Autentifikácia

Projekt 1

Prednáška 4

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 – ako môže používať appku
  - Mám admin práva, obmedzené práva, read-only práva
  - RBAC – Role-based authorization control

Autentifikácia

- Autentifikuje sa človek?
  - HTTP Basic Auth
  - Bearer/Token (JWT),
  - **OIDC** (OpenID Connect)
  - SAML, Kerberos, LDAP
- Autentifikuje sa stroj?
  - Bearer/Token, TLS client auth

HTTP Basic Auth

- Primitívna forma autentifikácie
- Každá HTTP požiadavka má hlavičku
  - `Authorization: Basic base64Encode(username:password)`
- Stále sa posiela meno+heslo
- Backend aplikácia musí ukladať heslo
  - Napr. v DB ako osolený heš

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“`
- Token môže mať časovo obmedzenú platnosť – minúty až hodiny
- Server môže overiť platnosť tokenu bez potreby ho uložiť
- Ak skončí platnosť tokenu:
  - Užívateľ sa musí znova prihlásiť
  - Alebo implementujeme tzv. refresh tokeny pre automatické prihlásenie
    - Jedno prihlásenie trvá dni až mesiace


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
  - Server si z DB vytiahne jej údaje a (hešované) heslo
  - Server skontroluje, či sedí heš hesla
  - Všetky relevantné informácie o Lucke + dobu platnosti dá do JWT
  - Pošle Lucke JWT
  - Lucka je následne oprávnená 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 Lucka stále posielať svoje meno+heslo
  - Pri JWT stačí poslať serveru heslo iba raz (kým neuplynie platnosť)
- Taktiež Lucka vidí všetky relevantné informácie
  - Aké má oprávnenia (autorizácia)
    - Je normálny užívateľ, alebo admin?
    - Je platiaci klient, alebo má prístup iba k free časti aplikácie?
- Server 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

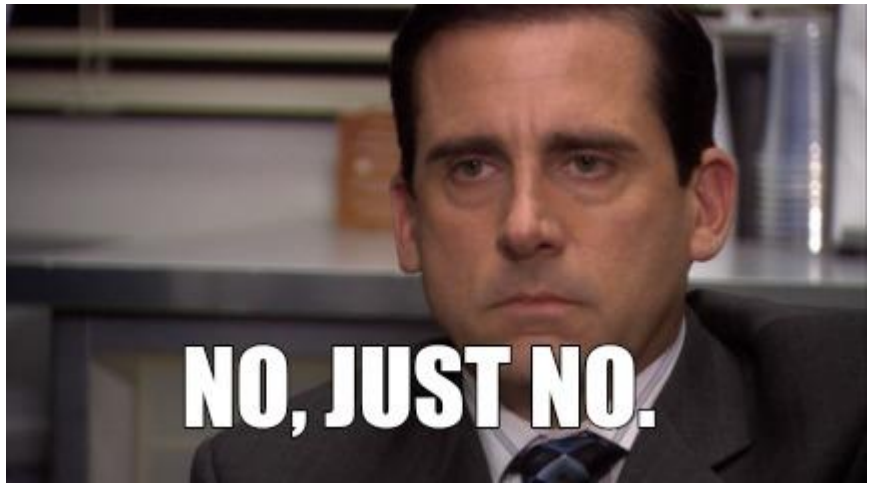
Falzifikácia JWT?

- Lucka má JWT
- 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@gmail.com"
 ...
 customer_tier: "FREE_TIER"
 roles: ["STANDARD_CUSTOMER"]
}.
```

Ako (ne)zabrániť falzifikácií?

- Každého asi napadne si JWT pamätať na serveri 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;
- Server pri vydaní JWT aplikuje tzv. kryptografický podpisový algoritmus (napr. RS, alebo HS)
  - HS – SHA hešovanie s konštantnou „soľou“ (privátny kľúč) spôsobom HMAC (hash message authentication code)
  - 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 po čase dostane od Lucky údajne „to isté“ JWT, tak:
  - Zoberie z jej JWT hlavičku a telo
  - Znova ich podpíše a porovná nový podpis s tým, čo už je v Luckinom JWT
- Ak Lucka zmenila hlavičku alebo telo, tak novýPodpis != jwt.podpis
  - Server prehlási JWT za falošný 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

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
  - 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 serverovú aplikáciu (backend) na dve samostatné časti
  - 1. časť bude logika aplikáce
  - 2. časť bude riešiť iba autentifikáciu/autorizáciu

HTTP Bearer

- Lucka sa najprv prihlási na server
- Dostane z neho JWT
- Následne pri komunikácii so serverom používa JWT token

OpenID Connect

- Lucka sa prihlási na *autentifikačný* server
- Dostane z neho JWT
- Následne pri komunikácii s našou aplikáciou používa JWT token
- Naša aplikácia si z autentifikačného servera 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

```
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:24.0.1-0
    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-bases 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.