 Found

Location: https://client.abc.com/callback

#access_token=2YotnFZFEjr1zCsicMWpAA

&token_type=bearer

&expires_in=3600

&scope=api.read api.write

&state=xyz

FURTHER OAUTH 2.0 STANDARDS

RFC 7636: Proof Key for Code Exchange (“Pixy”)

RFC 7662: Token Introspection

RFC 7009: Token Revocation

OPENID CONNECT 1.0

(OIDC)

101

OpenID Connect Core 1.0
OpenID Connect Dynamic Client Registration 1.0
OpenID Connect Discovery 1.0

OPENID CONNECT 1.0 IS FOR AUTHENTICATION

Jim Manico
@manicode

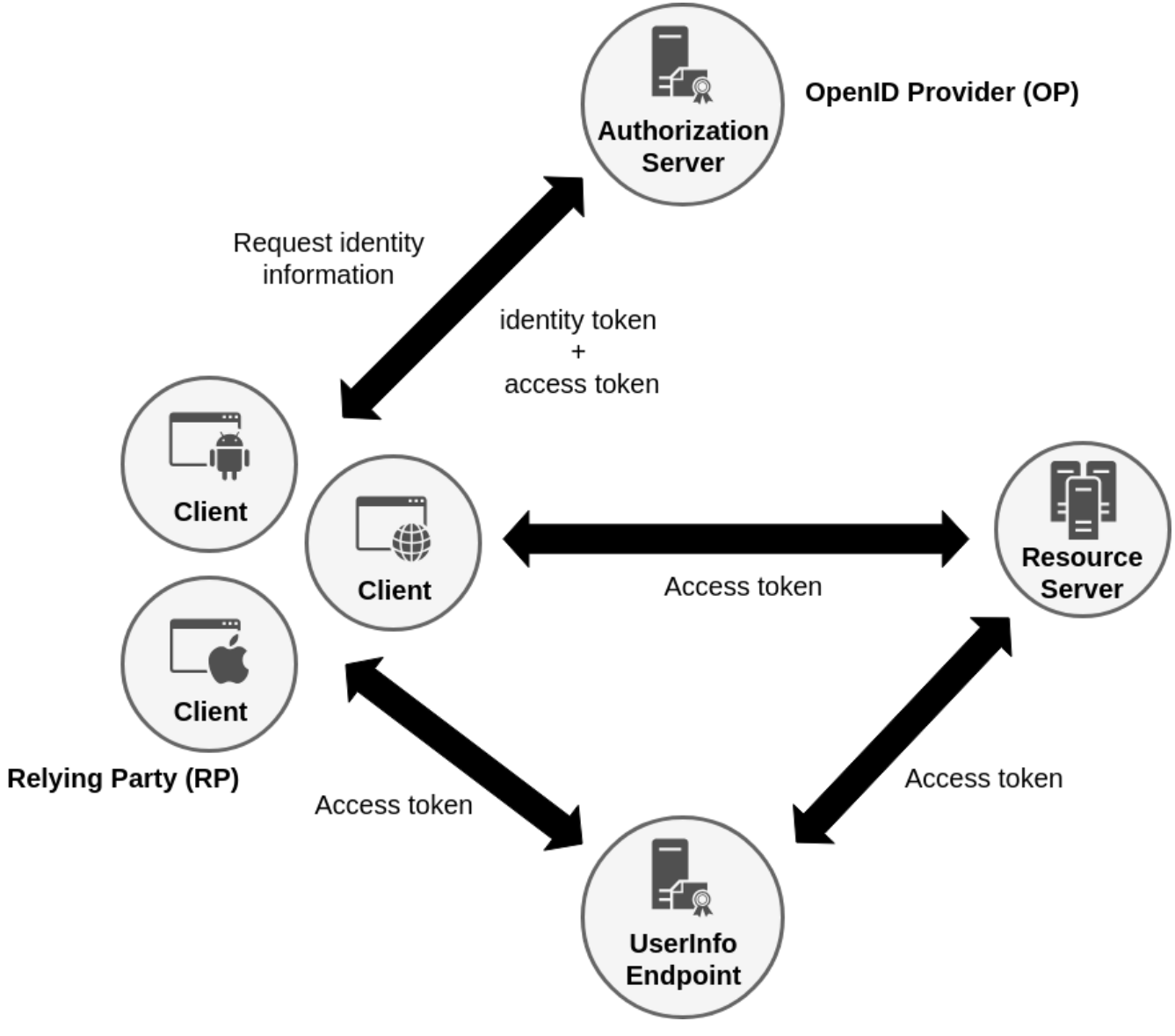
Follow

Repeat after me: OAuth 2.0 is NOT AN AUTHENTICATION PROTOCOL.
oauth.net/articles/authe …

12:22 PM - 2 Feb 2017 from Kapaa, HI

OAuth 2.0 is not an authentication protocol

OIDC MODEL


Authorization Server
OpenID Provider (OP)
Request identity information
identity token + access token
Client
Client
Client
Relying Party (RP)
Resource Server
Access token
Access token
Access token
UserInfo Endpoint

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ADDITIONS TO OAUTH 2.0

Id Token (JWT format)

User Info Endpoint

Standard Scopes

Hybrid Grant Flow

OpenID Provider Configuration Information

ID TOKEN

JSON WEB TOKEN (JWT)

Base 64 Encoded JSON Formatted Value of...

```
GET / HTTP/1.1
Host: localhost:8080
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...
```

- RFC 7519: JSON Web Token (JWT)
- JSON Web Token Best Current Practices
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

ID TOKEN

JSON WEB TOKEN (JWT)

Base 64 Encoded JSON Formatted Value of...

...Header

```
GET / HTTP/1.1
Host: localhost:8080
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...
```

- RFC 7519: JSON Web Token (JWT)
- JSON Web Token Best Current Practices
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

ID TOKEN

JSON WEB TOKEN (JWT)

Base 64 Encoded JSON Formatted Value of...

...Header

...Payload

```
GET / HTTP/1.1
Host: localhost:8080
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...
```

- RFC 7519: JSON Web Token (JWT)
- JSON Web Token Best Current Practices
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

ID TOKEN

JSON WEB TOKEN (JWT)

Base 64 Encoded JSON Formatted Value of...

...Header

...Payload

...Signature

```
GET / HTTP/1.1
Host: localhost:8080
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...
```

- RFC 7519: JSON Web Token (JWT)
- JSON Web Token Best Current Practices
- Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)

JSON WEB TOKEN (JWT)

Header

```
{
  typ: "JWT",
  alg: "RS256"
}
```

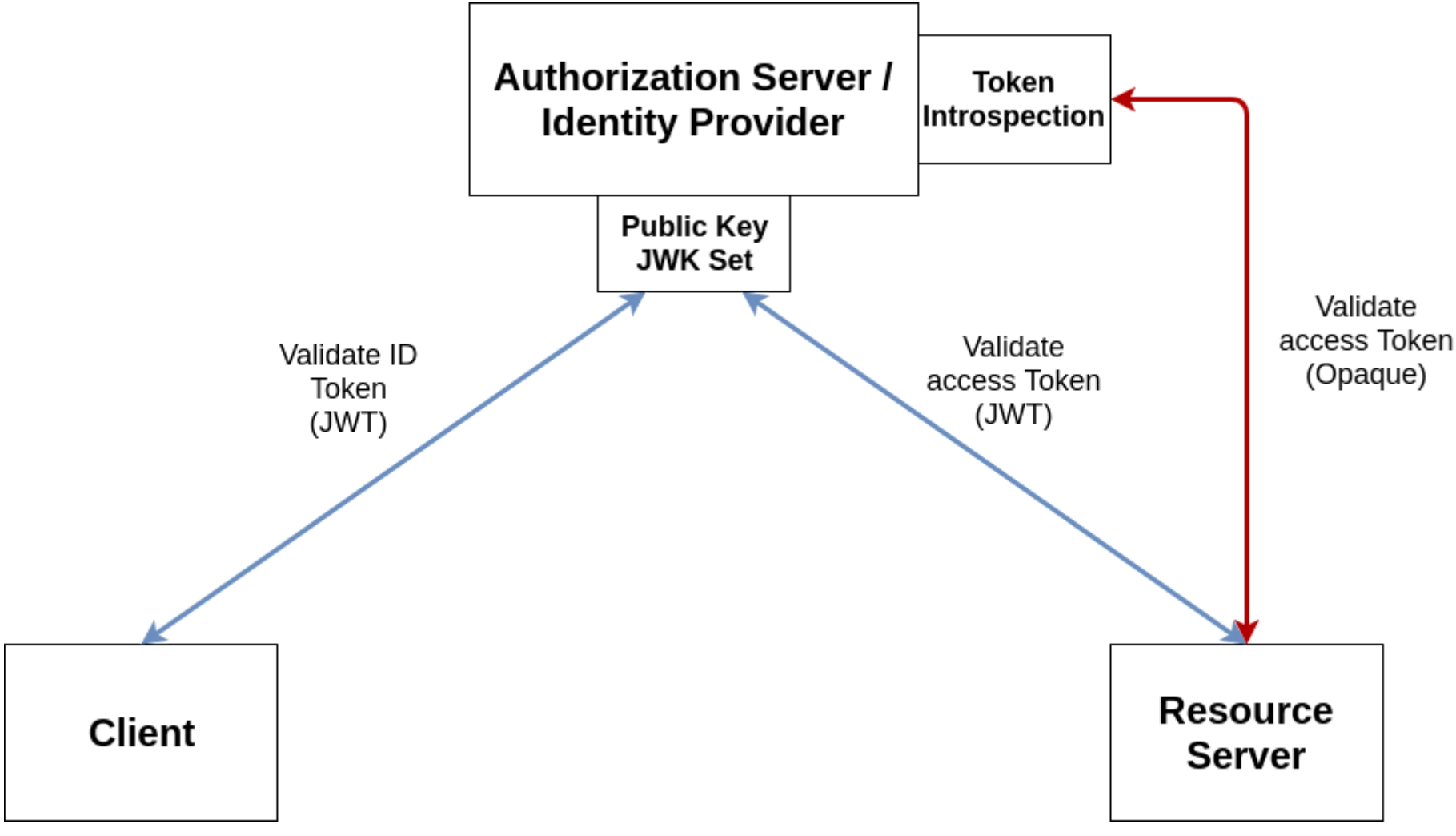
Payload

```
{
  iss: "https://identity.example.com",
  aud: "my-client-id",
  exp: 1495782385,
  nonce: "N0.468248572432335114957391247 49",
  iat: 1495739185,
  at_hash: "hC1NDSB8WZ9SnjXTid175A",
  sub: "mysubject",
  auth_time: 1495739185,
  email: "test@gmail.com"
}
```

ID TOKEN CLAIMS

Scope	Required	Description
iss	X	Issuer Identifier
sub	X	Subject Identifier
aud	X	Audience(s) of this ID Token
exp	X	Expiration time
iat	X	Time at which the JWT was issued
auth_time	(X)	Time of End-User authentication
nonce	--	Associate a client with an ID Token

TOKEN VALIDATION



USER INFO ENDPOINT

```
GET /userinfo HTTP/1.1
Host: identityserver.example.com
Authorization: Bearer SlAV32hkKG
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "sub": "248289761001",
 "name": "Jane Doe",
 "given_name": "Jane",
 "family_name": "Doe",
 "preferred_username": "j.doe",
 "email": "janedoe@example.com",
 "picture": "http://example.com/janedoe/me.jpg"
}
```

OIDC FLOWS

- Authorization Code (w/ or w/o PKCE)
- Implicit
- **Hybrid**

OPENID CONNECT 1.0 CONFIGURATION

https://example.com/.well-known/openid-configuration

```
{
  "authorization_endpoint": "https://idp.example.com/auth",
  "grant_types_supported": [
      "authorization_code",
      "implicit",
      "refresh_token"
  ],
  "issuer": "https://idp.example.com",
  "jwks_uri": "https://idp.example.com/keys",
  "token_endpoint": "https://idp.example.com/token",
  "userinfo_endpoint": "https://idp.example.com/userinfo",
  ...
}
```

OpenID Connect Discovery 1.0

4TH OAUTH SECURITY WORKSHOP 2019

Stuttgart

https://sec.uni-stuttgart.de/events/osw2019

Why you should stop using the OAuth implicit grant!

Torsten Lodderstedt Follow
Nov 9, 2018 · 3 min read

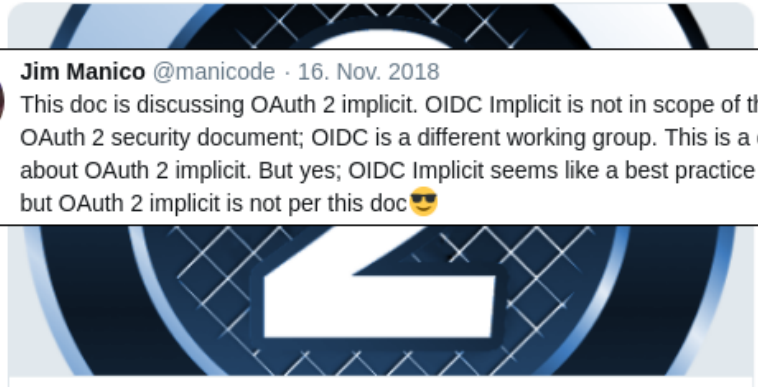
No one should any longer use the implicit grant! That's what IETF's OAuth working group, the authority for official OAuth specifications, recommends in the upcoming OAuth 2.0 Security Best Current Practice RFC. The decision was met during the IETF meeting this week in Bangkok.

https://medium.com/oauth-2/why-you-should-stop-using-the-oauth-implicit-grant-2436ced1c926

Lots of discussions and comments

Jim Manico @manicode · 17. Nov. 2018
To my @SecAppDev friends @YoPeeters and @PhilippeDeRyck please give medium.com/@torsten_lodde... a read. This is where I'm getting my info from. This is •OAuth 2• specific; not OIDC implicit.

Tweet übersetzen



Why you should stop using the OAuth implicit grant!
No one should any longer use the implicit grant! That's what IETF's OAuth working group, the authority for official OAuth specifications...
medium.com

Jim Manico @manicode · 16. Nov. 2018
This doc is discussing OAuth 2 implicit. OIDC Implicit is not in scope of the OAuth 2 security document; OIDC is a different working group. This is a quote about OAuth 2 implicit. But yes; OIDC Implicit seems like a best practice for SPA but OAuth 2 implicit is not per this doc😎

Torsten Lodderstedt @tlodderstedt · 18. Nov. 2018
„token id_token"+exact redirect uri matching help to detect token injection and prevent open redirection, does not prevent access token leakage in general and token replay. That's why the OAuth WG recommends sender constraint tokens. Impossible with tokens issued in frontchannel.

Tweet übersetzen

1

Manfred Steyer @ManfredSteyer · 18. Nov. 2018
You are repeating arguments that have been already disproved in this thread + the wg is currently recommending nothing -- it's just a draft. And no flow prevents from token leakages b/c no save storage in browser. Even the access code can be leaked and used b/c no user secret.

Jim Manico @manicode · 16. Nov. 2018
"Clients SHOULD NOT use the implicit grant and any other response type causing the authorization server to issue an access token in the authorization response" This is the RFC I'm referring to tools.ietf.org/id/draft-ietf-... which was just released November 9th, 2018

Tweet übersetzen

2

Philippe De Ryck @PhilippeDeRyck · 16. Nov. 2018
This is too nuanced for tweets. The doc does not say "do not use access tokens in browser". It says that the PKCE-based flow has better properties than the IG flow, and it is recommended instead. If used in an SPA, the result is still an access token, just not in the redirect ...

damienbod
@damien_bod
Folge ich

Antwort an @manicode @PhilippeDeRyck @dfett42

The OIDC Implicit Flow with a good CSP solves almost everything mentioned in the referenced draft docs...

The main problem in SPAs of storing tokens and refreshing the session is still unsolved in the code flow with PCKE, so there is no real improvement =>

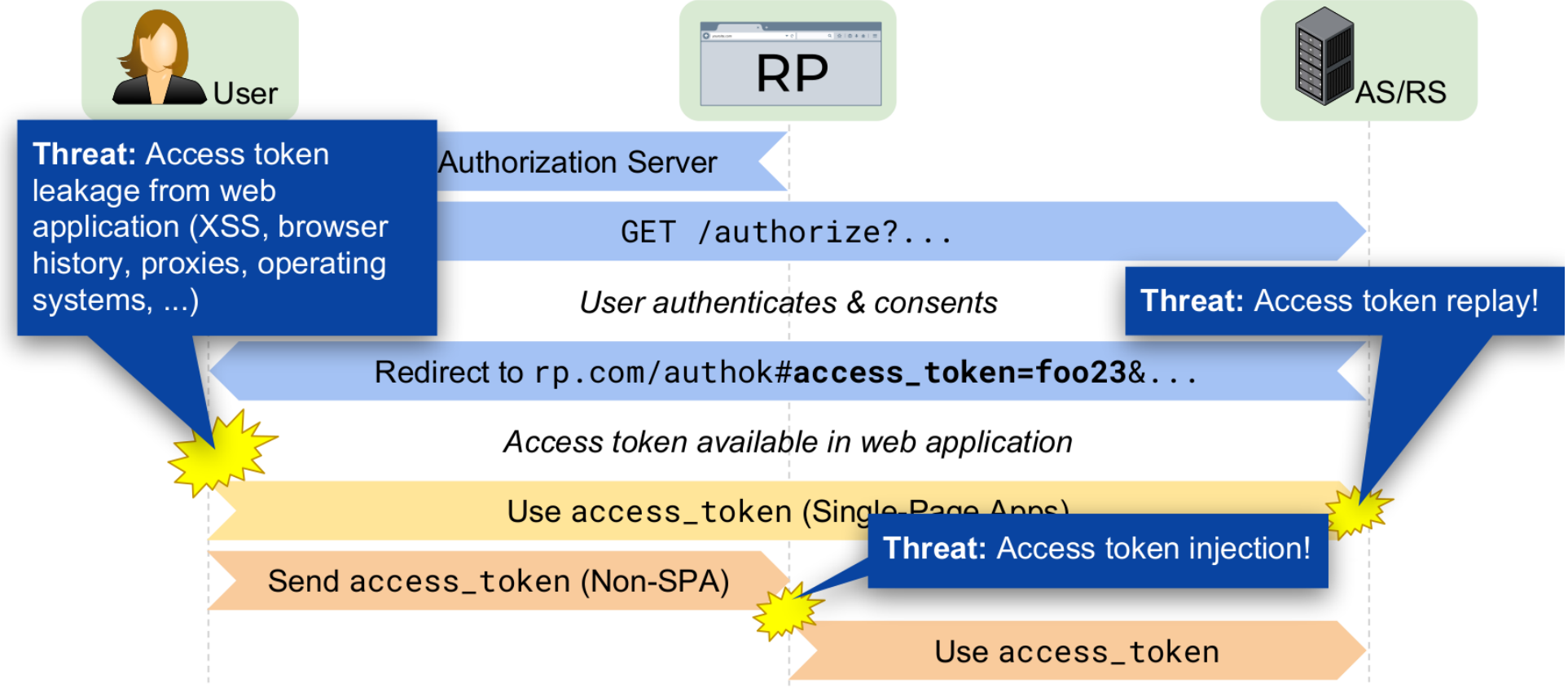
OAUTH 2.0 SECURITY BEST CURRENT PRACTICE

Torsten Lodderstedt and Daniel Fett

OAuth 2.0 Security Best Current Practice

IMPLICIT FLOW ATTACKS

Don’t use the OAuth Implicit Grant any longer!



Source: Torsten Lodderstedt and Daniel Fett

OAUTH 2.0 FOR BROWSER-BASED APPS

David Waite (PingFederate)

OAuth 2.0 for Browser-Based Apps

OAUTH 2.0 FOR BROWSER-BASED APPS

Content-Security Policy

Use a unique redirect URI

NOT issue refresh tokens

OAuth 2.0 for Browser-Based Apps

OTHER KNOWN OAUTH 2.0 ATTACKS

- Lack of CSRF protection
- Authorization code leakage and replay
- Authorization code injection
- Open Re-directors
- State leakage and replay
- Insufficient Redirect URI matching
- Too powerful access tokens
- Mix-Up Attacks

OPEN REDIRECT !!

RELEASES

CVE-2019-3778: Spring Security OAuth 2.3.5, 2.2.4, 2.1.4, 2.0.17 Released

RELEASES  JOE GRANDJA FEBRUARY 21, 2019 0 COMMENTS

We have released Spring Security OAuth 2.3.5, 2.2.4, 2.1.4 and 2.0.17 to address CVE-2019-3778: Open Redirector in spring-security-oauth2. Please review the information in the CVE report and upgrade immediately.

“OAUTH 2.1” GRANT FLOWS

Client Type	Flow	Refresh Tokens
Confidential	**Authorization Code (PKCE)**	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	**Authorization Code (PKCE)**	--
~~Trusted~~	~~RO Password Creds~~	~~X~~
No Resource Owner	Client Credentials	--

PROOF KEY FOR CODE EXCHANGE BY OAUTH PUBLIC CLIENTS (PKCE)

("Pixy")

Mitigates authorization code attacks

Mitigates token leakage in SPAs

Proof Key for Code Exchange by OAuth Public Clients

PKCE - AUTHORIZATION REQUEST

GET https://authserver.example.com/authorize

?response_type=code

&client_id=abcdefg

&redirect_uri=https://client.abc.com/callback

&scope=api.read api.write

&state=xyz

&code_challenge=xyz...&code_challenge_method=

PKCE - TOKEN REQUEST

Client-Id=123, Client-Secret=456

POST https://authserver.example.com/token

Content-Type:
application/x-www-form-urlencoded

grant_type=authorization_code&code=ab23bhW56X

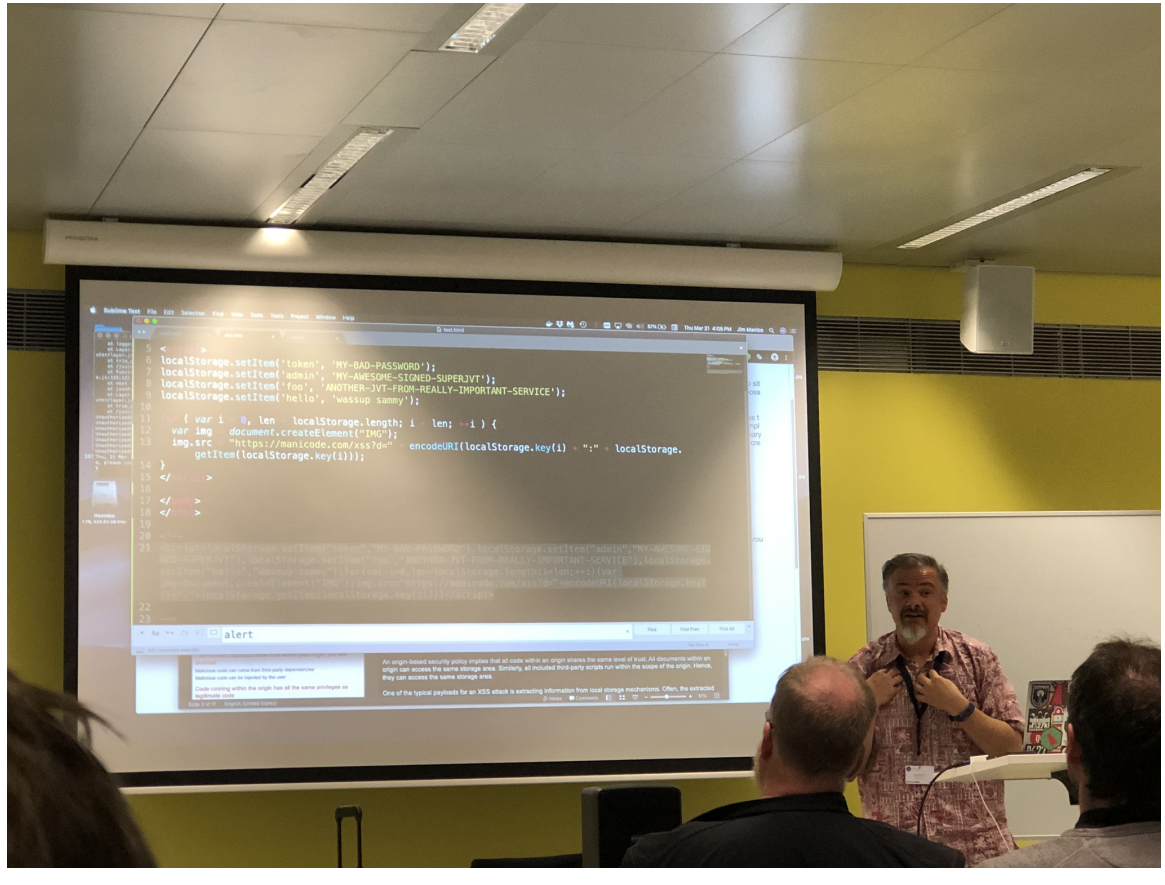
&redirect_uri=https://client.abc.com/callback

&client_id=123&client_secret=456

&code_verifier=4gth4jn78k_8

STEAL TOKENS VIA XSS

“XSS is Game-Over for OAuth 2” (Jim Manico)



OAUTH 2 ACCESS TOKEN JWT PROFILE

Vittorio Bertocci (Auth0)

JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens

OAUTH 2 ACCESS TOKEN JWT PROFILE

Required claims: iss, exp, aud, sub, client_id

Consider privacy restrictions for identity claims

Authorization claims according to SCIM Core (RFC7643):

- Groups
- Entitlements
- Roles

System for Cross-domain Identity Management (SCIM)

TOKEN BINDING

~~RFC8471: The Token Binding Protocol Version 1.0~~

~~RFC8472: (TLS) Extension for Token Binding Protocol Negotiation~~

~~RFC8473: Token Binding over HTTP~~

OAuth 2.0 Mutual TLS Client Authentication and Certificate-Bound Access Tokens

Google - Intent to Remove: Token Binding

FURTHER INTERNET-DRAFTS FOR OAUTH 2

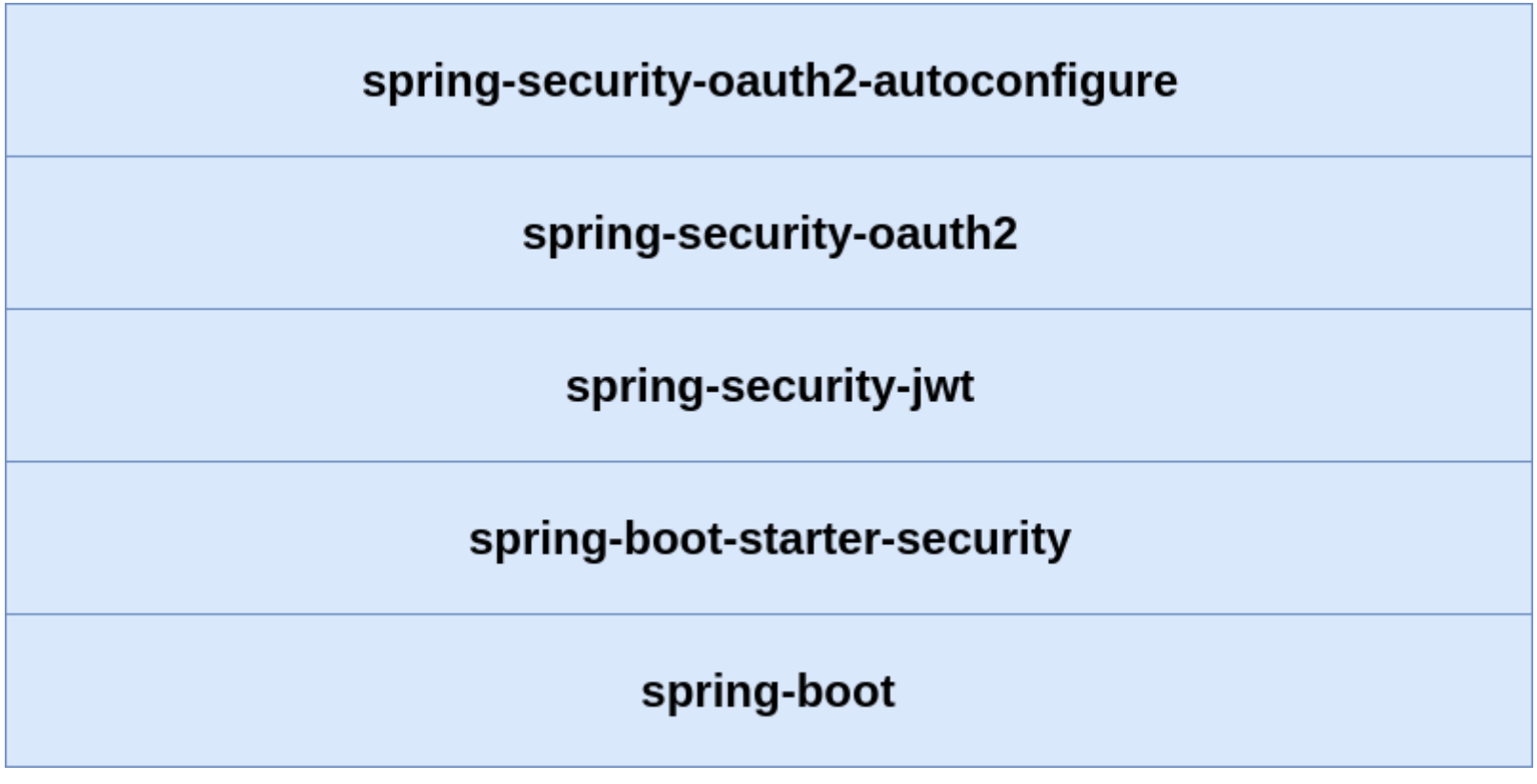
List of OAuth 2 Internet-Drafts (by date)

DEMO TIME

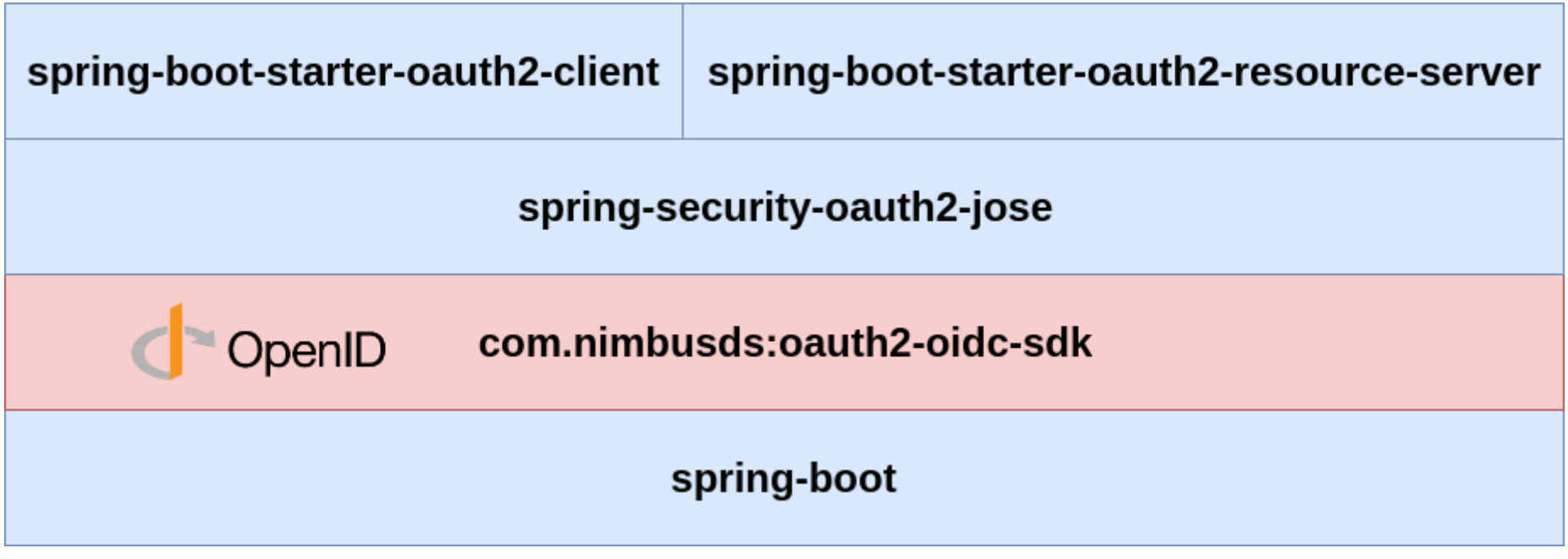
OAUTH 2.0 & OPENID CONNECT 1.0

WITH SPRING SECURITY 5

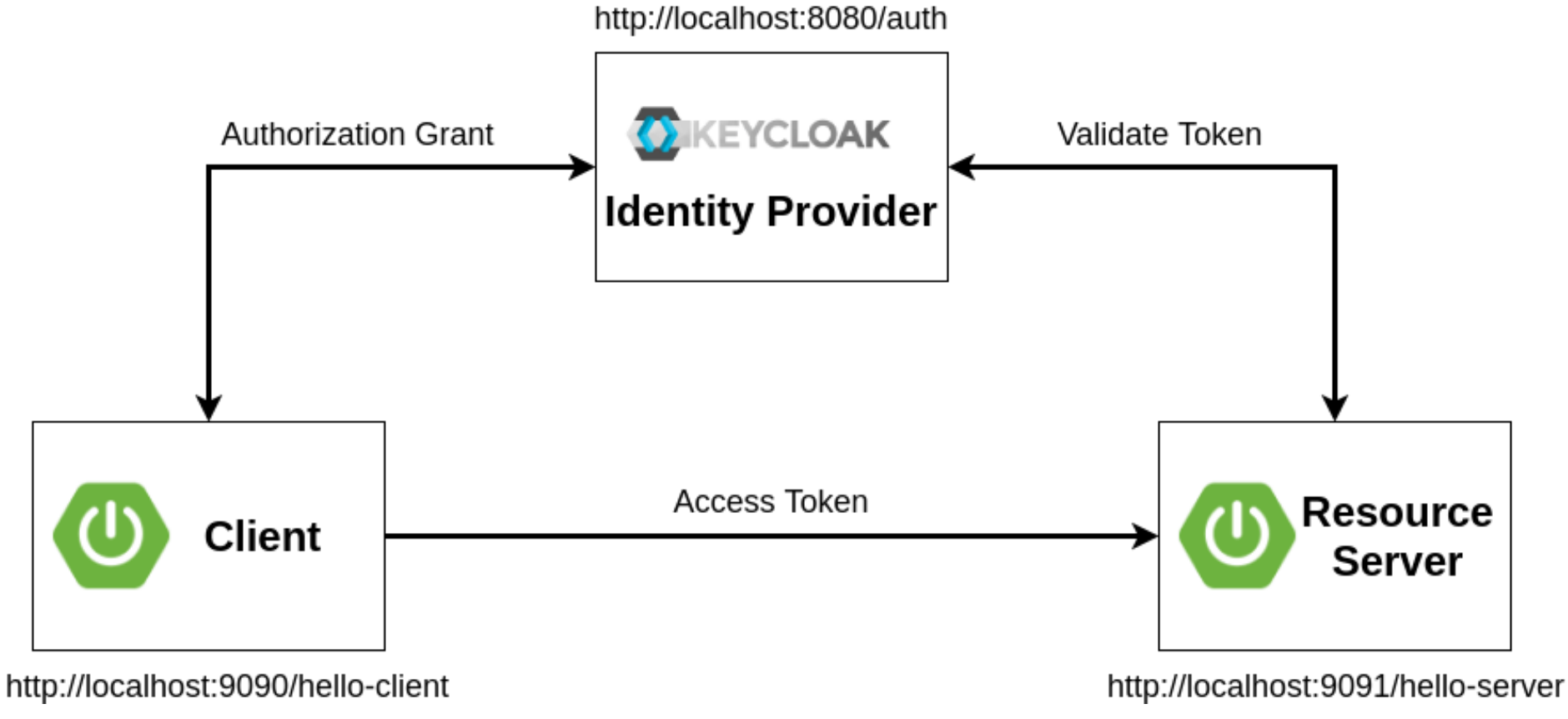
“LEGACY” SPRING SECURITY OAUTH 2 STACK



“NEW” SPRING SECURITY OAUTH 2 STACK



DEMO APPLICATION



WHAT'S NEW IN

SPRING SECURITY 5.2 & 5.3

Spring Security 5.2.0.M2 Released

RELEASES JOSH CUMMINGS APRIL 16, 2019 2 COMMENTS

On behalf of the community, I'm pleased to announce the release of Spring Security 5.2.0.M2! This release includes 100+ updates. You can find the highlights below:

OAuth 2.0

gh-6446 - Client Support for PKCE

PKCE isn't just for native or browser-based apps, but for any time we want to have a public client. Spring Security 5.2 introduces a secure way for backends to authenticate as public clients.

gh-5350 - OpenID Connect RP-Initiated Logout

gh-5465 - Ability to use symmetric keys with `JwtDecoder`

gh-5397 - Ability for `NimbusReactiveJwtDecoder` to take a custom processor

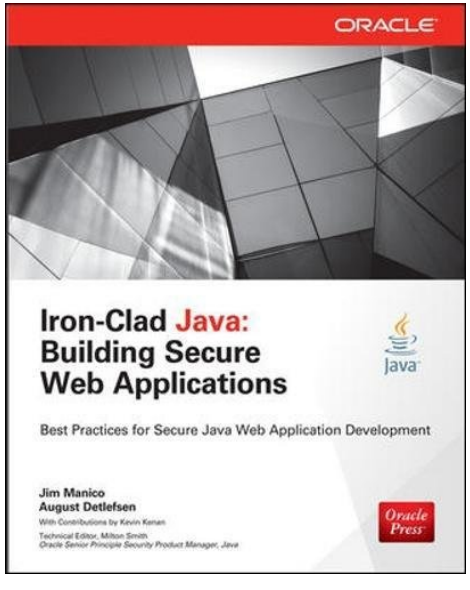
gh-6513 & gh-5200 - Support for Resource Server Token Introspection

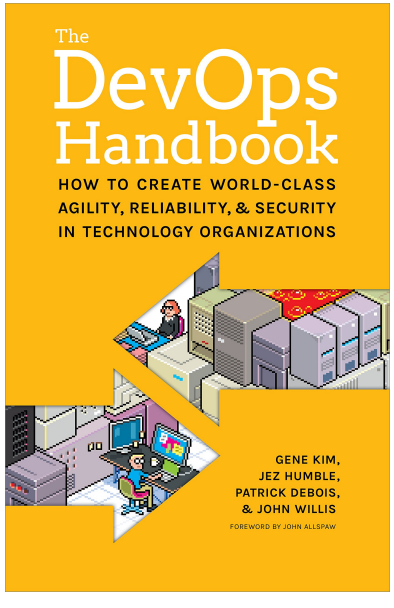
SPRING SECURITY 5.2

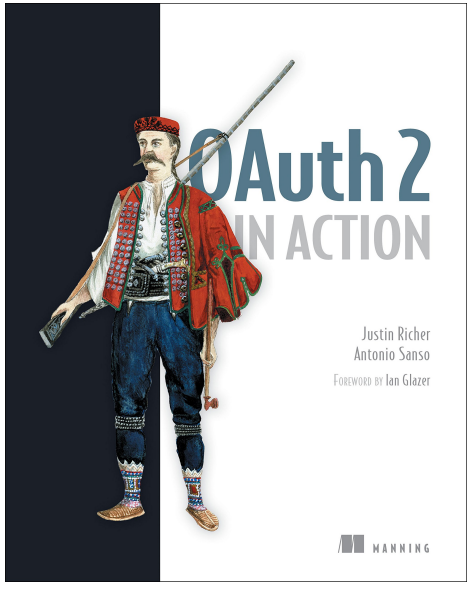
- Client Support for PKCE
- OpenID Connect RP-Initiated Logout
- Support for OAuth 2.0 Token Introspection
- Support for Resource Server Multi-tenancy

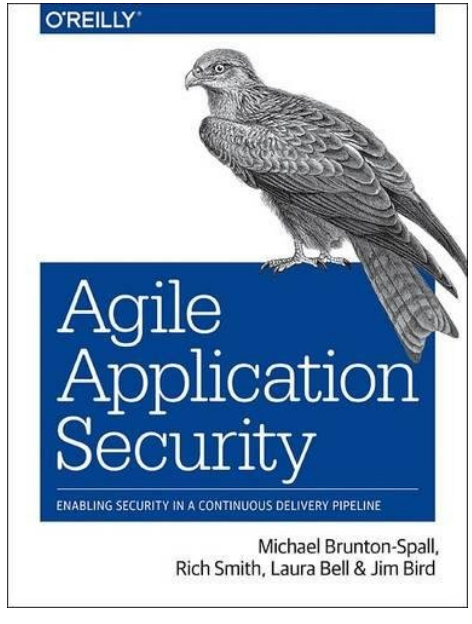
Spring Security 5.2.0 M2 GitHub Issues
Spring Security 5.2.0 RC1 GitHub Issues

BOOK REFERENCES









Q&A

https://www.novatec-gmbh.de
https://blog.novatec-gmbh.de
andreas.falk@novatec-gmbh.de
Twitter: @andifalk

ONLINE REFERENCES

- RFC 6749: The OAuth 2.0 Authorization Framework
- RFC 6750: OAuth 2.0 Bearer Token Usage
- RFC 6819: OAuth 2.0 Threat Model and Security Considerations
- RFC 7636: Proof Key for Code Exchange (“Pixy”)
- OpenID Connect Core 1.0
- OpenID Connect Dynamic Client Registration 1.0
- OpenID Connect Discovery 1.0
- RFC 7519: JSON Web Token (JWT)
- JSON Web Token Best Current Practices
- 4. OAuth Security Workshop 2019 event web page
- Why you should stop using the OAuth implicit grant
- OAuth 2.0 Security Best Current Practice
- OAuth 2.0 for Browser-Based Apps
- OAuth 2.0 Mutual TLS Client Authentication and Certificate-Bound Access Tokens
- JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
- Spring Security