#spring

SECURING MICROSERVICES WITH OPENID CONNECT AND SPRING SECURITY 5.1 [WORKSHOP]

Andreas Falk

@andifalk

Slides: https://andifalk.github.io/oidc-workshop-spring-io-2019
Source-Code: https://github.com/andifalk/oidc-workshop-spring-io-2019.git

ANDREAS FALK

Novatec Consulting GmbH

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

NOVATEC OFFICES

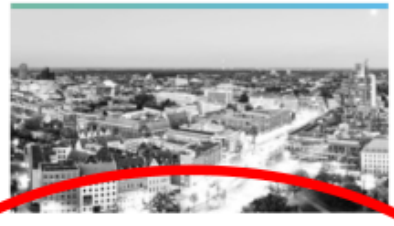
Leinfelden- Echterdingen

Berlin

Frankfurt

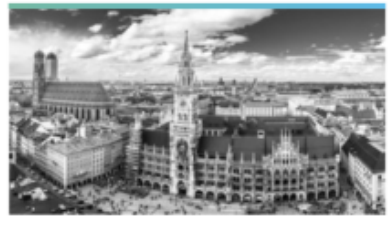
Hamburg

Hannover

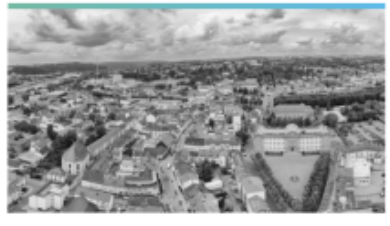


Karlsruhe

München



Zweibrücken



Granada

AGENDA

Intro to OAuth 2.0

Intro to OpenID Connect 1.0

Hands-On Part with Spring Security 5.1

What's new in Spring Security 5.2 and 5.3

IN THE MEANTIME...

git clone https://tinyurl.com/yylensdd oidc_workshop

import as gradle project into your Java IDE

AND

copy/extract keycloak.zip from provided USB sticks

OR follow setup instructions at
https://tinyurl.com/y3rgohqo

OAUTH 2.0

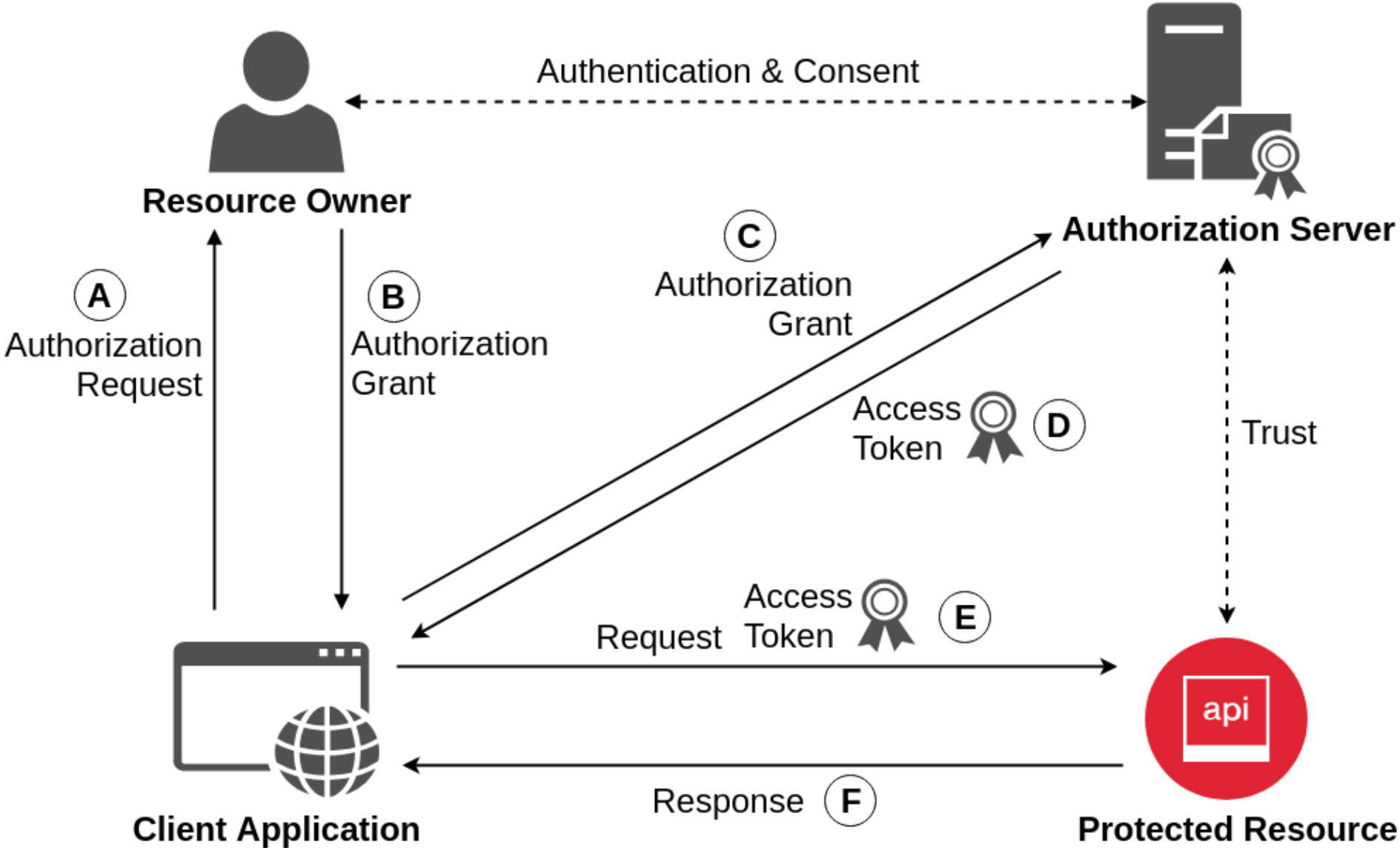
101

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

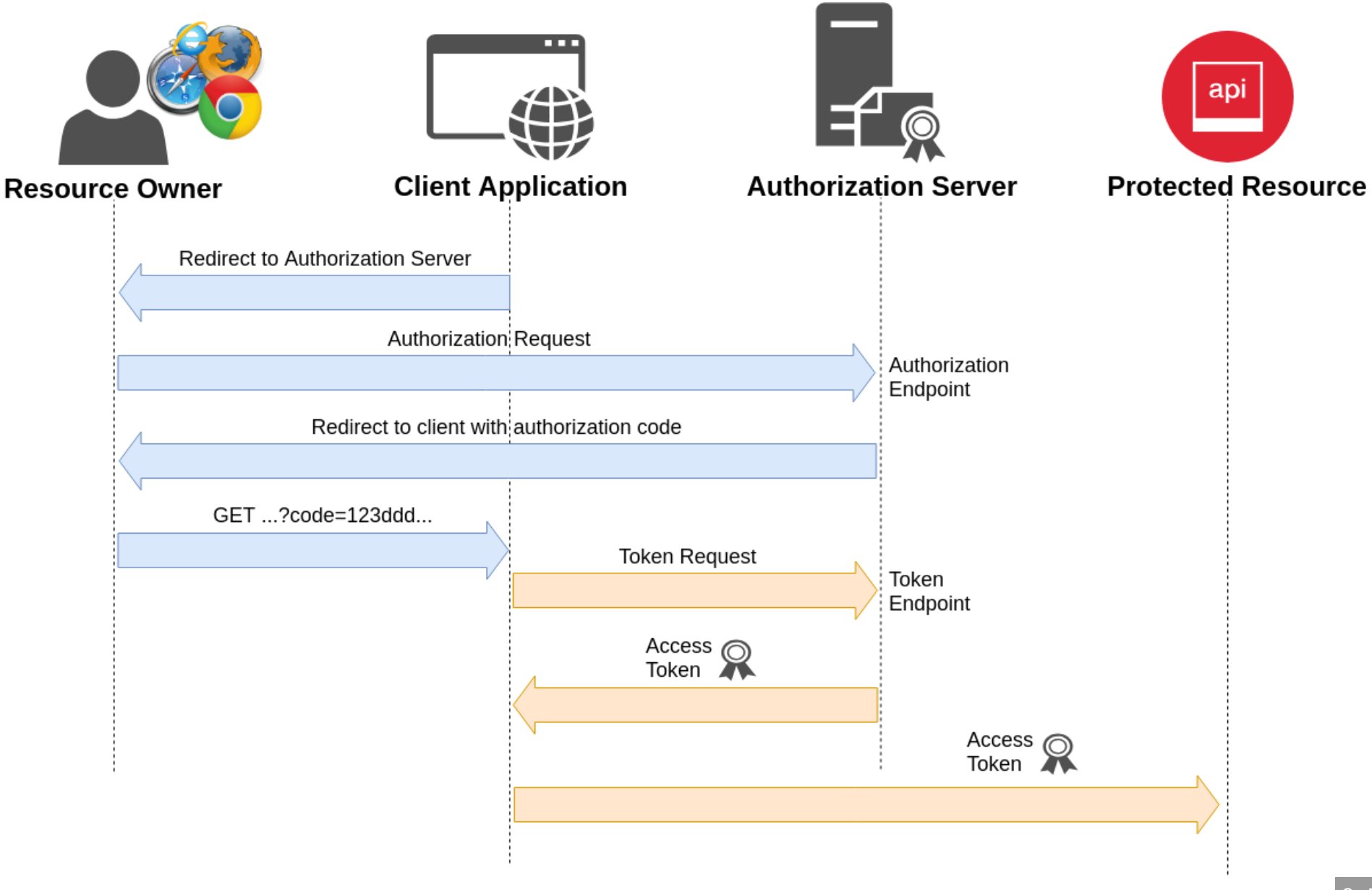
OAUTH 2.0 MODEL



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



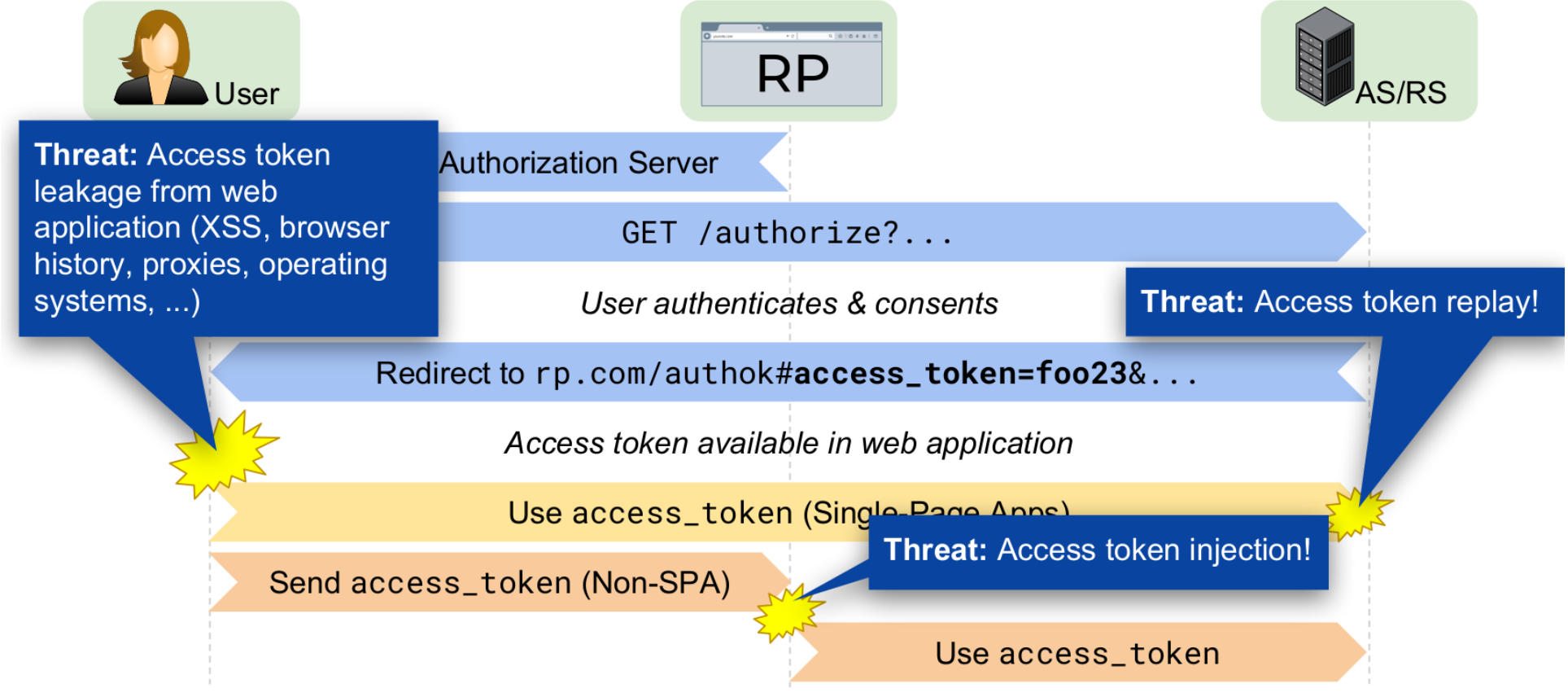
WE'LL SEE

AUTHORIZATION CODE GRANT FLOW

in action as part of Intro Lab

IMPLICIT FLOW ATTACKS

Don’t use the OAuth Implicit Grant any longer!



Source: Torsten Lodderstedt and Daniel Fett

“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=S256

PKCE - TOKEN REQUEST

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

OPENID CONNECT 1.0 101

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

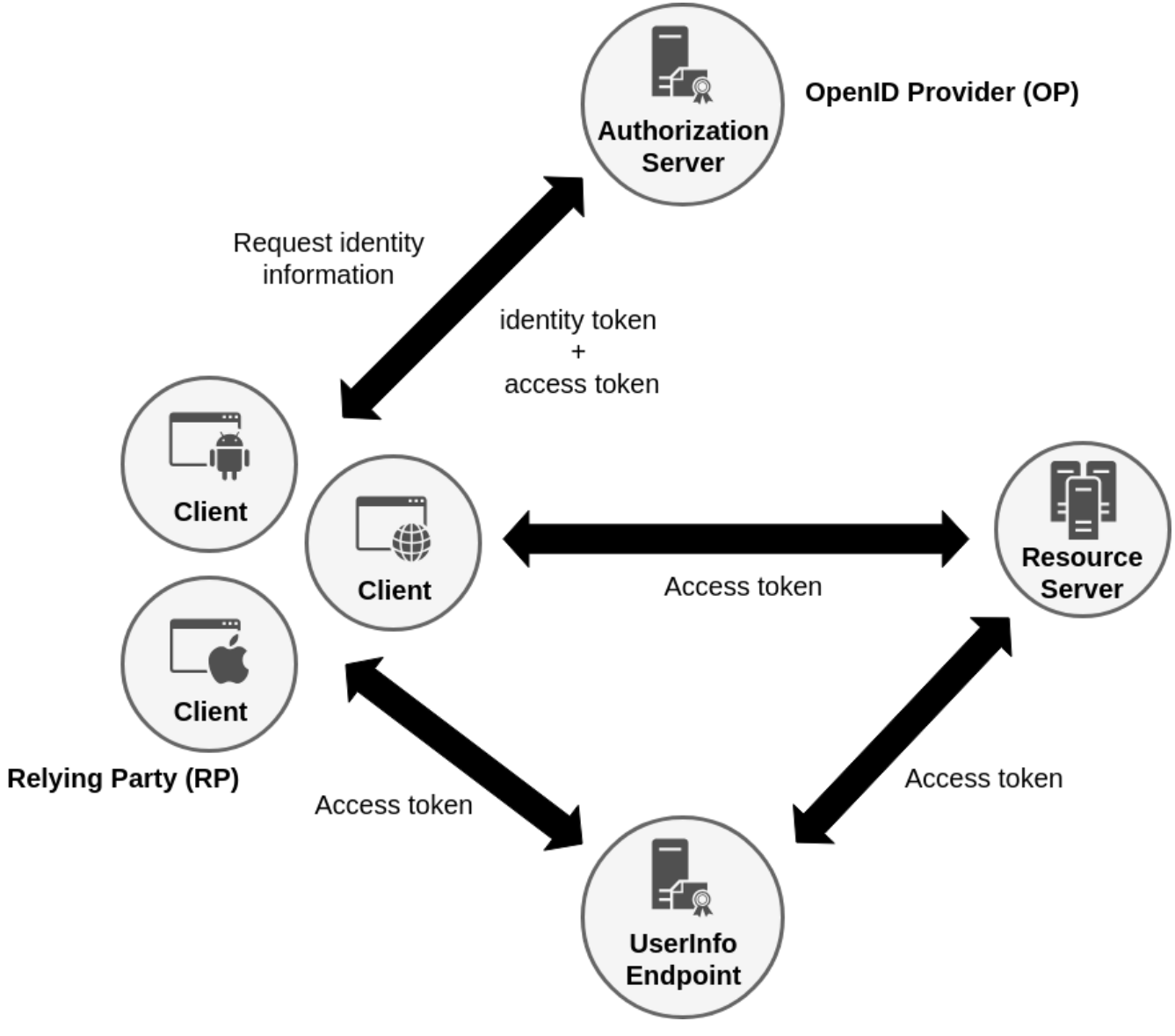
Follow

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

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"
}
```

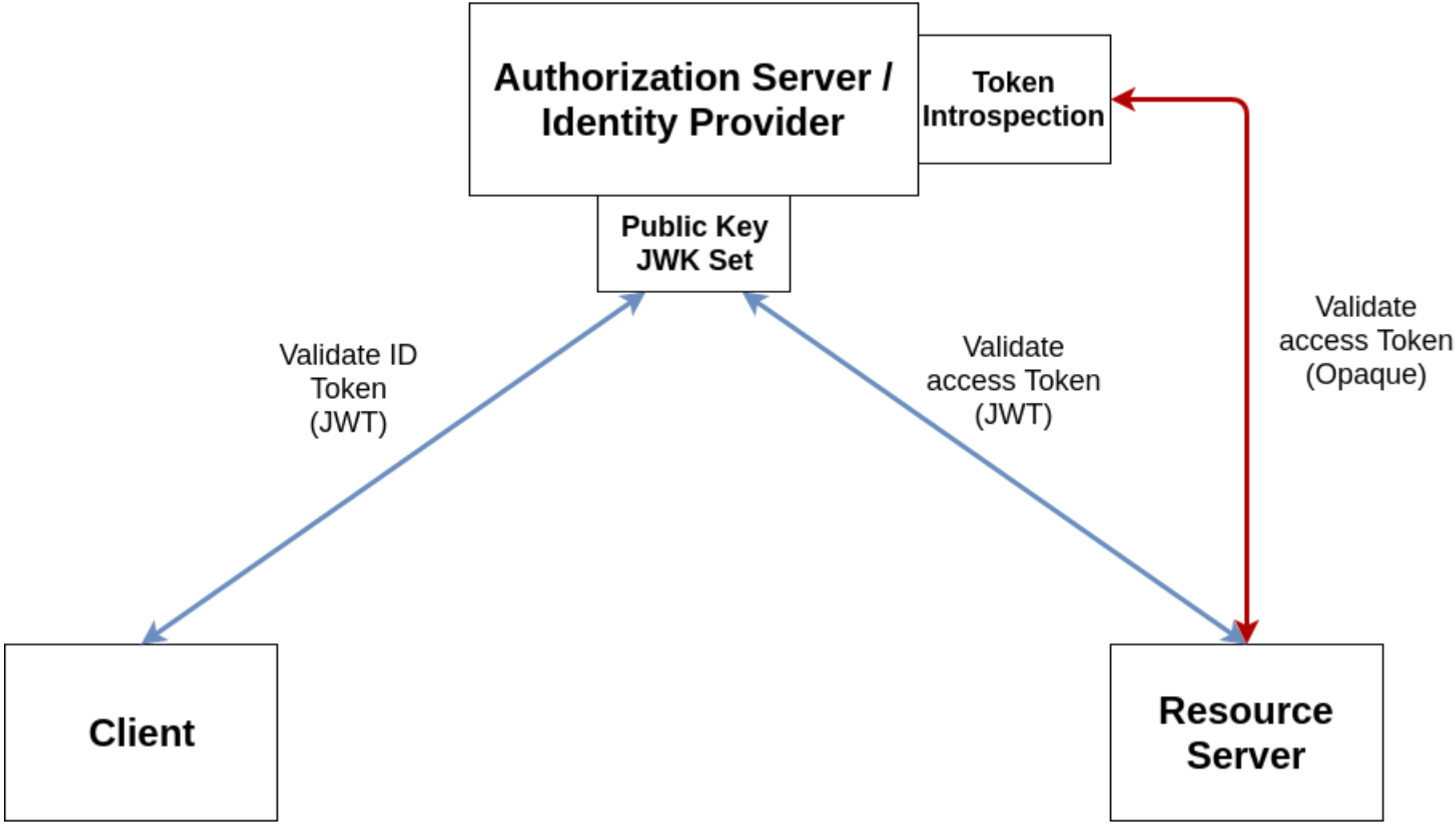
Payload

```
{
  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"
}
```

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",
  ...
}
```

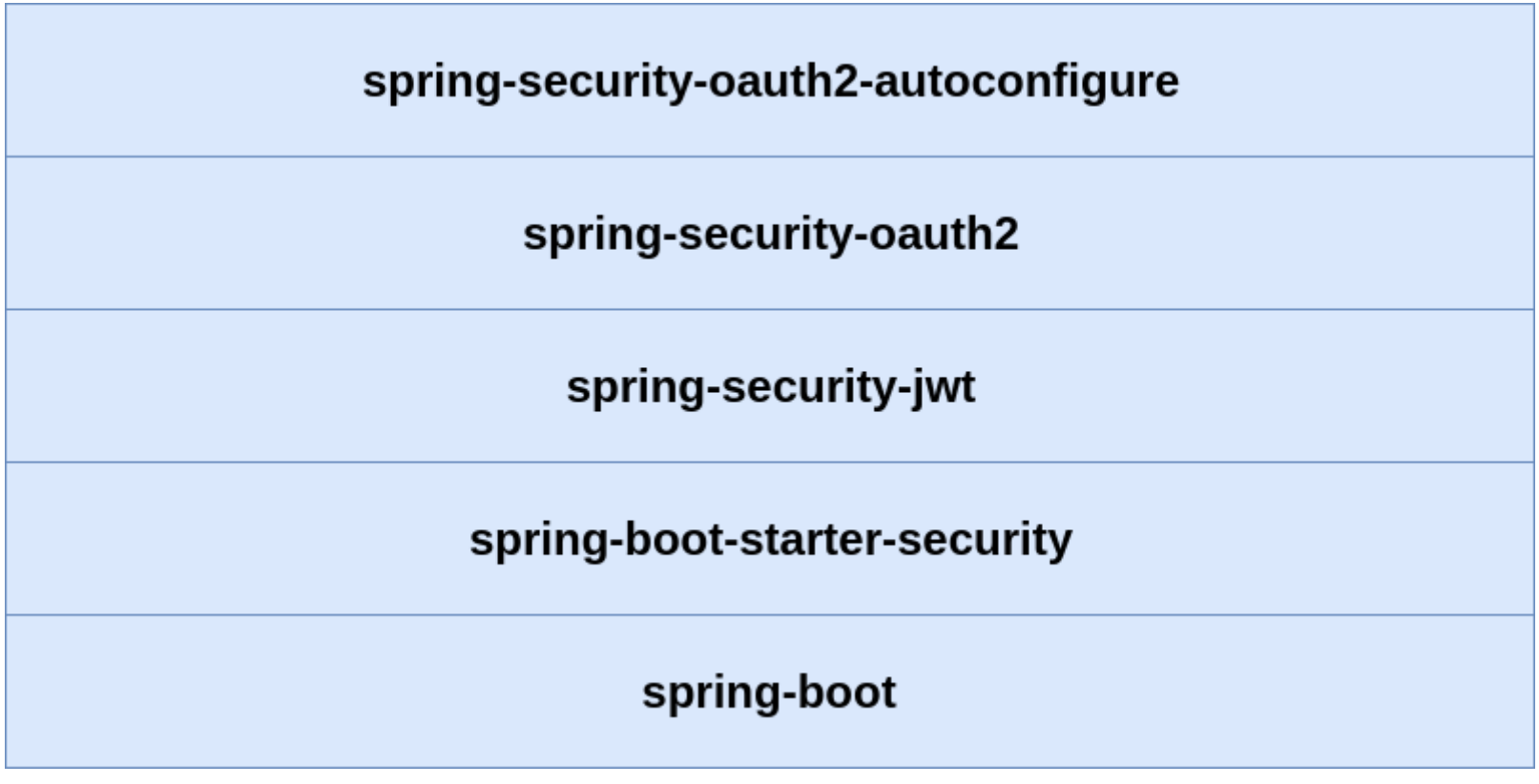
OpenID Connect Discovery 1.0

HANDS-ON PART

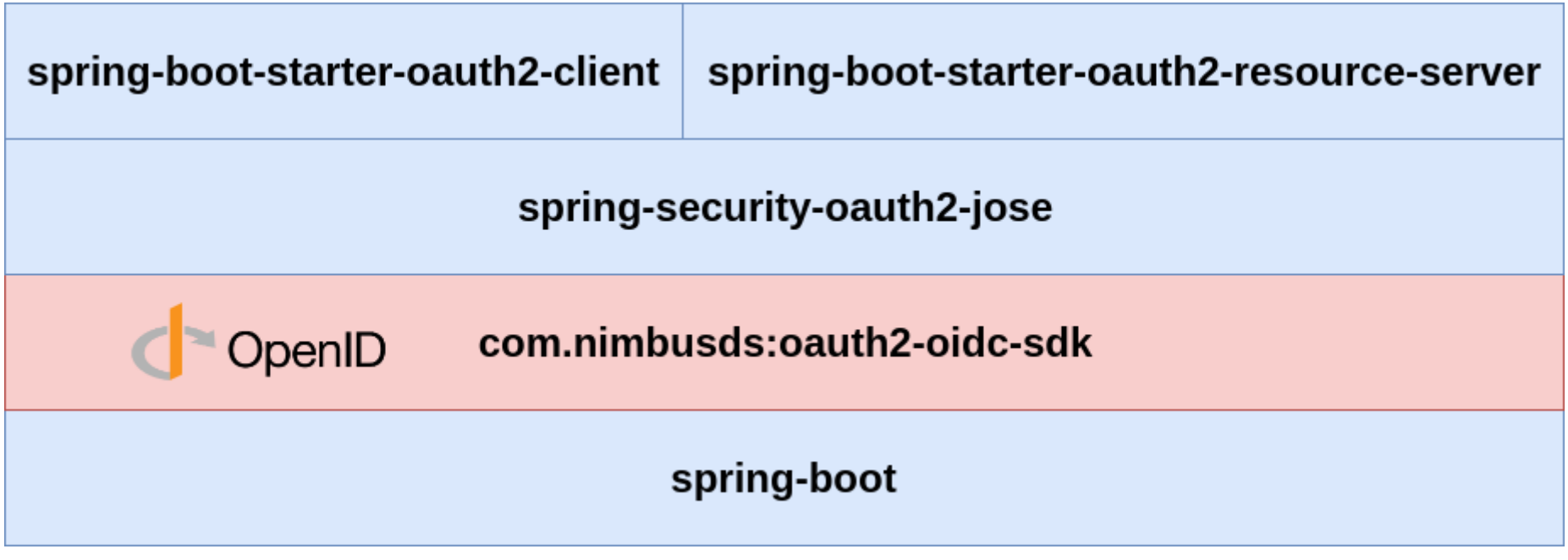
OAUTH 2.0 & OPENID CONNECT 1.0

WITH SPRING SECURITY 5

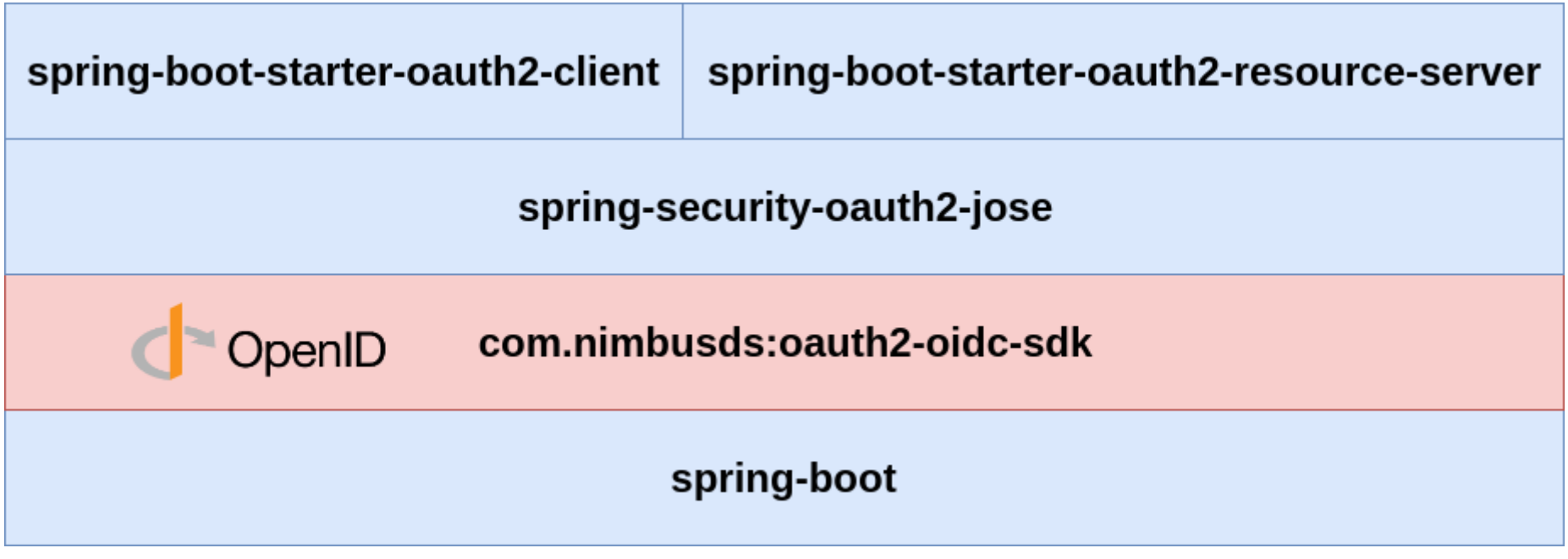
“LEGACY” SPRING SECURITY OAUTH 2 STACK



“NEW” SPRING SECURITY OAUTH 2 STACK



“NEW” SPRING SECURITY OAUTH 2 STACK



WE WILL USE THIS STACK !!!

LET'S CODE!!

git clone https://tinyurl.com/yylensdd oidc_workshop

import as gradle project into your Java IDE

AND

copy/extract keycloak.zip from provided USB sticks

SETUP KEYCLOAK

Follow setup instructions at:
https://tinyurl.com/y3rgohqo

1. Download the binary distribution:
   https://tinyurl.com/y2dcuu2o
2. Extract the downloaded **keycloak-6.0.1.zip** file
3. Download the prepared keycloak configuration:
   https://tinyurl.com/y3gpuwkj
4. Extract the downloaded file **keycloak_data.zip** into the sub directory *standalone* of extracted keycloak

START KEYCLOAK

Start keycloak on Linux/Mac OS

```
[keycloak_install_dir]/bin/standalone.sh
```

or on Windows

```
[keycloak_install_dir]\bin\standalone.bat
```

SPRING

Session

Look who's talking!

How to secure your Spring apps with Keycloak

Thomas Darimont - Codecentric

Session tomorrow at 12:30

INTRO-LAB

AUTHORIZATION CODE GRANT FLOW IN ACTION

See intro-labs/auth-code-demo for instructions

HANDS-ON APPLICATION

AN ONLINE BOOK LIBRARY

Administer Books

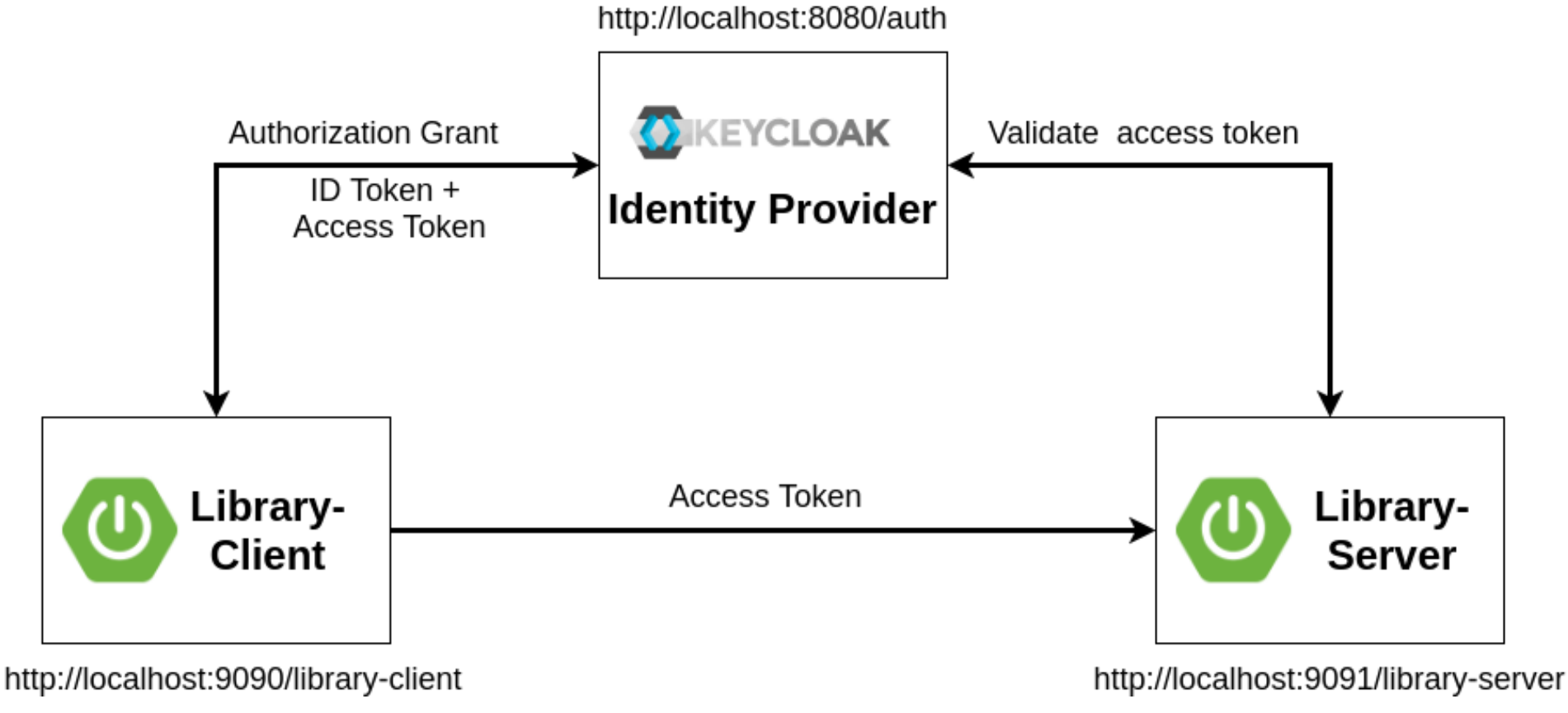
List available Books

Borrow a Book

Return a borrowed Book

Administer Library Users

HANDS-ON APPLICATION



HANDS-ON LABS

- Lab 1: OAuth2/OIDC Resource Server
- Lab 2: OAuth2/OIDC Client (Auth Code Flow)
- Lab 3: OAuth2/OIDC Client (Client-Credentials Flow)
- Lab 4: OAuth2/OIDC Testing Environment

Please follow lab tutorials in GitHub Repo

WHAT'S NEW IN SPRING SECURITY 5.2 & 5.3

SPRING SECURITY 5.2 (08/2019)

- Client Support for PKCE
- OpenID Connect RP-Initiated Logout
- Support for OAuth 2.0 Token Introspection
- Resource Server Multi-tenancy (Servlet & Reactive)
- Use symmetric keys with JwtDecoder
- JWT Flow API in Test Support

Spring Security 5.2.0 M2 GitHub Issues
Spring Security 5.2.0 M3 GitHub Issues
Spring Security 5.2.0 RC1 GitHub Issues

OAUTH 2.0 TOKEN INTROSPECTION

Opaque Tokens

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http)
            throws Exception {
        http.oauth2ResourceServer()
                .opaqueToken()
                    .introspectionUri(this.introspectionUri)
                    .introspectionClientCredentials(
                        this.clientId, this.clientSecret);
    }
}
```

https://github.com/spring-projects/spring-security/issues/5200

RESOURCE SERVER MULTI-TENANCY

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {

    @Override protected void configure(HttpSecurity http) {
        http.oauth2ResourceServer()
            .authenticationManagerResolver(
                multitenantAuthenticationManager());
    }

    @Bean AuthenticationManagerResolver<HttpServletRequest>
        multiTenantAuthMgr() {...}

    AuthenticationManager jwt() {...}
    AuthenticationManager opaque() {...}
}
```

https://github.com/spring-projects/spring-security/issues/5351
https://github.com/spring-projects/spring-security/issues/6727

USE SYMMETRIC KEYS WITH JWTDECODER

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {

  @Value("${spring.security.oauth2.resourceserver.
                jwt.key-value}") RSAPublicKey key;

  @Override protected void configure(HttpSecurity http) {
    http.oauth2ResourceServer().jwt().decoder(jwtDecoder());
  }

  @Bean JwtDecoder jwtDecoder() throws Exception {
      return NimbusJwtDecoder.
              withPublicKey(this.key).build();
  }
}
```

https://github.com/spring-projects/spring-security/issues/5465

JWT FLOW API IN TEST SUPPORT

```
public class OAuth2ResourceServerTest {

    @Test
    public void testRequestPostProcessor() {
      mockMvc.perform(get("/message")
      .with(mockAccessToken().scope("message:read")))
      .andExpect(status().isOk())

      mockMvc.perform(get("/")
      .with(jwt().claim(SUB, "the-subject")))
      .andExpect(status().isOk())
    }
}
```

https://github.com/spring-projects/spring-security/issues/6634

SPRING SECURITY 5.3

Support OAuth 2.0 Authorization Server:

- OAuth 2.0 Authorization Code Grant
- OpenID Connect 1.0 (Authorization Code Flow)
- PKCE
- OAuth 2.0 Client Credentials Grant
- JWT Access Token format
- JWK Set Endpoint
- Opaque Access Token format
- OAuth 2.0 Token Revocation

Spring Security 5.3.0 GitHub Issues

SPRING 15

Session

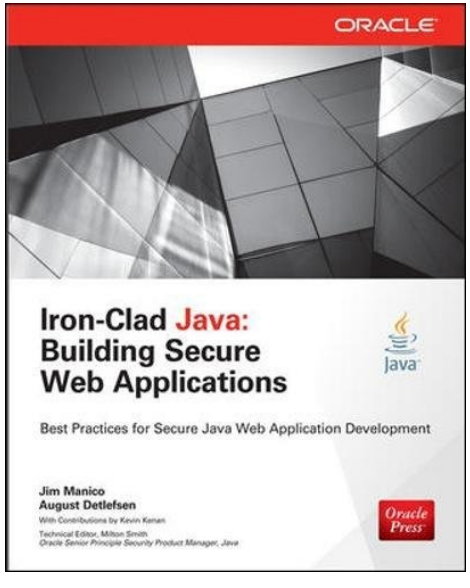
Look who's talking!

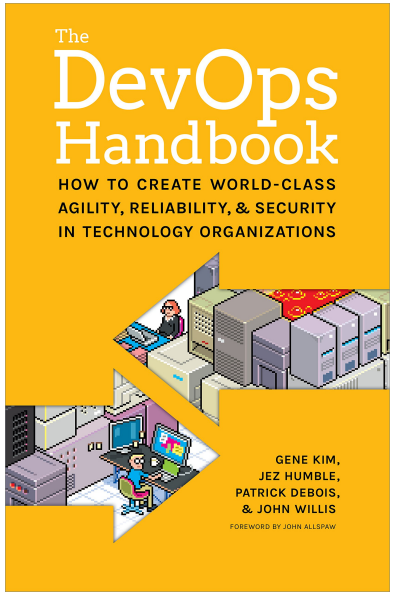
Implementing Microservices Security Patterns & Protocols with Spring Security 5

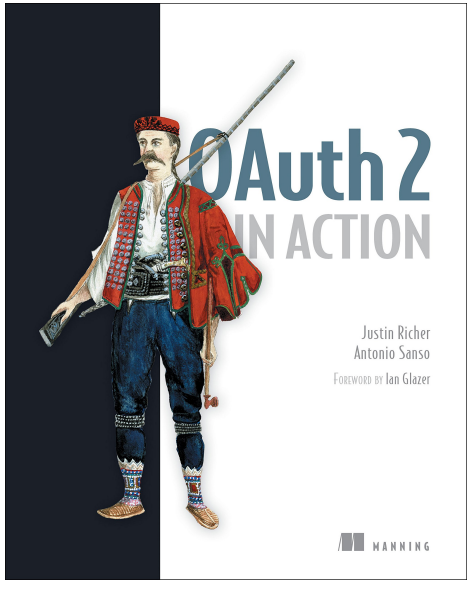
Joe Grandja - Pivotal

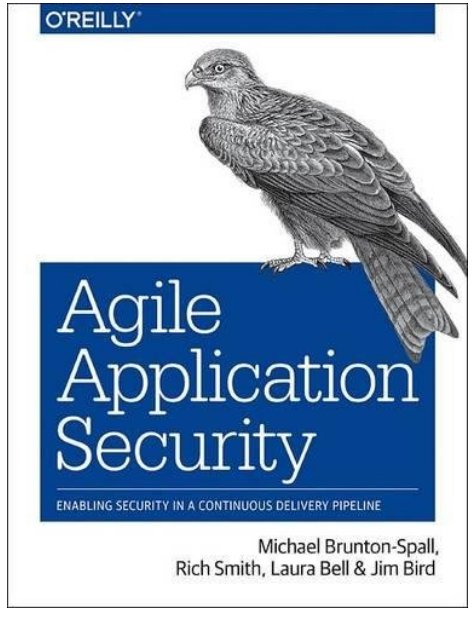
Session tomorrow at 14:30

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