Security

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies** http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/react-typescript-academy-2022

Security Basic Concepts

- Authentication
- Authorization
- Data integrity
- Confidentiality
- Non-repudiation
- Auditing
- Quality of Service

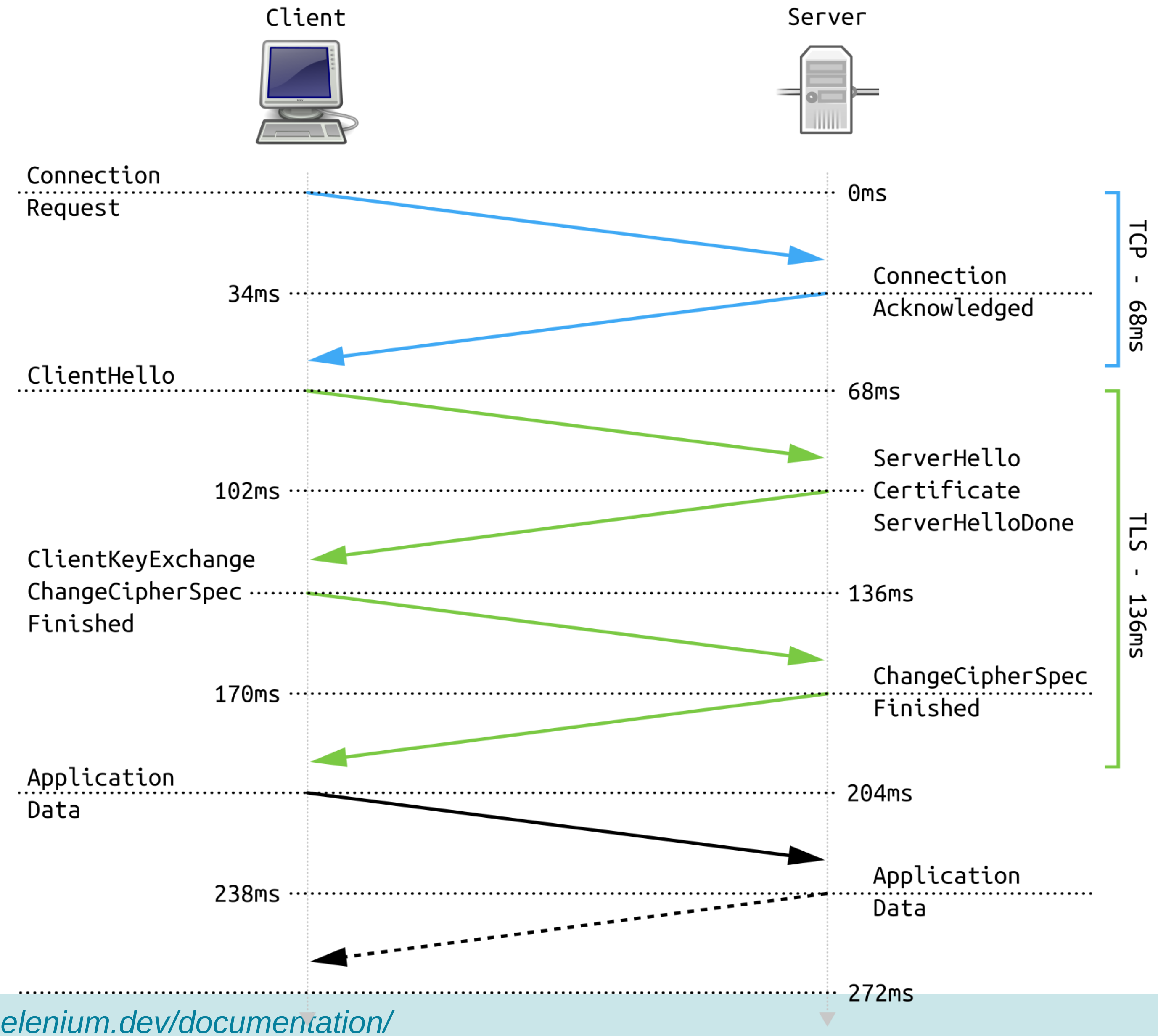
- Role
- Realm
- User
- Group
- Principal

Transport Layer Security (TLS)

- Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible.
- The TLS protocol aims primarily to provide cryptography, including privacy (confidentiality), integrity, and authenticity through the use of certificates, between two or more communicating computer applications. It runs in the application layer and is itself composed of two layers: the TLS record and the TLS handshake protocols.
- TLS is a proposed Internet Engineering Task Force (IETF) standard, first defined in 1999, and the current version is TLS 1.3, defined in August 2018.
- TLS is the successor of the now-deprecated Secure Sockets Layer (SSL).

https://www.selenium.dev/documentation/

Simplified TLS 1.2 Handshake



Client

Server

Connection Request

0ms

34ms

Connection Acknowledged

TCP - 68ms

ClientHello

68ms

ServerHello
Certificate
ServerHelloDone

102ms

TLS - 136ms

ClientKeyExchange
ChangeCipherSpec
Finished

136ms

170ms

ChangeCipherSpec
Finished

Application Data

204ms

238ms

Application Data

272ms

Using Self-Signed Certificates

- keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass changeit
- keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore springboot.p12 -validity 3650 -storepass changeit
- keytool -list -v -keystore springboot.jks
- keytool -list -v -keystore springboot.p12
- keytool -importkeystore -srckeystore springboot.jks -destkeystore springboot.p12 -deststoretype pkcs12
- keytool -import -alias springboot -file myCertificate.crt -keystore springboot.p12 -storepass password
- keytool -export -keystore springboot.p12 -alias springboot -file myCertificate.crt

Spring Boot SSL Configuration

```
server.ssl.key-store: classpath:springboot.p12
server.ssl.key-store-password: changeit
server.ssl.key-store-type: pkcs12
server.ssl.key-alias: springboot
server.ssl.key-password: changeit

server.port=8443
```

Authentication & Authorization

- Authentication is how we verify the identity of who is trying to access a particular resource. A common way to authenticate users is by requiring the user to enter a username and password.
- Authorization - Once authentication is performed we know the identity and can perform authorization.

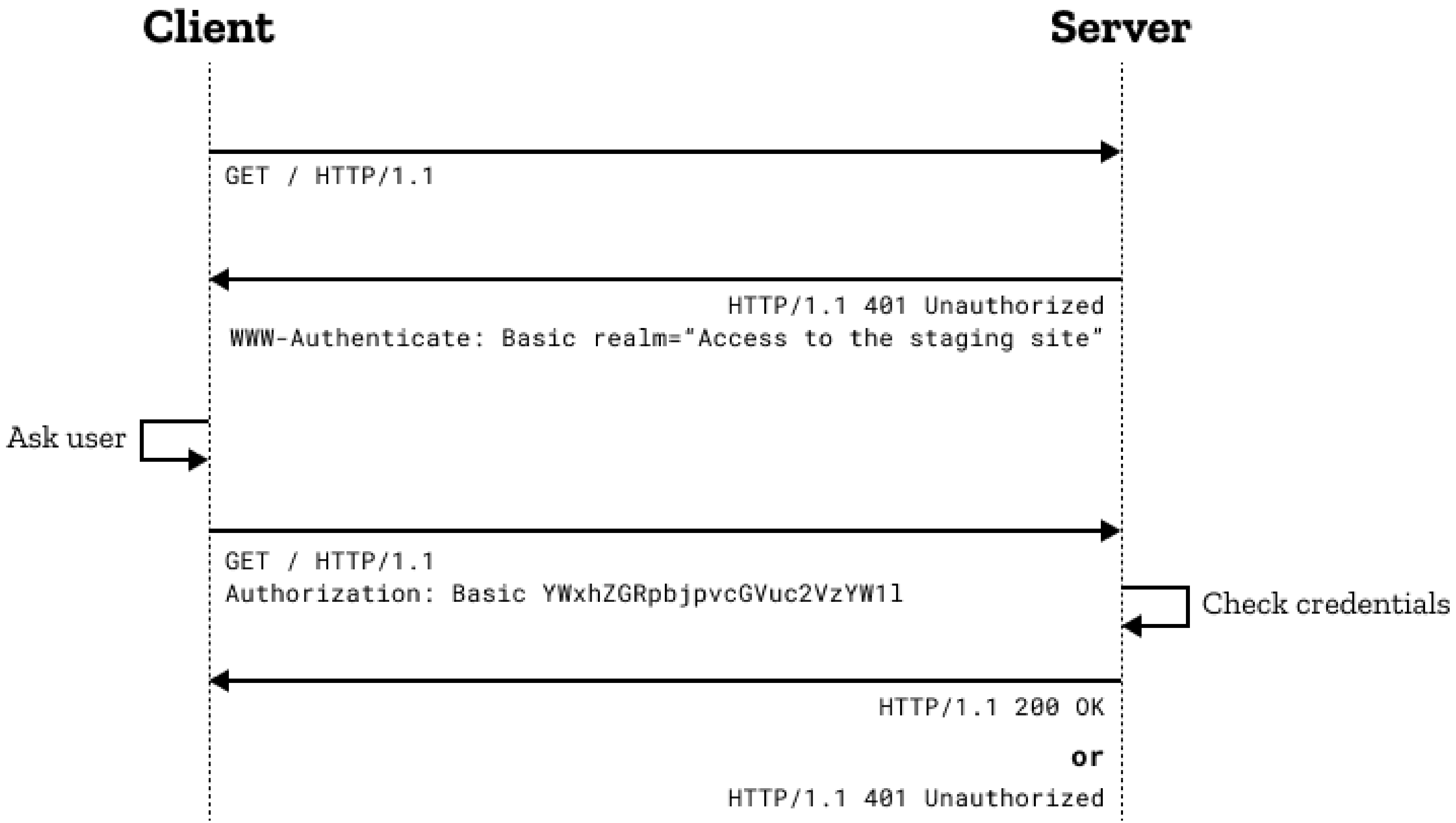
Source: https://docs.spring.io/spring-security

WWW-Authenticate – Authentication Schemes:

- **Basic -** See RFC 7617, base64-encoded credentials. More information below.
- **Bearer -** See RFC 6750, bearer tokens to access OAuth 2.0-protected resources
- **Digest -** See RFC 7616. Firefox 93 and later support the SHA-256 algorithm. Previous versions only support MD5 hashing (not recommended).
- **HOBA -** See RFC 7486, Section 3, **H**TTP **O**rigin-**B**ound **A**uthentication, digital-signature-based
- **Mutual -** See RFC 8120
- **Negotiate** / **NTLM -** See RFC4599
- **VAPID -** See RFC 8292
- **SCRAM -** See RFC 7804
- **AWS4-HMAC-SHA256 -** See AWS docs. This scheme is used for AWS3 server authentication.

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

WWW-Authenticate – Basic Authentication Scheme



Client

Server

```
GET / HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to the staging site"
```

Ask user

```
GET / HTTP/1.1
Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
```

Check credentials

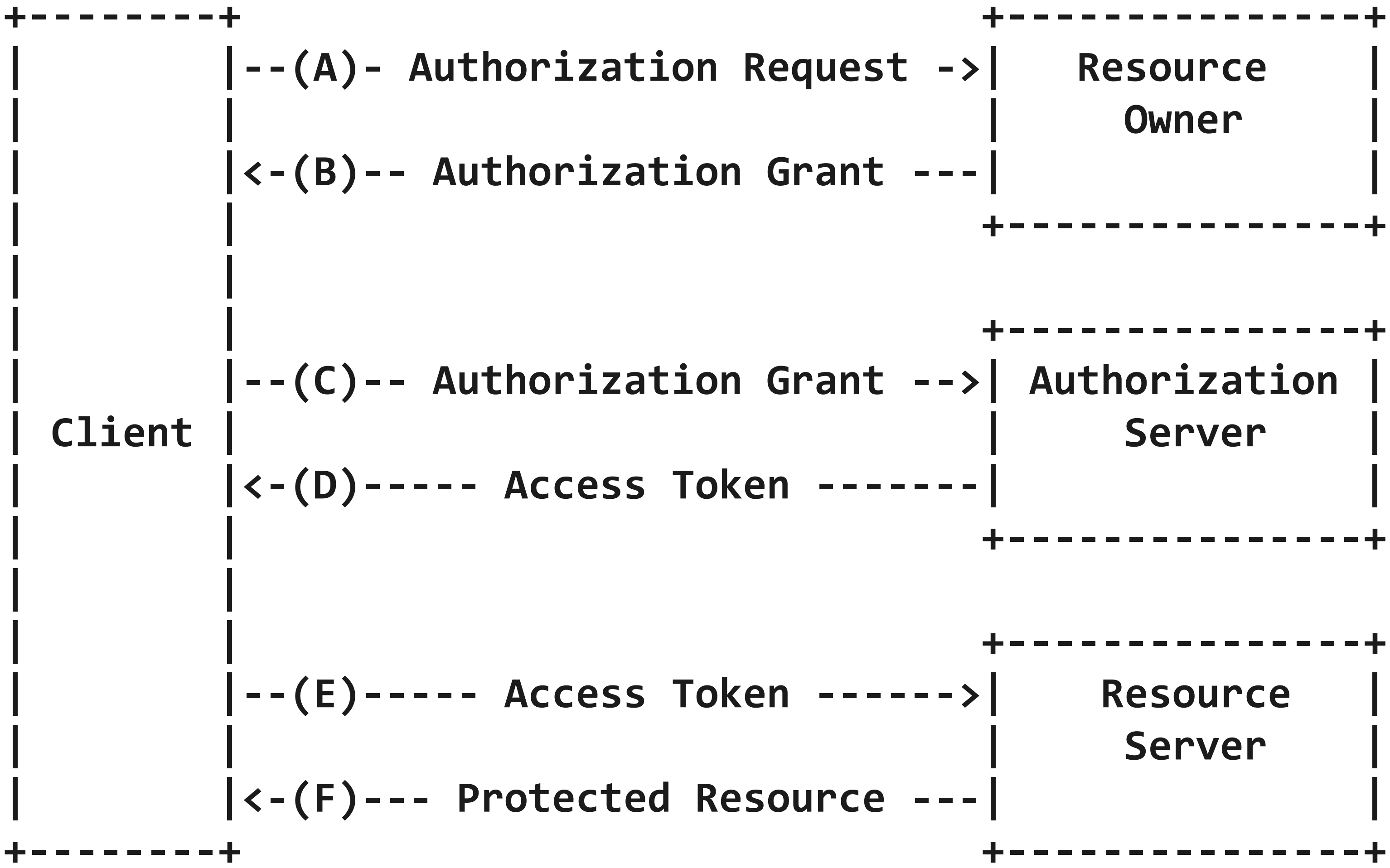
```
HTTP/1.1 200 OK
```

or

```
HTTP/1.1 401 Unauthorized
```

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication

The OAuth 2.0 Authorization Framework: Bearer Token Usage



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Figure 1: Abstract Protocol Flow

Source: https://datatracker.ietf.org/doc/html/rfc6750

WWW-Authenticate Response Header Field

- Example - the WWW-Authenticate Response Header Field:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example“
```

- Example - the– token expired:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                  error="invalid_token",
                  error_description="The access token expired"
```

Source: https://datatracker.ietf.org/doc/html/rfc6750

Authorization Request Header Field

- Example Bearer token authentication request:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1Jq
```

Source: https://docs.spring.io/spring-security

Json Web Token (JWT)

- Specific type of token that can be exchanged using Bearer authentication scheme:

```
HMAC_SHA256( secret, base64urlEncoding(header) + '.' + base64urlEncoding(payload) )
```

The three parts are encoded separately using Base64url Encoding RFC 4648, and concatenated using periods to produce the JWT:

```
const token = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' +
base64urlEncoding(signature)
```

- JWT example:

GET /resource HTTP/1.1

Host: server.example.com

Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjLiuyu5CSpyHI

Source: https://en.wikipedia.org/wiki/JSON_Web_Token

Json Web Token (JWT) Standard Claim Fields

Code	Name	Description
Standard claim fields		The internet drafts define the following standard fields ("claims") that can be used inside a JWT claim set.
iss	Issuer	Identifies principal that issued the JWT.
sub	Subject	Identifies the subject of the JWT.
aud	Audience	Identifies the recipients that the JWT is intended for. Each principal intended to process the JWT **must** identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the aud claim when this claim is present, then the JWT **must** be rejected.
exp	Expiration Time	Identifies the expiration time on and after which the JWT **must not** be accepted for processing. The value must be a NumericDate:[9] either an integer or decimal, representing seconds past 1970-01-01 00:00:00Z.
nbf	Not Before	Identifies the time on which the JWT will start to be accepted for processing. The value must be a NumericDate.
iat	Issued at	Identifies the time at which the JWT was issued. The value must be a NumericDate.
jti	JWT ID	Case-sensitive unique identifier of the token even among different issuers.

Source: https://en.wikipedia.org/wiki/JSON_Web_Token

Thank’s for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD