Autentifikácia I

Projekt 1

Motivácia

- Lenka má Internet Banking
- Banka jej musí garantovať, že:
  - iba ona má prístup k svojim peniažkom
    - Musí sa prihlásiť so svojím menom a heslom
  - nikto nebude odpočúvať otvorený kanál
    - Banka šifruje spojenie (HTTPS)
  - IB stránka je legitímna
    - Adresa web stránky (doména) je certifikovaná dôveryhodnou autoritou

Zabezpečenie

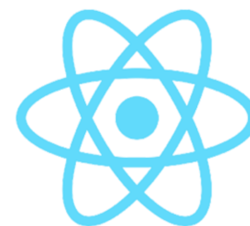
- Vitajte vo svete kryptografie – hešovanie, šifrovanie, podpisovanie
- TLS (SSL) + HTTPS – šifrovanie komunikačného kanála
- Encryption at rest – šifrovanie uložených dát v DB, v súboroch, ...
- **Autentifikácia** – kto môže používať appku
    - Registrácia, prihlasovanie používateľov
- Autorizácia – kto ako môže používať appku
    - Mám admin práva, obmedzené práva, read-only práva (na PAZ1c)
    - ACL – Access control list
    - **RBAC** – Role-based authorization control (o dva týždne)
    - ABAC – Attribute-based AC
    - PBAC – Policy-based AC
    - ReBAC – Relationship-based AC

Autentifikácia

- Autentifikuje sa človek?
  - HTTP Basic Auth,
  - Bearer/Token Auth (JWT),
  - **OIDC** (OpenID Connect),
  - SAML, Kerberos, LDAP
- Autentifikuje sa stroj?
  - napr. REST API voči DB
  - Bearer/Token Auth, TLS Client Auth

Full-stack aplikácia

- Pre našu appku Entrance máme
  - Web GUI – React
  - REST API – Spring Boot
- Príklad: Lucka chce zobraziť všetky osoby v Entrance
  - 1. Lucka chce v UI vykonať akciu
    - React pošle GET /persons požiadavku na Spring Boot
    - Spring Boot pošle zoznam osôb



1

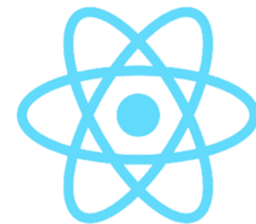
HTTP Basic Auth

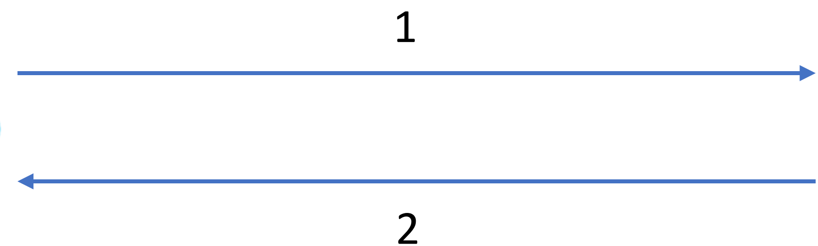
- Primitívna forma autentifikácie
- Každá HTTP požiadavka má hlavičku
  - `Authorization: Basic base64Encode(username:password)`
- Každé volanie stále posiela meno+heslo
- REST appka musí ukladať heslo
  - Napr. v DB ako osolený heš

HTTP Basic Auth

Príklad: Lucka chce zobraziť všetky osoby v Entrance

1. Prihlási sa s menom a heslom v Reacte
   - POST /login
   - SpringBoot povie OK
   - React si do browsera uloží správne meno a heslo
2. Lucka chce v UI vykonať akciu
   - React pošle GET /persons požiadavku na Spring Boot
   - Pribalí hlavičku Authorization: Basic s menom a heslom
   - Spring Boot pošle zoznam osôb





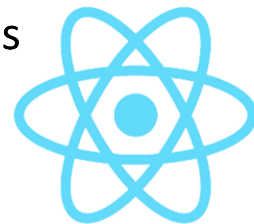
HTTP Bearer Auth

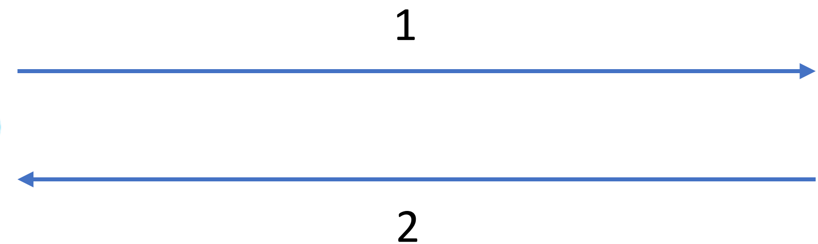
- Lepší ako Basic auth
- username+pass pošleme iba raz pri prihlásení
- Server vráti *zvláštny string* (`token`), ktorý potom posielame v ďalších HTTP požiadavkách ako hlavičku
    - `Authorization: Bearer “token“`

HTTP Bearer Auth

Príklad: Lucka chce zobraziť všetky osoby v Entrance

1. Prihlási sa s menom a heslom v Reacte
   - POST /login
   - SpringBoot vráti token
   - React si do browsera uloží token
2. Lucka chce v UI vykonať akciu
   - React pošle GET /persons požiadavku na Spring Boot
   - Pribalí hlavičku Authorization: Bearer s tokenom
   - Spring Boot pošle zoznam osôb





Basic Auth
Bearer Auth
Corporate needs you to find the differences between this picture and this picture.

Junior Dev
They're the same picture.

Token vs. heslo

- Token môže mať časovo obmedzenú platnosť – minúty až hodiny
  - Ukradnutie tokenu hackerom je trocha menej závažné
  - Ak užívateľ používa všade rovnaké heslo, tak token ho ochráni
- Spring Boot môže overiť platnosť tokenu **bez potreby ho uložiť**
  - Digitálne podpisovanie
- Ak skončí platnosť tokenu:
  - Užívateľ sa musí znova prihlásiť
  - Alebo implementujeme tzv. refresh tokeny pre automatické prihlásenie
  - Jedno prihlásenie tak bezpečne trvá dni až mesiace
- Viac REST API môžu token overovať bez potreby si ukladať užívateľa

JWT

Pokémon

JSON Web Token (JWT)

- JWT je formát tokenu, ktorý sa používa pri HTTP Bearer autentifikácii
- Obsahuje hlavičku a telo a podpis
  - Telo obsahuje atribúty (claims)
  - Máme dané claims a vlastné claims
  - Vlastné claims si dáme aké chceme
- „Algoritmus autentifikácie“
  - Lucka zadá meno+heslo v React rozhraní
  - Spring Boot si z DB vytiahne jej údaje a (hešované) heslo
  - Spring Boot skontroluje, či sedí heš hesla
  - Všetky relevantné informácie o Lucke + dobu platnosti dá do JWT
  - Pošle Reactu digitálne podpísané JWT
  - React je následne oprávnený po dobu platnosti JWT sa autentifikovať ním

```
{
 alg: "RS256",
 typ: "JWT",
 kid: "WpDCqYXnYTluKRnaehCRsR"
}.
{
 exp: 1710352337,
 iat: 1710352037,
 name: "Lucia Hrabavá",
 preferred_username: "lucka93",
 email: "lucia.hrabava@gmail.com"
 …
 customer_tier: "FREE_TIER"
 roles: ["STANDARD_CUSTOMER"]
}.
[SIGNATURE]
```

Výhody JWT oproti Basic Auth

- Pri Basic Auth musí React stále posielať svoje meno+heslo
  - Pri JWT stačí poslať Springu heslo iba raz (kým neuplynie platnosť tokenu)
- Taktiež React vidí o Lucke všetky relevantné informácie
  - Aké má oprávnenia (autorizácia)
    - Je normálny užívateľ, alebo admin?
    - Je platiaci klient, alebo má prístup iba k free časti aplikácie?
- Spring Boot keď dostane JWT od Lucky, tak už ju nemusí hľadať v DB
  - Efektívnosť, lebo SELECTy do DB sú „pomalé“
  - Všetko potrebné o Lucke si server vie vyčítať priamo z JWT
  - Vie akceptovať JWT aj od inej BE služby
    - Stačí, že zdieľajú overovací kľúč

Falzifikácia JWT?

- Spring Boot síce vyrobil JWT pri prvotnom prihlásení a poslal ho Reactu
- Ten JWT si už však nedrží
- Pri každej ďalšej požiadavke už však iba akceptuje JWT, čo pošle React
- JWT obsahuje všetky potrebné informácie o užívateľovi
  - Email, role, ...
- React si JWT typicky drží v
  - SessionStorage (zraniteľný voči XSS)
  - LocalStorage (zraniteľný voči XSS)
  - HTTP-only cookies (zraniteľný voči CSRF, nevieme však čítať z JS)

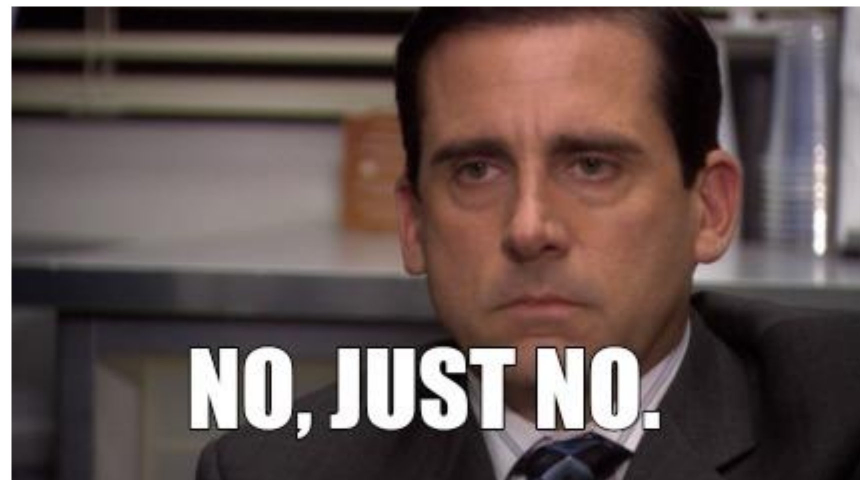
Falzifikácia JWT?

- Lucka má JWT vo svojom browsri
- Čo ak ju napadne si JWT „vylepšiť?
- Prepíše customer_tier z FREE_TIER na napr. PRO_TIER
- Prepíše roles na ADMIN, ...
- Alebo zmení email a username na niekoho iného
- Alebo predĺži platnosť (exp) tokenu
- ...

```
{
 alg: "RS256",
 typ: "JWT",
 kid: "WpDCqYXnYTluKRnaehCRsRHqzwN7FUlGXsN5u"
}.
{
 exp: 1710352337,
 iat: 1710352037,
 name: "Lucia Hrabavá",
 preferred_username: "lucka93",
 email: "lucia.hrabava@g[illegible]il.com"
 …
                   PRO
 customer_tier: "FREE_TIER"
 roles: ["STANDARD_CUSTOMER"]
}.
```

Ako (ne)zabrániť falzifikácií?

- Každého asi napadne si JWT pamätať v REST appke počas doby platnosti
  - NErobte to
  - Čo ak Lucka a Ferko sú obaja prihlásení?
    - Stále majú platné JWT.
    - Nezbedná Lucka sa môže pokúsiť upraviť svoj JWT aby vyzeral ako Ferkov



Správne zabránenie falzifikácie JWT

- JWT má okrem hlavičky a tela aj **podpis**
  - JWT = hlavicka + “.” + telo + “.” + **podpis**;
- Spring Boot pri vydaní JWT aplikuje tzv. kryptografický podpisový algoritmus (napr. HS, alebo **RS**)
  - HS – SHA hešovanie s „korením“ (privátny kľúč) spôsobom HMAC (hash message authentication code)
    - Soľ je pri hešovaní hesla iná pre každého používateľa
    - Korenie (pepper) je pri hešovaní tokenu stále rovnaké
  - RS – SHA hešovanie + RSA „šifrovanie“ s obrátenou úlohou verejného a privátneho kľúča

```
var podpis = podpíš(base64Enc(jwtHlavicka + jwtTelo), privatnyKluc);
return jwtHlavicka + “.” + jwtTelo + “.” + podpis;
```

Rozpoznanie falošného JWT

- Keď sa React autentifikuje cez JWT, tak:
  - Spring Boot zoberie z požiadavky JWT hlavičku a telo
  - Zahešuje ich, potom "rozšifruje" podpis a porovná nový svoj heš s pôvodným v rozšifrovanom podpise
- Ak Lucka zmenila hlavičku alebo telo, tak novýHeš != starýHeš
  - Spring Boot prehlási JWT za plagiát a odmietne ho
- „Kvalitný“ falzifikát musí mať primerane zmenený aj podpis
  - Toto je prakticky nemožné bez znalosti privátneho kľúča servera
  - Bez priv. kľ. by Lucka musela backtrackovať vš. možné podpisy
    - Výpočet by trval milióny rokov
  - Alebo si zoženie nVidia GeForce QTX 9090 Ti Super, r.v. 2050 a zvládne to za 2 minúty

Sumarizácia HTTP Bearer Auth a JWT

- Autentifikácia užívateľa (takmer) bez hesla
- JWT obsahuje mnoho dát o užívateľovi
- Server si nesmie JWT ukladať
- Server dôveruje prijatému JWT
- ... a dôveru si preveruje kryptografickým podpisovaním
- JWT majú obmedzenú platnosť (zvyčajne) na minúty-hodiny
  - Ak chceme aby užívateľ mohol ostať prihlásený bez hesla dlho (hodiny-mesiace), tak existujú dodatočné obnovovacie (refresh) tokeny

Delegovanie autentifikácie

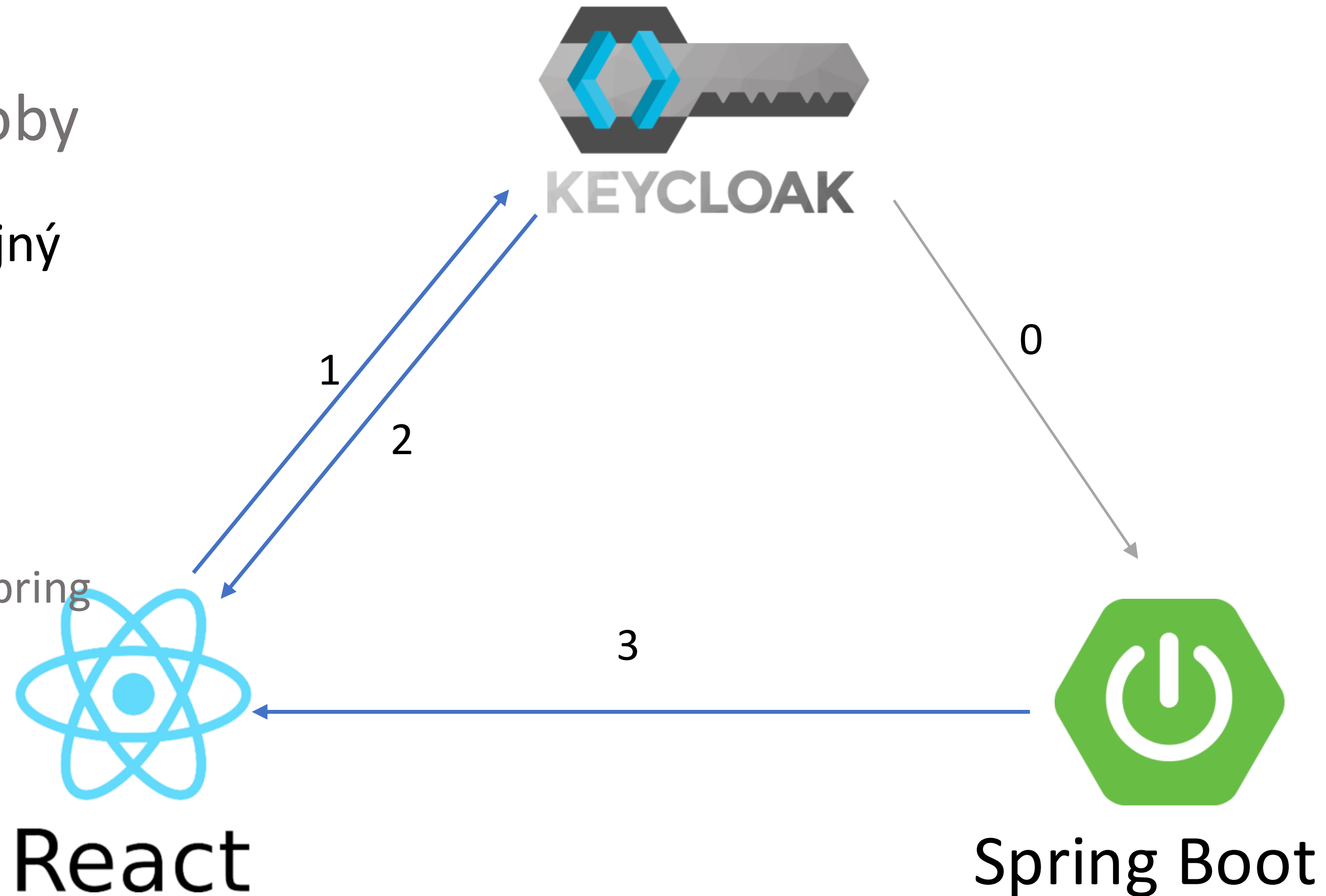
Áutsórsujme sekjúritu

OpenID Connect (OIDC)

- Rozšírenie myšlienky HTTP Bearer + JWT
- Tiež (nepresne) známe ako OAuth 2
  - Technicky je OIDC nadmnožinou OAuth 2
  - OAuth2 rieši autorizáciu na iné webové služby
    - Napr. integrujeme do našej appky Outlook/Google kalendár
  - OIDC je novší a robustnejší protokol
  - Single sign-on (SSO) – Prihlásim sa do 1 aplikácie => prihlásil som sa do všetkých
- Rozdelíme naše Spring Boot REST API na dve samostatné časti
  - 1. časť bude logika aplikáce
  - 2. časť bude riešiť iba autentifikáciu/autorizáciu

Full-stack aplikácia s OIDC

- Máme
  - Web GUI – React
  - REST API – Spring Boot
  - OIDC poskytovateľ – Keycloak
- Príklad: Lucka chce zobraziť všetky osoby v Entrance
  - 0. Spring Boot si pri spustení vypýta verejný kľúč od Keycloaku (JWKS)
  - 1. React Lucku presmeruje na Keycloak
    - Tá sa tam prihlási cez meno+heslo
  - 2. Keycloak vydá JWT a pošle ho Reactu
  - 3. Lucka chce v UI vykonať akciu
    - React pošle GET /persons požiadavku na Spring Boot
    - No teraz do hlavičky pribalí JWT
    - Spring Boot overí platnosť JWT, ... a pošle zoznam osôb



HTTP Bearer

- Lucka si otvorí naše GUI web/mobil/desktop rozhranie
- Zadá login+heslo
- GUI pošle do REST appky login+heslo
- REST appka vráti JWT
- GUI následne pri komunikácii s REST appkou používa JWT token
  - Pomocou svojho validačného kľúča následne overuje pravosť všetkých JWT

OpenID Connect

- Lucka si otvorí naše GUI web/mobil/desktop rozhranie
- GUI ju presmeruje na OIDC appku
- Lucka zadá login+heslo
- OIDC appka vráti JWT do GUI
- GUI následne pri komunikácii s REST appkou používa JWT token
  - REST appka si z OIDC appky stiahne verejný (validačný) kľúč
  - Pomocou toho kľúča následne overuje pravosť všetkých JWT

Výhody OpenID Connect

- Naša aplikácia si nemusí pamätať heslá užívateľov
- Vieme si nainštalovať hotový OIDC server, kde máme:
  - Obnovovanie zabudnutých hesiel cez emaily
  - Viac faktorové overovanie (email, SMS, TOTP appky v mobiloch)
- Alebo vieme použiť OAuth 2/OIDC služby tretích strán
  - Prihlasovanie do našej appky cez Google, Twitter, Office 365, ...

Keycloak

- Je open-source OIDC server
- Napísaný v Jave
- Potrebuje nejakú SQL databázu
- Široko používaný
- Veľa možnosti manažmentu používateľov
  - Notifikácie, resetovanie heslá, viac-faktorové overovanie (MFA)
- Integráca s OAuth2 od Google, Microsoft, ...

Inštalácia cez Docker Compose

```yaml
services:
  mysql:
    image: 'mysql:latest'
    environment:
      - 'MYSQL_DATABASE=keycloak'
      - 'MYSQL_PASSWORD=secret'
      - 'MYSQL_ROOT_PASSWORD=verysecret'
      - 'MYSQL_USER=kcuser'
    ports:
      - '3306:3306'

  keycloak:
    image: keycloak/keycloak:26.5.6-0
    environment:
      - KEYCLOAK_ADMIN=admin
      - KEYCLOAK_ADMIN_PASSWORD=admin
      - KC_DB=mysql
      - KC_DB_URL_HOST=mysql
      - KC_DV_URL_DATABASE=keycloak
      - KC_DB_USERNAME=kcuser
      - KC_DB_PASSWORD=secret
      - KC_HTTP_PORT=9080
    command: ["start-dev"]
    ports:
      - '9080:9080'
```

Riešime problémy

- Cez Docker Compose spúšťame dve appky
  - MySQL a Keycloak
- Docker všetko spúšta naraz
- Potrebujeme zabezpečiť, aby sa Keycloak spustil až po MySQL

```
services:
 mysql:
 …

 keycloak:
  …
  depends_on:
   - mysql
```

Riešime problémy II

- Stále Keycloak padne (asi)
- Docker spustí Keycloak až po MySQL
  - Ale MySQL nie je pripravená hneď
  - Potrebuje zopár sekúnd na „kompletné naštartovanie“
- Potrebujeme Dockeru povedať, aby sa Keycloak spustil až keď je MySQL pripravená

```
services:
  mysql:
    ...
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
      interval: 10s # default is 30s
      timeout: 5s   # default is 30s
      retries: 5    # default is 3

  keycloak:
    ...
    depends_on:
      mysql:
        condition: service_healthy
```

Demo

- Prihlásenie ako admin
- Vytvorenie nového používateľa
  - Nastavme dočasné heslo
  - Nastavme viacfaktorové prihlásenie cez TOTP (Time-based One-Time Password)
- Prihlásme sa ako nový používateľ
  - Musíme si zmeniť heslo a nastaviť TOTP v mobile

Ďakujem za pozornosť

Pán Sulu, kurz domov. Warp 5.