JavaCro'17

Session is Dead
Long live JWT

JavaCro Rovinj, Croatia – May 2017

Hrvoje Crnjak

Software Developer @ Five

@HrvojeCrnjak

FIVE

Session vs. **JWT**

Reference vs. Data object

Stateful vs. Stateless

Debunked

(JSON Web Token)

NOT Authentication Protocol

NOT Authorization Protocol

Debunked

(JSON Web Token)

NOT Protocol

Standard for representing claims (data)

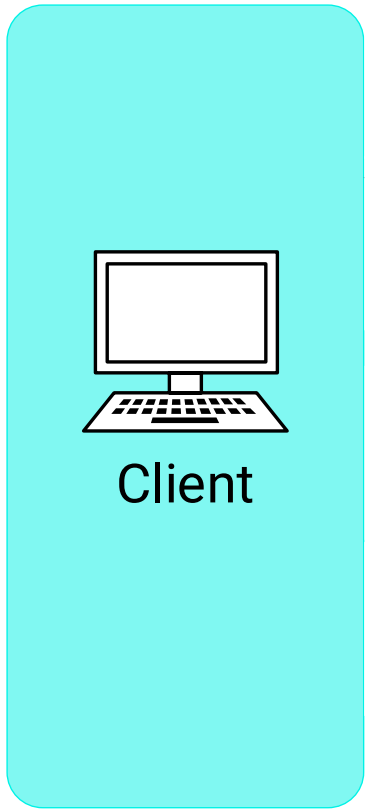
Then what's the catch?

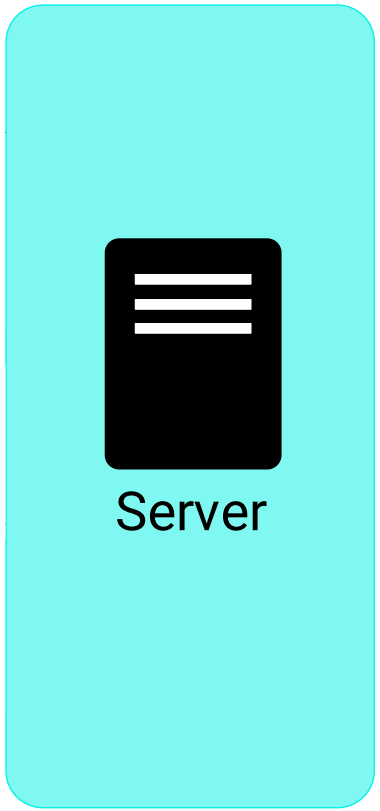
The catch is in how we **use JWT**s

Client – Server communication

Session ID + Cookie

https://app.yourapp.com





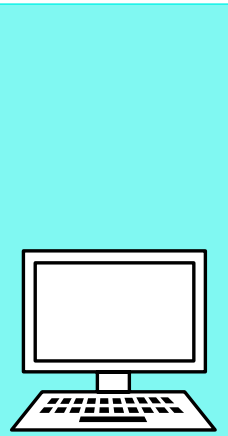
Create session and store it on server-side

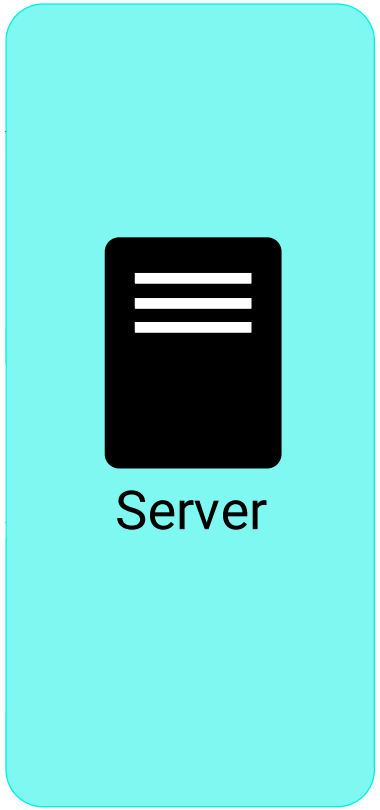
Find stored session based on Session ID

Client – Server communication

JWT + Authorization header

https://app.yourapp.com





Create token with necessary User data (basic info, roles, etc.)

Validate token and extract data

Session vs. JWT **analysis**

Session

- Authentication
  - On the Server
    - Create Session
    - Store Session
  - Response
    - Session ID
    - Client storage in Cookie
- Accessing Resource
  - Cookie header with Session ID
  - Find Session based on ID

JWT

- Authentication
  - On the Server
    - Create Token
    - **DO NOT** store Token
  - Response
    - Token
    - Client storage up to JS
- Accessing resource
  - Authorization header with JWT
  - Read data from JWT itself

JUnit

JWT

- JSON Web Token
- Pronounced “*jot*” /dʒɒt/
- Open standard
  - IETF - RFC 7519
  - Proposed in May 2015

JWT

- Standard for representing claims between parties
  - Transferred securely and in compact and URL-safe way
- Compact and URL-safe
  - Base64 encoded with URL-safe alphabet
  - (- , _) instead of (+ , /)
- Claim
  - A piece of information about a subject (user)
  - Represented as name/value pair
    - Claim Name - always string
    - Claim Value - any JSON value

JWT

- Claim names
  - Registered Claim names
    - Defined by the JWT standard
  - Public Claim names
    - Defined by 3rd parties and registered at IANA
    - Usually defined by standards that rely on JWT
      - example : OpenId Connect (given_name, family_name, nicnkname, ...)
  - Private Claim names
    - Custom names defined by JWT users
    - Anyone creating and using JWTs in their App
- All Claims are **optional**

JWT

- Registered Claim names

```
{
        jti : “JWT ID”
}
```

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”

}

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”

}

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”,

aud : “Audience – intended recipients”

}

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”,

aud : “Audience – intended recipients”,

iat : “Time at which JWT was issued”

}

Seconds Since the Epoch

MAY 12, 1984

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”,

aud : “Audience – intended recipients”,

iat : “Time at which JWT was issued”

}

Seconds Since the Epoch

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”,

aud : “Audience – intended recipients”,

iat : “Time at which JWT was issued”,

nbf : “Time **before** which JWT must not be accepted”

}

JWT

- Registered Claim names

{

jti : “JWT ID”,

iss : “JWT Issuer”,

sub : “Subject of JWT”,

aud : “Audience – intended recipients”,

iat : “Time at which JWT was issued”,

nbf : “Time **before** which JWT must not be accepted”,

exp : “Time **after** which JWT must not be accepted”

}

JWT

- Securing JWT
  - Signed
    - Claims in “plain text”
    - Guaranteed token integrity
    - Defined through JWS (JSON Web Signature) standard
  - Encrypted
    - Claims encrypted
    - Guaranteed token integrity and data privacy
    - Defined through JWE (JSON Web Encryption) standard

JWT

- Securing JWT
  - Signed
    - Claims in “plain text”
    - Guaranteed token integrity
    - Defined through JWS (JSON Web Signature) standard
  - Encrypted
    - Claims encrypted
    - Guaranteed token integrity and data privacy
    - Defined through JWE (JSON Web Encryption) standard

JWT

- Signing JWT
  - A number of supported algorithms
    - Defined in JWA (JSON Web Algorithms) standard
  - Most notable are
    - HS256 – HMAC using SHA-256
      - Symmetric key alg.
      - Offers integrity and authenticity
    - RS256 – RSA using SHA-256
    - ES256 – ECDSA using SHA-256
      - Asymmetric key alg.
      - Offers integrity, authenticity and non-repudiation

JWT

- Composing JWT

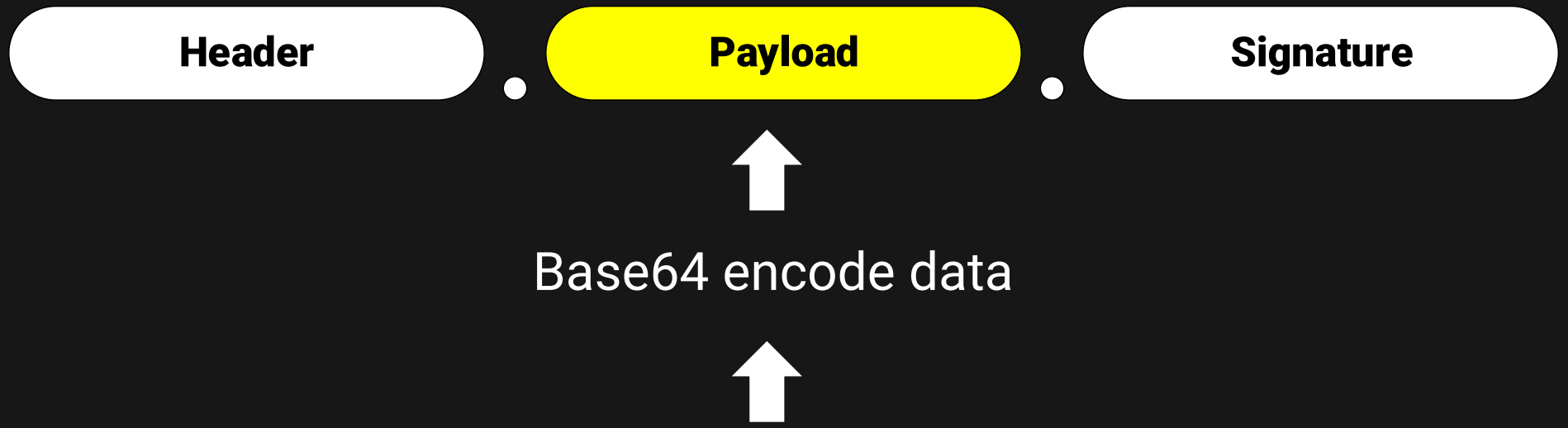
JWT

- Composing JWT

  - 3 parts
    - Header
      - Holds info about algorithm used for signing
    - Payload
      - Holds claims
    - Signature
  - Separated by “*dot*” and Base64 encoded

JWT

- Composing JWT

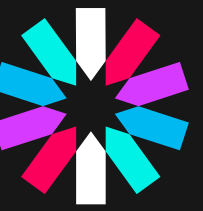


JWT

- Composing JWT

Base64 encode data

```
{
  alg : “HS256",
  typ : “JWT“
}
```



JWT

- Composing JWT

Base64 encode data

SIGN (
encoded_Header + “.” + encoded_Payload
, **secretKey**
)

JWT

- Composing JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

FAQ #1

Who can create JWTs?

Anyone who knows **how** to create them

- Pick a library and start coding!
  - jjwt , java-jwt , etc.

FAQ #2

Who can read JWTs?

Anyone who knows how to **read** them

- You need
  - Library (again)
  - A secret key if you want to verify JWTs validity
    - For clients (like web browsers) that’s not necessary
    - For servers it should be

FAQ #3

How can JWTs be transferred between parties?

The standard **doesn't** specify that

- Possible scenarios
  - Authorization header
    - **Authorization : Bearer** ..JWT... scheme used
    - Most common, standardized
  - Custom header
  - Part of URL
  - Part of request body

Pros & Cons

JWT is **Self-Contained**

Self-contained

- All the relevant data is **inside** JWT
  - User information represented through claims
  - Data required to check its validity
- **Benefits**
  - Client-side
    - Data can be read directly from token
    - User ID, name, email, roles, …
  - Server-side
    - On the next slide …

Session ID

JSESSIONID=8D4D409560B
D4D0CCCA00105325115F3

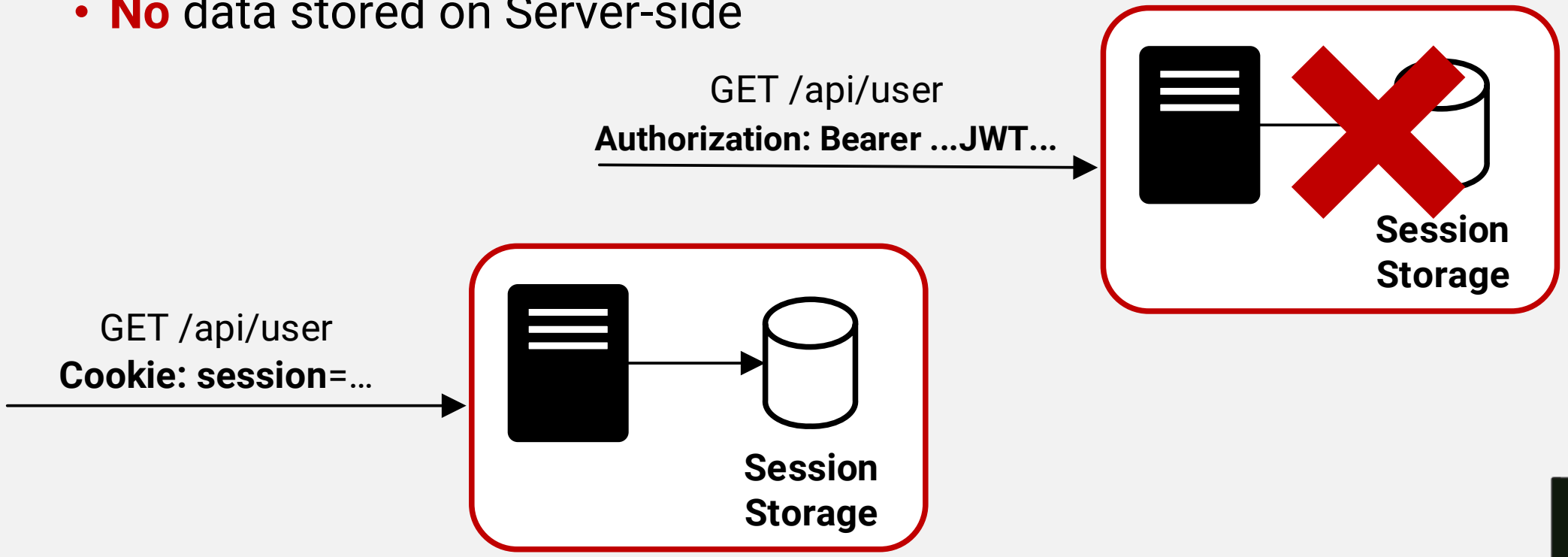
JWT

```
{
  sub : "1234567890",
  name : "John Doe",
  role : “ADMIN”
}
```

JWT is **Stateless**

Stateless

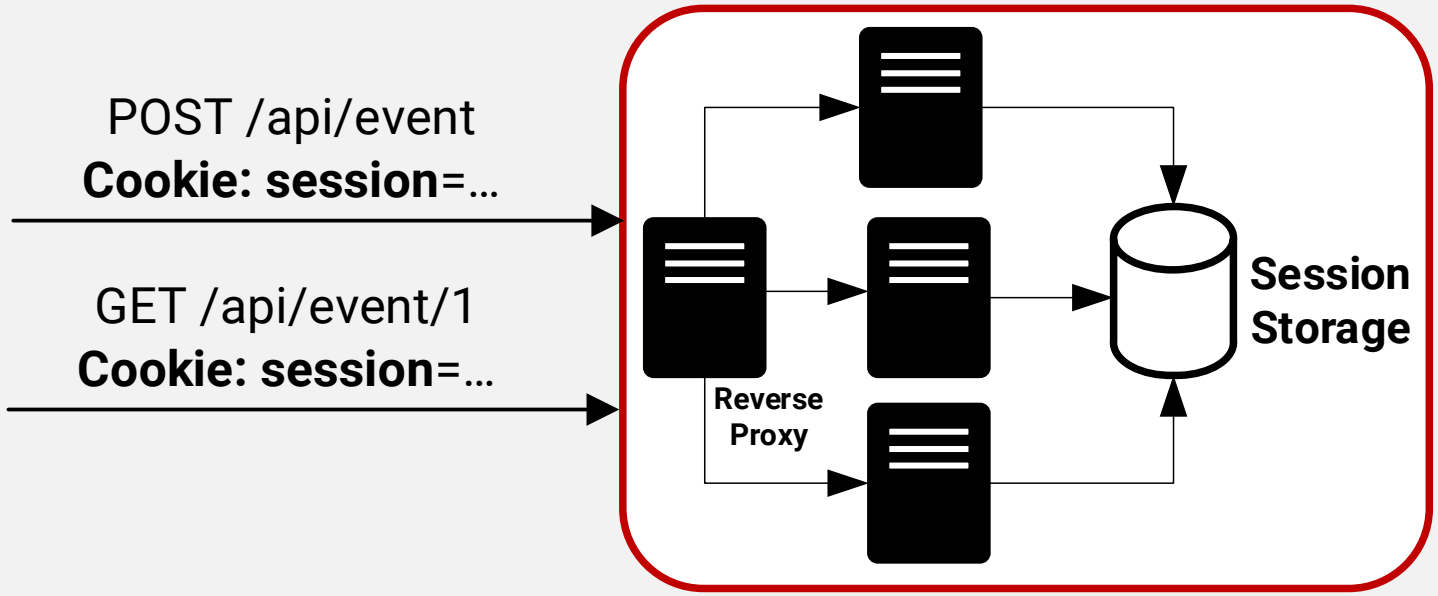
- JWT is self-contained
  - **No need** for State
  - **No** data stored on Server-side



JWT is **Scalable**

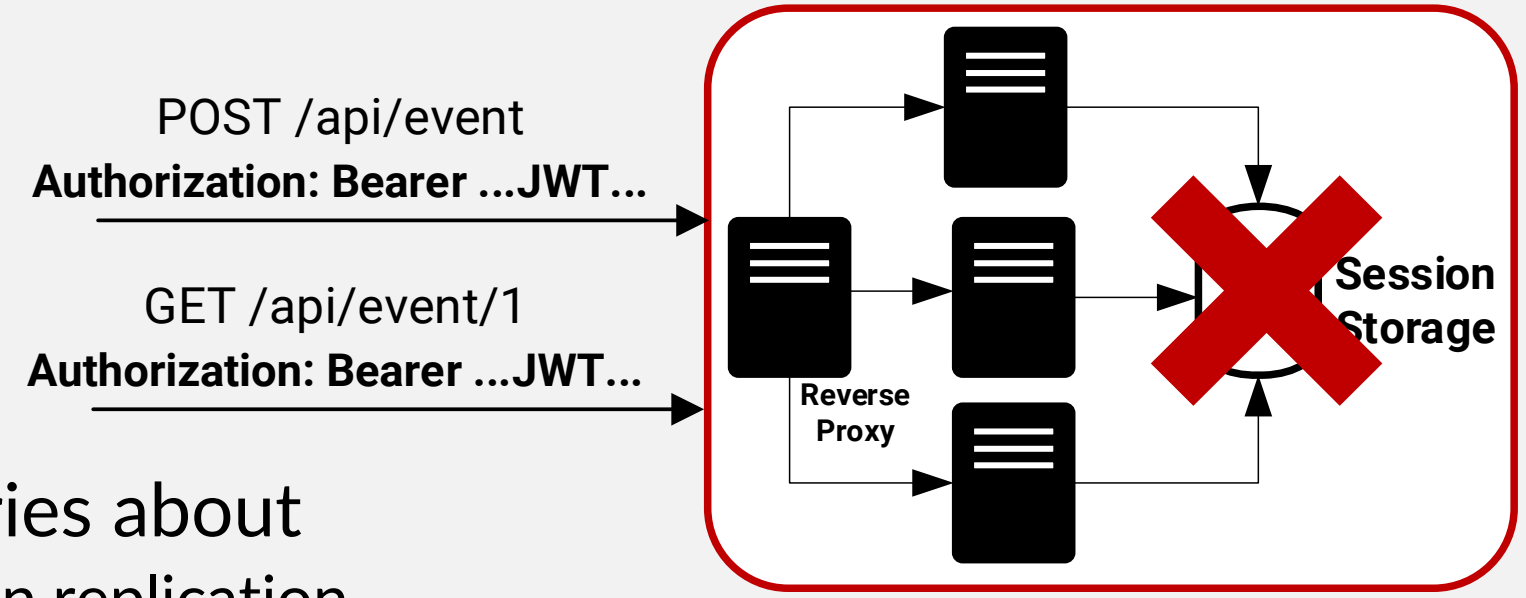
Scalable

- State is **evil**
  - When we use Session…



Scalable

- State is **evil**
  - No State (Session) **no problem!**

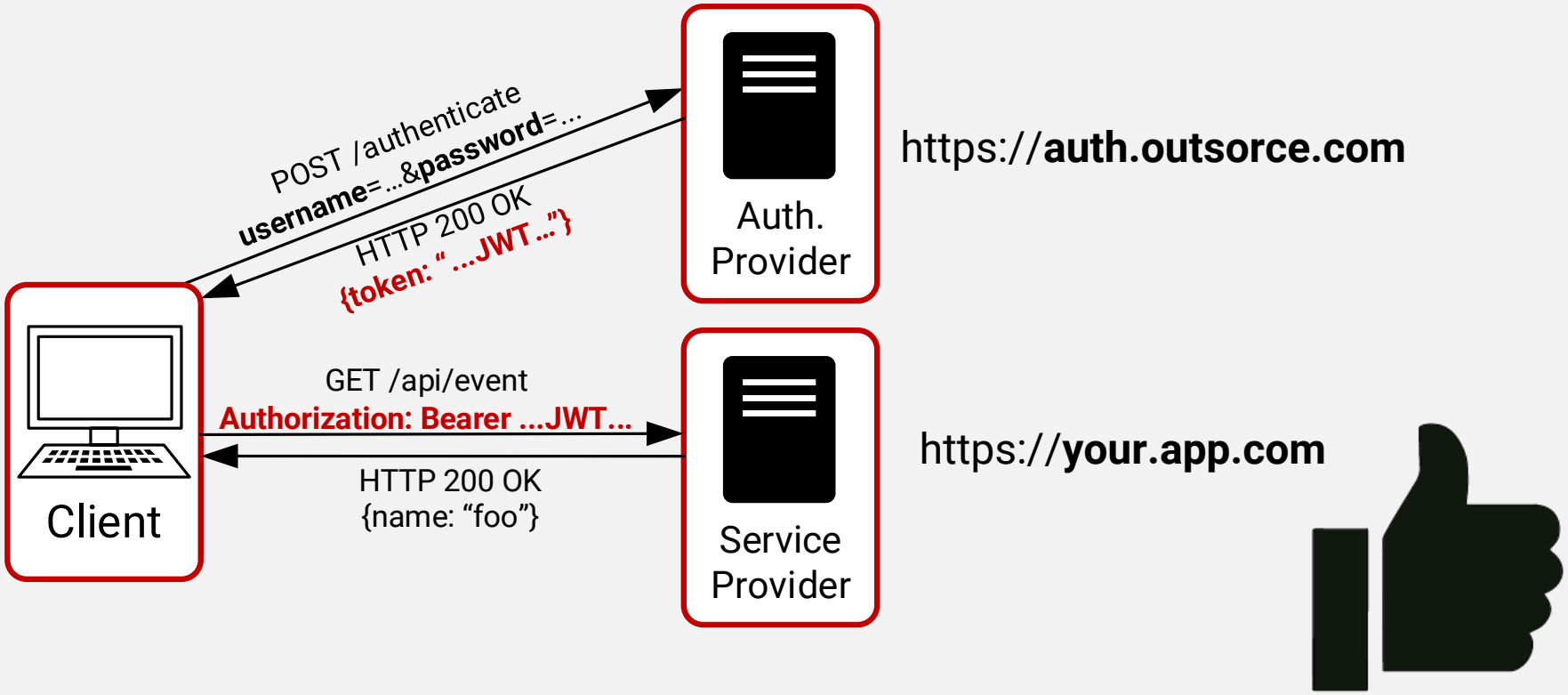


- No worries about
  - Session replication
  - Synchronization
  - Inter-node communication

JWT is **more flexible**

Flexible

- Content of JWT completely **configurable**
- Token creation and usage can be **separated**



Flexible

- “Separation of concerns”
  - **Benefits**
    - Outsource your auth. concerns
    - FB, Google, Amazon, Auth0, etc.
  - **Disadvantages**
    - Someone else can create valid tokens for your service
    - You need to trust the Auth. Provider
  - **Setup**
    - Register your App with Auth. Provider
      - Define claims needed in JWT
    - Agree on a shared key

JWT is **immune to CSRF attacks***

Immune to CSRF*

- “Session riding”
  - User tricked into making harmful requests
    - User already logged to that service
  - Relies on Session Cookie
- No Session Cookie
  - **No Problem**

Quick recap

JWT Pros

- Self-contained
- Stateless
- Scalable
- More flexible
  - Auth. outsourcing
- Immune to CSRF*

JWT Cons

Size

Size

- Smallest JWT **larger** than Session ID
  - Avg. Session ID + Cookie name = **45 Bytes**
- Header size constraints
  - HTTP defines none
  - But servers do
    - Tomcat – 8 KB
    - Jetty – 6 KB

No. of Claims	Size (Bytes)
0	75
1	125
3	150
5	225
10	350
15	550

Storing and Sending JWTs

Storage and transmission

- With Cookies everything is handled **automatically**
  - Server instructs Browser to store Session ID
  - Browser automatically sends stored Session ID
- With JWTs
  - **Storage**
    - Handled by client-side JS
    - Options
      - **Cookie** (not the same as Cookies used to store Session ID)
      - **LocalStorage**
  - **Transmission**
    - Handled by client-side JS
    - Options
      - Some frameworks have built-in support

More vulnerable to XSS

XSS problem

- Cross-site scripting
  - Like “SQL injection” – but for client-side JS
- **Rogue** JS
  - **It can**
    - Change the data displayed to the User
    - Make harmful requests to the Server
  - **It can’t**
    - Steal your Session ID (inside **HttpOnly** Cookie)
- But it can **steal** JWT
  - Handled by client-side JS
- **Solution**
  - Sanitize User inputs!

Signed, not encrypted*

Signed, not encrypted*

- Common use-case
  - Signed JWTs
  - **Readable** on client-side
- **“Solution”**
  - Use encrypted JWTs
- **Disadvantage**
  - Client can’t read User info from JWT
  - Client needs to contact Server to get that info
- **Best practice**
  - Don’t put anything too sensitive inside JWT
  - Always use HTTPS

Token revocation

Token revocation

- How to deny access to already logged User?
- **Friendly User**
  - User pressed Logout button
  - **Session**
    - Session is invalidated server-side
  - **JWT**
    - Token is deleted client-side

Token revocation

- How to deny access to already logged User?
- **Harmful User**
  - Hijacked Session ID or JWT
  - Someone is sending requests with *stolen* credentials
  - **Session**
    - Session is invalidated server-side
  - **JWT**
    - No explicit way of revoking already issued tokens
    - Only **trade-offs**

Token revocation

- “Revoking” JWT token
  - Keep a list of revoked JWTs
    - **Disadvantage**
      - Introducing state to server-side
  - Make JWTs short-lived
    - **Disadvantage**
      - User needs to login more frequently
    - Refresh tokens can be used to mitigate frequent login issue
  - **Nuclear option**
    - Change JWT **signing** key
    - **Disadvantage**
      - Invalidate **all** tokens
      - All users denied access

Quick recap

JWT Pros

- Self-contained
- Stateless
- Scalable
- More flexible
  - Auth. outsourcing
- Immune to CSRF*

JWT Cons

- Size
- Storage and transmission
- More vulnerable to XSS
- Signed vs. encrypted
- Token revocation

Quick recap

JWT Pros

- Self-contained
- Stateless
- Scalable
- More flexible
  - Auth. outsourcing
- Immune to CSRF*

JWT Cons

- Size
- Storage and transmission
- More vulnerable to XSS
- Signed vs. encrypted
- **Token revocation**

Thanks for your time!

Q & A

Spring Security + JWT example

Github

hcrnjak / spring-jwt-example