

Sigurna implementacija autentifikacije



ОБАВЕШТЕЊЕ ЗА СТУДЕНТЕ

- Настава на предмету Развој безбедног софтвера подразумева изучавање различитих механизма којима се нарушава информациона безбедност и врше напади на интернет апликације и софтверске системе.
- Студенти на предмету Развој безбедног софтвера могу ове методе за потребе изучавања да користе искључиво у оквиру затвореног лабораторијског окружења које је обезбеђено за наставу на предмету Развој безбедног софтвера.
- Студенти не могу да подразумевају да су на било који начин охрабрени од стране наставника или да им се препоручује да користе ове методе који се изучавају према другим апликацијама Електротехничког факултета или апликацијама било ког трећег правног или физичког лица.
- Свака евентуална активност коју би предузео неки студент коришћењем ових метода и механизма према апликацијама које нису у оквиру лабораторије на предмету искључива је одговорност студента.

Uvod

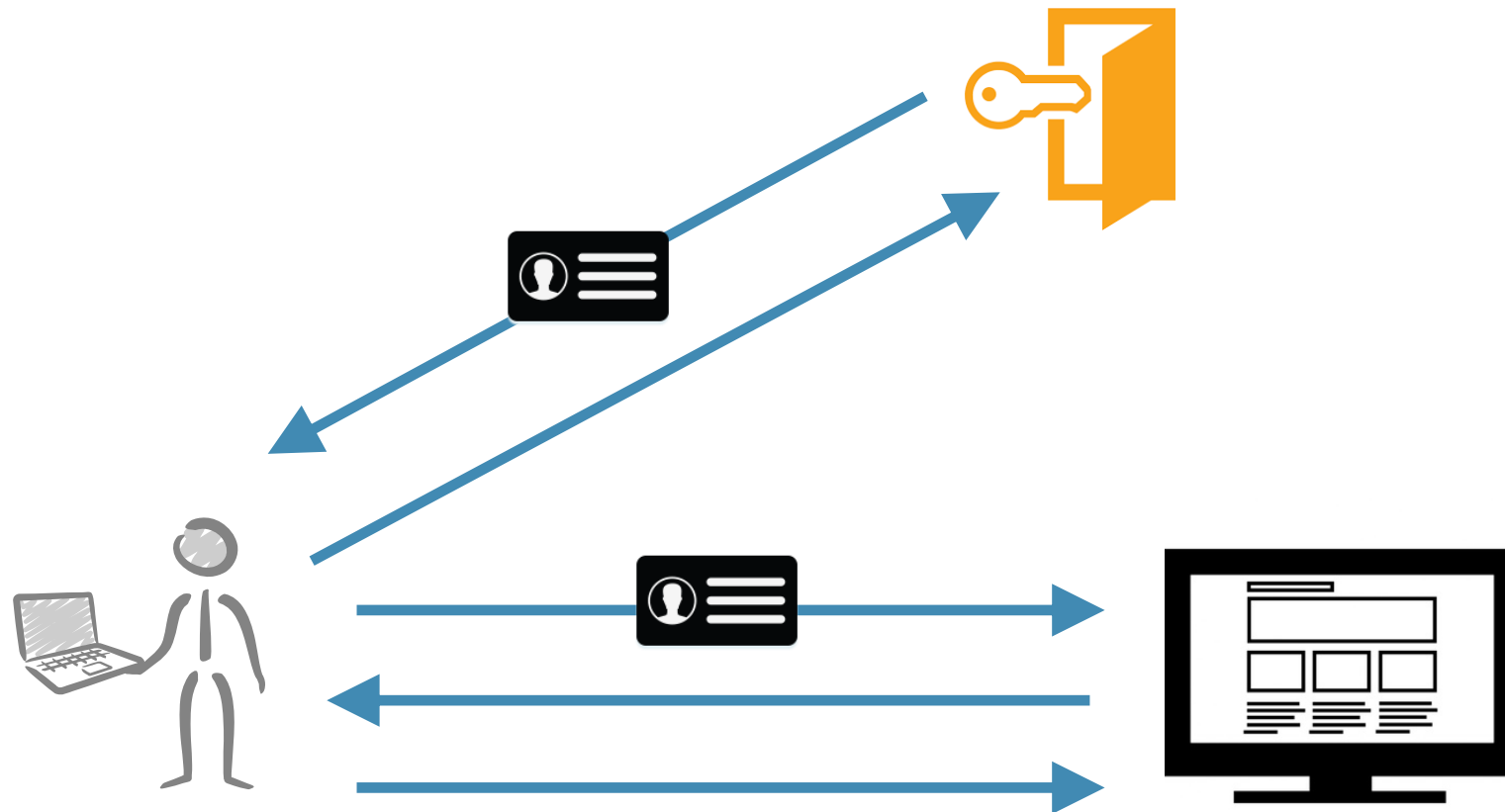
- Autentifikacija je sigurnosna usluga koja utvrđuje korisnikov identitet.
 - Odgovor na pitanje: Ko je korisnik?
- Nasuport autorizaciji koja određuje koje podatke korisnik može da vidi i koje akcije korisnik može da izvrši.
 - Odgovor na pitanje: Šta korisnik može da uradi?

Vrste autentifikacije – Basic (RFC 7235)

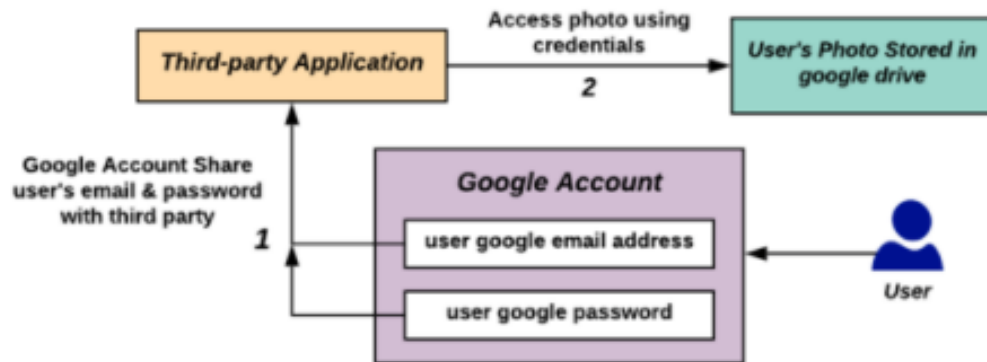
- Basic authentication: Username + password su ubačeni kao deo HTTP Authorization headera sa svakim HTTP zahtevom
 - Header: Authorization
 - Header vrednost: Basic encodeToBase64(<username>:<password>)



Vrste autentifikacije – OAuth 2.0 (RFC 6749)



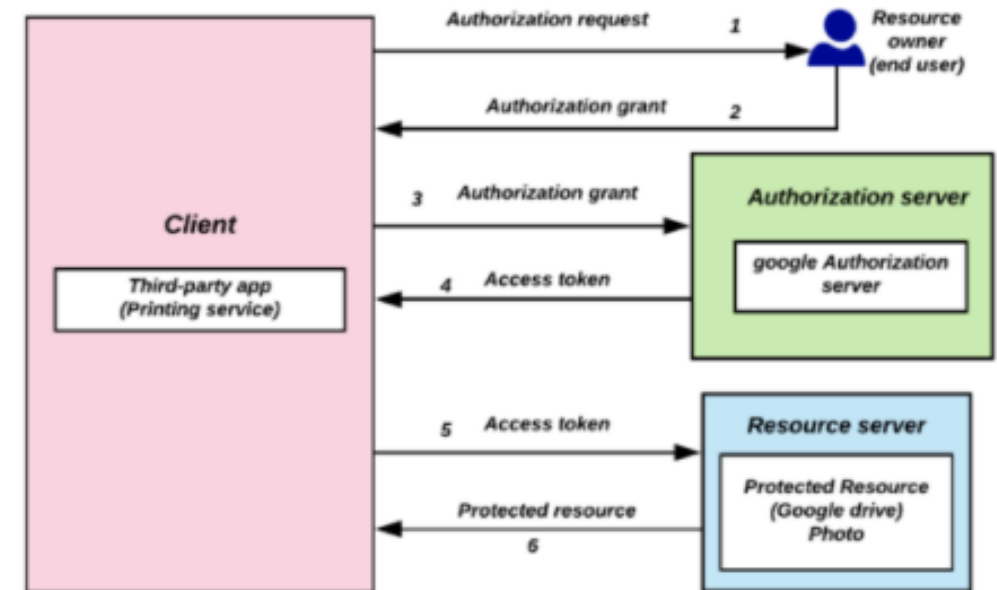
World without OAuth



A third party application in order to access user's photo stored in a google drive, google needs to share user's email address and password with the third party.

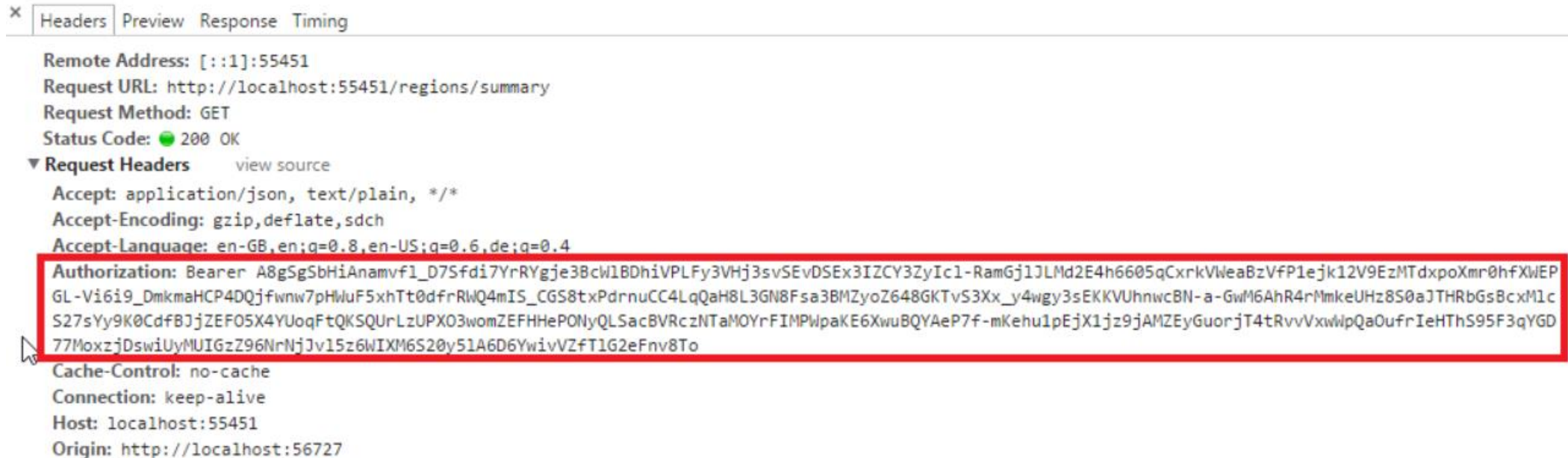
✗ Nobody want this Right ?

Abstract Flow




Vrste autentifikacije – OAuth 2.0 (RFC 6749)

- Bearer token je ubačen kao deo Authorization headera sa svakim HTTP zahtevom



```
Headers Preview Response Timing
Remote Address: [::1]:55451
Request URL: http://localhost:55451/regions/summary
Request Method: GET
Status Code: 200 OK
▼ Request Headers view source
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en;q=0.8,en-US;q=0.6,de;q=0.4
Authorization: Bearer A8gSgSbHiAnamvf1_D7Sfdi7YrRYgje3BcW1BDhiVPLFy3VHj3svSEvDSEx3IZCY3ZyIc1-RamGj1JLMd2E4h6605qCxrkvWeaBzVfP1ejk12V9EzMTdxpoXmr0hfXWEP
GL-Vi6i9_DmkmaHCP4DQjfwnw7pHwUf5xhTt0dfrRWQ4mIS_CGS8txPdrnuCC4LqQaH8L3GN8Fsa3BMZyoZ648GKTvS3Xx_y4wgy3sEKKVUhnwcBN-a-GwM6AhR4rMmkeUH8S0aJTHRbGsBcxM1c
S27sYy9K0CdFBJjZEF05X4YUoqFtQKSQURLzUPX03womZEFHHePOnyQLSacBVRczNTaMOYrFIMPWpaKE6XwuBQYAeP7f-mKehu1pEjX1jz9jAMZEyGuorjT4tRvvVxwWpQa0ufrIeHTHS95F3qYGD
77MoxzjDswiUyMUIGzZ96NrNjJv15z6WIXM6S20y51A6D6YwivVZfT1G2eFNV8To
Cache-Control: no-cache
Connection: keep-alive
Host: localhost:55451
Origin: http://localhost:56727
```

Vrste autentifikacije – OAuth 2.0 (RFC 6749)

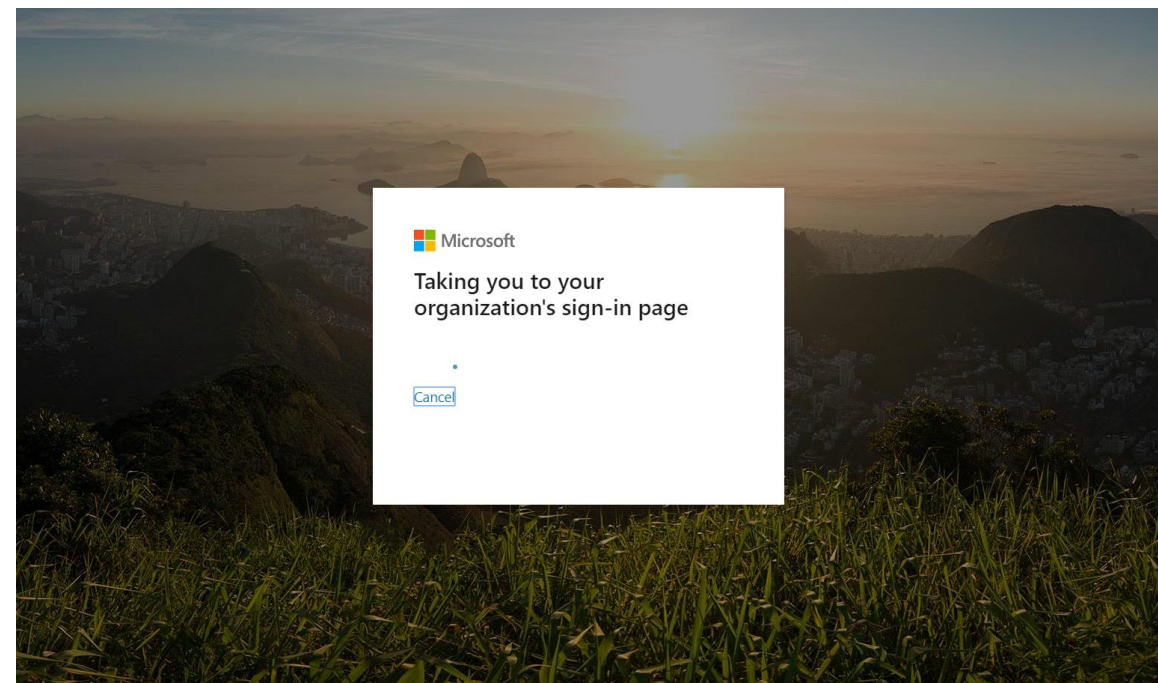


Elektrotehnički Fakultet
Univerzitet u Beogradu

Sign in with your organizational account

[Sign in](#)

© 2013 Microsoft



Poređenje

Basic authentication	OAuth 2.0
Dodavanje lozinke u header	Dodavanje tokena u header
Ukraden header otkriva username i lozinku	Ukraden header otkriva token ✓
Lak za implementaciju ✓	Težak za implementaciju
Nije ekstenzibilan i konfigurabilan	Ekstenzibilan i konfigurabilan (primer OpenId) ✓
Kredencijali su validni uvek	Token je validan ograničeno (< 1h) ✓
Mora se koristiti HTTPS	Mora se koristiti HTTPS



Sigurno skladištenje lozinke

Da li je dovoljno enkriptovati lozinku ili bazu podataka?

Da li je dovoljno enkriptovati lozinku ili bazu podataka?

Administrator može da je dekriptuje sa master ključem!

➤ Sigurnosni princip: neporecivost (Non-repudiation)

Ukoliko napadač ukrade bazu i master ključ

➤ Može imati pristup celom sistemu a ne samo bazi sa lozinkama

Heš (*hash*) funkcija

- Računanje numeričke vrednosti na osnovu *plain text*-a. Za svaki set podataka, ova vrednost je unikatna.
- Koristi se za utvrđivanje **integriteta** podataka, tj. vrsta checksum vrednosti

Osobine sigurnosnih hash funkcija

- **Determinizam** – za datu ulaznu vrednost mora da se generiše uvek ista hash vrednost
- **Definisan opseg** – izlazna vrednost mora imati istu fiksnu dužinu kad se koristi u istom kontekstu, npr. 160-bit, 512-bit...
- **Uniformnost** – dobra hash funkcija ima dobru uniformnu raspodelu mapiranja ulaznih vrednosti u izlazne vrednosti, tj. pojava izlaznih vrednosti mora biti slične verovatnoće
- **Niska verovatnoća kolizije** – nije lako ponoviti izlaznu vrednost za dve drugačije ulazne vrednosti
- **Jednosmerna** – nije realistično da se rekonstruiše ulazna vrednost na osnovu izlazne vrednosti, tj. nije inverzna – asimetrična

Primer (SHA-2 512bita)

„**P**rimera vrednosti“

BD8DECDD0CE230396753A86F947001259F2FFAEB76A655236DFB
11E8BADB9CC375353762596CB52E1DDC83049CFECB1E93C72D57
3A8CFD422B5CFC36B5EEDA6B

„**p**rimera vrednosti“

CF9C5A88A348B4CEC95F304E0168F05B4E384E28C68CC482AB73
44B9AFFCDA91A6E555E2838C59386C96A4A3A6AF458985F2E64A
1F0F70CDA04C5C5C446BB14B

Poznate heš funkcije / algoritmi

- MD5 (128-bit)
- SHA-1 (160-bit)
- SHA-2 (224, 256, 384, ili 512 bita)
- SHA-3 (arbitrarna dužina)

Zašto hešovati lozinku?

Zašto ne enkriptovati lozinku nego hešovati?

Zašto hešovati lozinku?

Zašto ne enkriptovati lozinku nego hešovati?

- Enkriptovana lozinka može da se dekriptuje i pročitati ukoliko neko ima master ključ
- Hešovana lozinka ne može nikada da se pročitati / jednosmernost heš funkcije
 - Nemogućnost da se krađom baze lako dođe do lozinke
- Dokaz da samo korisnik zna svoju lozinku
 - Neporecivost akcije, nemogućnost krađe identiteta, legitimitet akcije

Hešovanje nije dovoljno

Rainbow Table napad korišćenjem velike baze unapred izračunatih hash vrednosti

Napadač pronalazi izlaznu vrednost u svojoj bazi i koristi odgovarajuću ulaznu vrednost

Moguće je da ulazna vrednost nije ista kao korisnikova lozinka ali ima istu hash vrednost

Šta je rešenje?

Salt

„Soljenje“ lozinke označava ojačanje heša lozinke tako što se salt vrednost konkatenuira na lozinku pre izvršenja hash funkcije

Salt je random generisana vrednost fiksne dužine i **jedinstvena je za korisnika**

Hash(salt + password)

Salt primer

Username	Password	Salt value	String to be hashed	Hashed value = SHA256 (Salt value + Password)
user1	password123	E1F53135E559C253	E1F53135E559C253password123	72AE25495A7981C40622 D49F9A52E4F1565C90F0 48F59027BD9C8C8900D 5C3D8
user2	password123	84B03D034B409D4E	84B03D034B409D4Epassword123	B4B6603ABC670967E99 C7E7F1389E40CD16E78A D38EB1468EC2AA1E62B 8BED3A

Password-Based Key Derivation Function 2 (PBKDF2)

- Standard koji olakšava kreiranje heša lozinke sa saltom
- Zahteva kao ulaz
 - Salt korisnika
 - Algoritam za heširanje koji će se koristiti
 - Broj iteracija heširanja (koliko puta se heš funkcija primenjuje)
 - Broj bajtova za izlaz heš funkcije
 - Lozinku koja se hešuje (koju želimo da validiramo)

Elementi autentifikacije u kodu

Demonstracija

Elementi autentifikacije u kodu

- Na stranici **login.html** postavili smo da se autentifikacija vrši **POST** zahtevom na **/perform-login**

```
<form class="d-flex flex-column justify-content-around" style="height: 200px"
  method="POST"
  action="/perform-login">
  <input type="text" class="form-control" name="username" placeholder="Username" required>
  <input type="text" class="form-control" name="password" placeholder="Password" required>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```


Elementi autentifikacije u kodu

- U klasi **SecurityConfig** obaveštavamo framework da zahtev za prijavu treba da očekuje na putanji **/perform-login**
- U istoj klasi registrujemo **DatabaseAuthProvider** klasu koju smo mi implementirali zaduženu da proveriti da li su kredencijali ispravni

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/login").permitAll()
        .antMatchers("/**").authenticated()
        .and()
        .formLogin()
        .loginPage("/login")
        .loginProcessingUrl("/perform-login")
        .defaultSuccessUrl("/cars")
        .failureUrl("/login?error")
        .and()
        .logout()
        .logoutSuccessUrl("/login")
        .invalidateHttpSession(true)
        .deleteCookies("JSESSIONID");

    // We need this one in order to access h2-console
    http.headers().frameOptions().sameOrigin();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) {
    auth.authenticationProvider(databaseAuthProvider);
}
```

Elementi autentifikacije u kodu

- U metodi authenticate DatabaseAuthenticationProvider-a, koristeći pomoćnu metodu validCredentials proveravamo da li su kredencijali ispravni

```
public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    String username = authentication.getName();
    String password = authentication.getCredentials().toString();

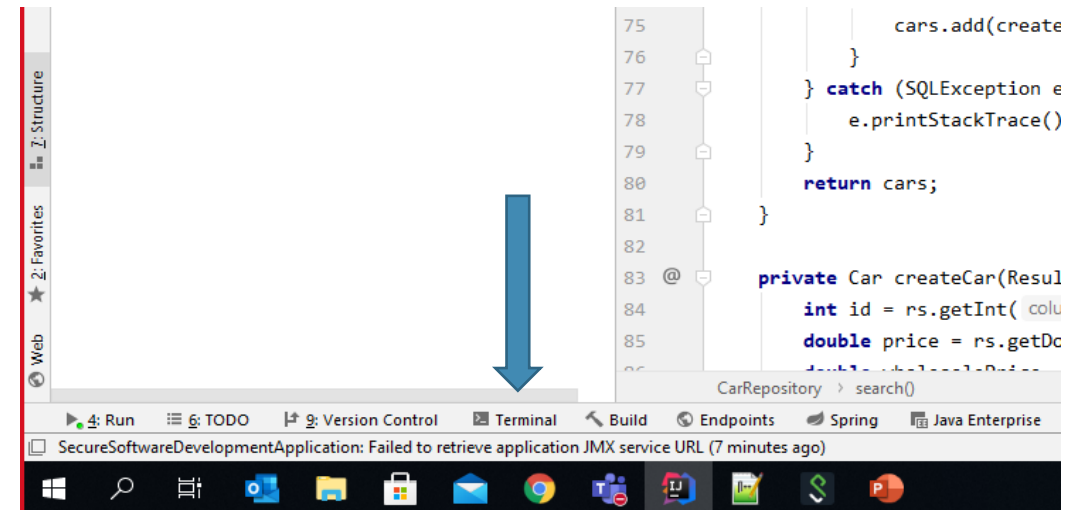
    boolean success = validCredentials(username, password);
    if (success) {
        User user = userRepository.findByUsername(username);
        List<GrantedAuthority> grantedAuthorities = getGrantedAuthorities(user);
        return new UsernamePasswordAuthenticationToken(user, password, grantedAuthorities);
    }

    throw new BadCredentialsException(String.format(PASSWORD_WRONG_MESSAGE, username, password));
}

private boolean validCredentials(String username, String password) {
    return userRepository.validCredentials(username, password);
}
```

Priprema za samostalni rad

1. Otvorite IntelliJ Idea
2. U terminalu izvršite **git checkout sigurna-implementacija-autentifikacije**
3. Zatim **git reset --hard**
4. Pokrenite aplikaciju
5. Prijavite se
 - username: **bruce**
 - password: **wayne**

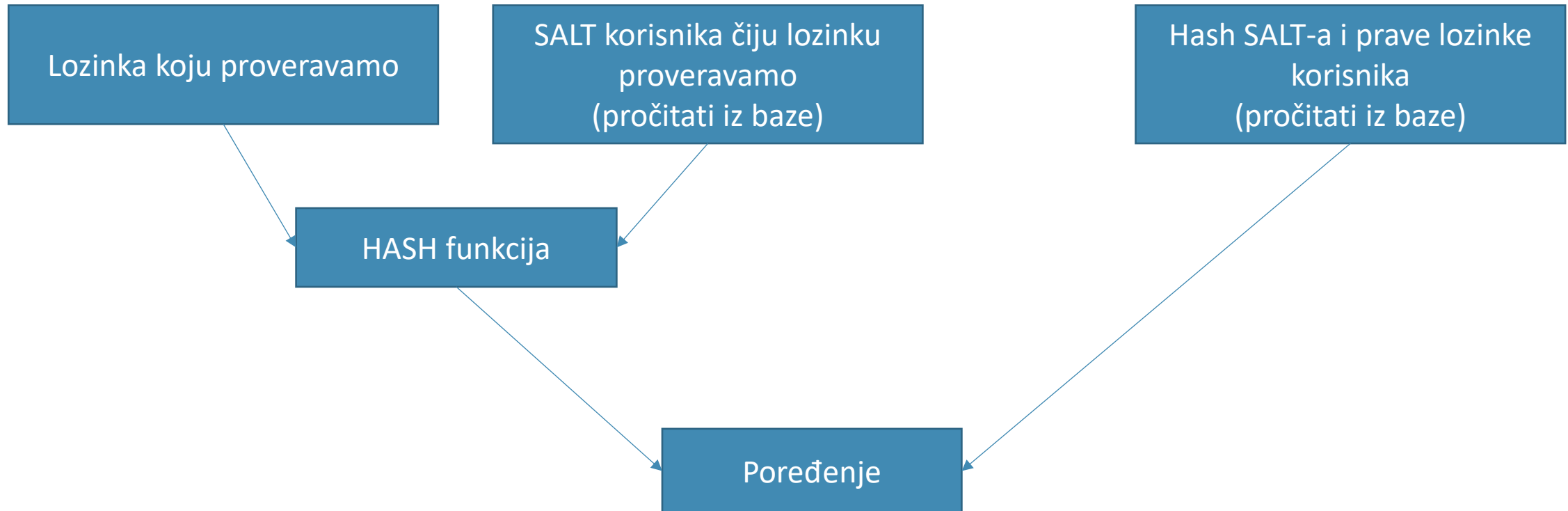


Samostalni rad

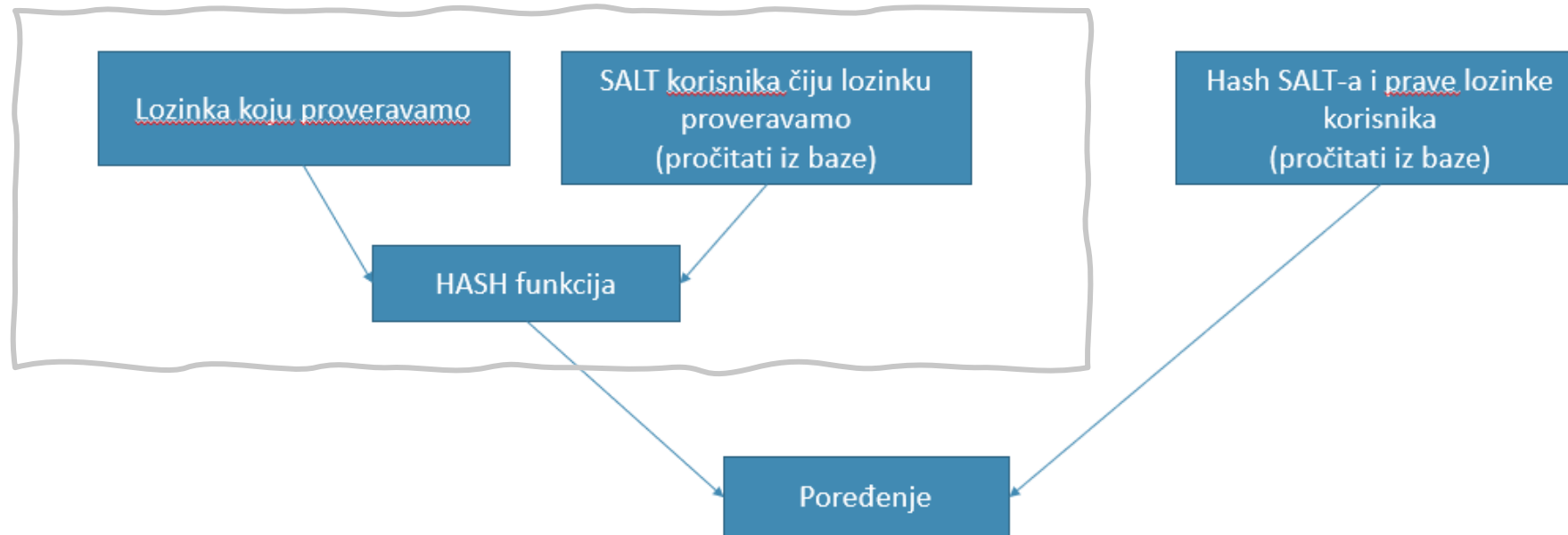
- Opis: Implementirajte Salt + Hashing u klasi **DatabaseAuthProvider** i metodi **validCredentials**
 - Vreme: **20 minuta**
1. Zamenite postojeći repozitorijum **UserRepository** sa **HashedUserRepository** u konstruktoru
 2. Primenite PBKDF2 sa
 - Algoritam za heširanje: PBKDF2WithHmacSHA512
 - Broj iteracija: 10 000
 - Broj bitova heširane vrednosti: 256
 - Koristite JAVA klase **SecretKeyFactory**, **PBEKeySpec**, **SecretKey**

passwordHash i salt u bazi su uskladišteni u base64 stringu

Samostalni rad



Samostalni rad – Primer



```
KeySpec spec = new PBEKeySpec(passwordToCheck.toCharArray(), salt.getBytes(), 65536, 128);
SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
byte[] hashedPassToCheckBytes = factory.generateSecret(spec).getEncoded();
byte[] hashedPassToCheckBase64 = Base64.getEncoder().encode(hashedPassToCheckBytes);
```

Rešenje zadatka

Demonstracija

Multi-faktorska autentifikacija

Koji su problemi u autentifikaciji sa lozinkom?

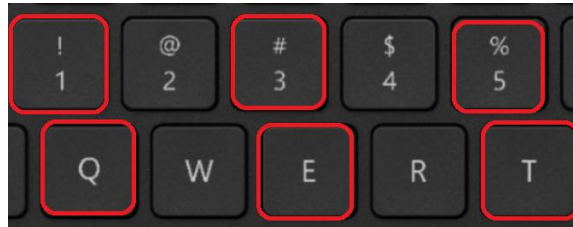
Koji su problemi u autentifikaciji sa lozinkom?

- Napadač može da ukrade ili kupi bazu username/password za neki hakovan servis
 - Korisnici obično koriste iste lozinke
 - Vaš sajt je osiguran hešovanjem lozinke, ali nisu svi na svetu!
 - „Credential surfing“ napad
- Krađa lozinke „social engineering“ napadom
 - „Administratoru treba lozinka“

Koji su problemi u autentifikaciji sa lozinkom?

Brute force napad ukoliko je slaba lozinka

- Mogu se uvesti zahtevi za jaku lozinku (min dužina, karakteri...), ali:
- Korisnici koriste „l33t speak“ lozinke (h@v3ANic4D@y) – podložni „dictionary napadu“
- Nalaze druge načine za izigravanje sistema poput redoslednih vrednosti (135!#%qetQET)



Multi-faktorska autentifikacija

Autentifikacija davanjem više dokaza koji nemaju isti izvor.

Faktori

- Znanje – nešto što znamo (lozinka, sigurnosna pitanja...)
- Posedovanje – nešto što imamo (token, kartica, ključ)
- Fizička osobina – nešto što jesmo (biometrija: otisak prstiju, dužica oka, glas, brzina kucanja...)
- Lokacija – negde gde jesmo (GPS pozicija, računarska mreža)

Dvo-faktorska autentifikacija

Kombinacija bilo koja dva faktora autentifikacije

- Znanje (lozinka) i posedovanje (token)
- Znanje (lozinka) i fizička osobina (otisak prsta)
- Posedovanje (ključ) i fizička osobina (dužica oka)

Postoji i tro-faktorska autentifikacija

Time-based One-time Password (TOTP)

- Algoritam za implementaciju tokena prema standardu RFC 6238
- Zasnovan je na vremenu i početnoj vrednosti (TOTP ključ)
 - Implementacija je takva da je reprodukcija jako teška
- Vezuje se za uređaj koji ima registrovan TOTP autentifikator
 - Google/Microsoft Authenticator, FreeOTP, OTP Auth
 - Svi implementiraju isti algoritam i kompatibilni su

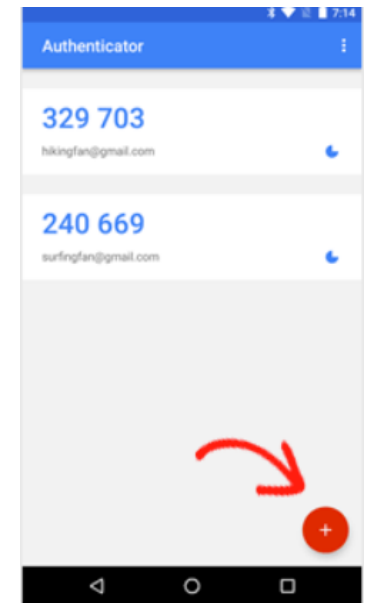
Koraci za implementaciju TOTP

1. Server: generiše TOTP ključ
2. Korisnik: registruje ključ u TOTP aplikaciji
3. Server: skladišti TOTP ključ u bazi
4. Korisnik: Prilikom svakog sledećeg logina upisuje TOTP vrednost iz TOTP aplikacije
5. Server: proverava prvi faktor (username i password)
6. Server: generiše TOTP vrednost od TOTP ključa iz baze podataka i poredi sa korisnikovom TOTP vrednošću

Enter this into Authenticator App

TQ6TTFIBFPR6OM2V

Submit



Registracija TOTP ključa

Demonstracija

[My Profile](#)[Register second factor](#)[Logout](#)

Enter this into Authenticator App

R7CUZOV5BZVAWC3X

Submit



```
@GetMapping("/register-totp")
public String showRegisterTotp(Model model, Authentication authentication) {
    GoogleAuthenticator gAuth = new GoogleAuthenticator();

    final GoogleAuthenticatorKey key = gAuth.createCredentials(); // TOTP key
    model.addAttribute("totpKey", key.getKey());

    // Used for QR Code
    final HashedUser user = (HashedUser) authentication.getPrincipal();
    String totpUrl = GoogleAuthenticatorQRGenerator.getOtpAuthTotpURL("Insecure Car Store", user.getUsername(), key);
    model.addAttribute("totpUrl", totpUrl);

    return "register-totp";
}
```

id="totpKey" je element koji sadrži TOTP ključ

id="qrcode" je element gde se generiše QR kod

id="totpUrl" je element koji sadrži URL za QR kod

QRCode je klasa iz JavaScript biblioteke za generisanje QR koda

```
<form th:if="${registered == null}" class="d-flex flex-column justify-content-around" style="height: 120px"
method="POST" action="/register-totp">
  <div class="form-group">
    <label for="totpKey">Enter this into Authenticator App</label>
    <input type="text" id="totpKey" readonly="readonly" class="form-control" name="totpKey" th:value="${totpKey}">
    <input type="hidden" id="totpUrl" disabled="disabled" name="totpUrl" th:value="${totpUrl}">
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>

  <div id="qrcode" class="mt-3"></div>
</form>
<div th:if="${registered != null}">
  Saved second factor successfully
</div>

<script type="text/javascript">
  const qrCodeTargetElement = document.getElementById("qrcode");
  const totpUrl = document.getElementById("totpUrl").value;

  new QRCode(qrCodeTargetElement, totpUrl);
</script>
```

```
@PostMapping("/register-totp")
public String registerTotp(@RequestParam() String totpKey, Model model, Authentication authentication) {
    final HashedUser user = (HashedUser) authentication.getPrincipal();
    repository.saveTotpKey(user.getUsername(), totpKey);
    model.addAttribute("registered", true);
    return "register-totp";
}
```

Priprema za samostalni rad

1. Instalirajte Google Authenticator ili FreeOTP aplikaciju
 - Google Play Store, Apple App Store, Microsoft Store
2. Pokrenite aplikaciju
3. Postojeći korisnik
 - username: **bruce**
 - password: **wayne**



Samostalni rad

- Opis: Implementirajte proveru TOTP vrednosti koju korisnik unosi
 - TOTP vrednost treba da se proverava samo ukoliko je korisnik registrovao token
- Vreme: **20 minuta**
- **DatabaseAuthProvider** već ima varijablu **Integer totp** u **authenticate** metodi
- Koristite metodu **authorize** klase **GoogleAuthenticator**
- Izmenite **DatabaseAuthProvider.validCredentials** metodu
- Preduslov je da koristite **HashedUserRepository**

Ubacuje parametar **totp** iz HTML Login forme u Authentication details polje

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .authorizeRequests()
        .antMatchers("/login").permitAll()
        .antMatchers("/**").authenticated()
        .and()
        .formLogin()
        .authenticationDetailsSource(request -> request.getParameter("totp"))
        .loginPage("/login")
        .loginProcessingUrl("/perform-login")
        .defaultSuccessUrl("/cars")
        .failureUrl("/login?error")
        .and()
        .logout()
        .logoutSuccessUrl("/login")
        .invalidateHttpSession(true)
        .deleteCookies("JSESSIONID");

    // We need this one in order to access h2-console
    http.headers().frameOptions().sameOrigin();
}
```

```
Object details = authentication.getDetails();
Integer totp = StringUtils.isEmpty(details) ? null : Integer.valueOf(details.toString());
```

Provera TOTP vrednosti

Demonstracija implementacije