

Izveštaj rađen za stručni kurs „Razvoj bezbednog softvera”

Milica Šopalović

Uvod

Projekat se izvodi na aplikaciji Real Book Store koja pruža mogućnosti pretrage, ocenjivanja i komentarisanja knjiga.

Aplikacija RealBookStore omogućava sledeće:

Pregled i pretragu knjiga.

Dodavanje nove knjige.

Detaljan pregleda knjige kao i komentarisanje i ocenjivanje knjige.

Pregled korisnika aplikacije

Detaljan pregled podataka korisnika

Primena alata za statičku analizu

Analiza je izvršena korišćenjem SonarQube alata.

Rezultati su sledeći:

- **Security issues:** 0
- **Security Hotspots:** 0
- **Reliability issues:** 5

- **Maintainability issues (code smells): 34**

Primeri detektovanih problema:

1. AuditLogger.java:13

- Problem: *Rename this field "LOG" to match the regular expression*
- Tip: Maintainability (Low)
- Ocena: False Positive (nema uticaja na sigurnost, samo konvencija imena)

2. AuditLogger.java:25

- Problem: *Rename method "audit" to prevent clash with field "AUDIT"*
- Tip: Maintainability (Blocker)
- Ocena: True Positive (može zbuniti developere i izazvati bug)

3. BooksController.java:18

- Problem: *Remove this field injection and use constructor injection instead*
- Tip: Reliability & Maintainability (Medium)
- Ocena: True Positive (injection može dovesti do problema pri testiranju)

4. SomeClass.java:85

- Problem: *Replace `Stream.collect(Collectors.toList())` with `Stream.toList()`*
- Tip: Maintainability (Medium)
- Ocena: False Positive (stari način i dalje funkcioniše, nije sigurnosni propust)

Na osnovu ovih rezultata zaključujemo da u kodu nisu pronađene sigurnosne ranjivosti, već samo problemi čitljivosti i održavanja.

The screenshot shows the SonarQube community interface. The top navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', 'Administration', and 'More'. The left sidebar has 'My Favorites' and 'All' tabs, and a 'Filters' section with 'Quality Gate' (Passed: 1, Failed: 0) and 'Security' (0 info, 0 low, 0 medium, 0 high, 0 blocker issues). The main content area displays the project 'RealBookStore' with a 'Passed' status. It shows the last analysis was 1 day ago with 1.4k Lines of Code in Java and XML. The metrics are: Security (A 0), Reliability (C 5), Maintainability (A 34), Hotspots Reviewed (E 0.0%), Coverage (0.0%), and Duplications (2.6%). The bottom footer contains links to SonarSource SA, Community Build, LGPL v3, Community, Documentation, Plugins, and Web API.

The screenshot shows the SonarQube community interface with the 'Issues' tab selected. The left sidebar has 'My Issues' and 'All' tabs, and a 'Filters' section with a message about looking for bugs. The main content area displays a list of issues for the project 'RealBookStore' in the file 'src/.../realbookstore/audit/AuditLogger.java'. There are 34 issues with a total effort of 2h 55min. Two issues are highlighted: 'Rename this field "LOG" to match the regular expression "[a-z][a-zA-Z0-9]*\$"' (Maintainability, Low) and 'Rename method "audit" to prevent any misunderstanding/clash with field "AUDIT"' (Maintainability, Blocker). Both issues are marked as 'Consistency' and 'convention'.

My Issues

All

Filters

Looking for Bugs, Vulnerabilities, or Code Smells? If your team prefers working with these types, change it in the [settings](#)

Software Quality

Security0

Reliability5

Maintainability34

Severity

Projects

Issues

Rules

Quality Profiles

Quality Gates

Administration

More

Rename "id" which hides the field declared at line 14.

Intentionality

Maintainability

Medium

cert

suspicious

...

+

Open

Not assigned

L26

5min effort

1 year ago

RealBookStore / src/.../realbookstore/controller/BooksController.java

Remove this field injection and use constructor injection instead.

Consistency

Reliability

Medium

Maintainability

Medium

No tags

+

Open

Not assigned

L18

5min effort

1 year ago

Remove this field injection and use constructor injection instead.

Consistency

Reliability

Medium

Maintainability

Medium

No tags

+

Open

Not assigned

L27

5min effort

1 year ago

My Issues

All

Filters

Looking for Bugs, Vulnerabilities, or Code Smells? If your team prefers working with these types, change it in the [settings](#)

Software Quality

Security0

Reliability5

Maintainability34

Severity

Projects

Issues

Rules

Quality Profiles

Quality Gates

Administration

More

Remove this field injection and use constructor injection instead.

Consistency

Reliability

Medium

Maintainability

Medium

No tags

+

Open

Not assigned

L21

5min effort

1 year ago

Remove this field injection and use constructor injection instead.

Consistency

Reliability

Medium

Maintainability

Medium

No tags

+

Open

Not assigned

L24

5min effort

1 year ago

Remove this field injection and use constructor injection instead.

Consistency

Reliability

Medium

Maintainability

Medium

No tags

+

Open

Not assigned

L27

5min effort

1 year ago

SonarQube™ technology is powered by [SonarSource SA](#)

Community Build · v25.9.0.112764 · MQR MODE

LGPL v3

Community

Documentation

Plugins

Web API

SonarQube™ technology is powered by [SonarSource SA](#)

Community Build · v25.9.0.112764 · MQR MODE LGPL v3 Community Documentation Plugins Web API

SQL Injection i Cross-site scripting

Na stranici za komentarisanje knjiga (`/books/{id}`) pronađena je ranjivost koja omogućava SQL Injection.

U polje za unos komentara ubačen je sledeći payload:

```
injection comment'); INSERT INTO persons (firstname, lastname, email) VALUES ('milica','sopalovic',''); –
```

Genres:

- sci-fi

Overall rating: **0.0**

My rating: **Not rated yet!**

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 [Rate](#)

Book comments

Bruce Wayne

injection comment

Add comment

injection comment'); INSERT INTO persons
(firstname, lastname, email) VALUES
('dobra', 'knjiga', 'View profile |
2	Sam	Vimes	night-watch@gmail.com	View profile
3	Tom	Riddle	theyGotMyNose@gmail.com	View profile
4	Quentin	Tarantino	qt5@gmail.com	View profile
6	dobra	knjiga		View profile

© 2023 Copyright: [RBS](#)

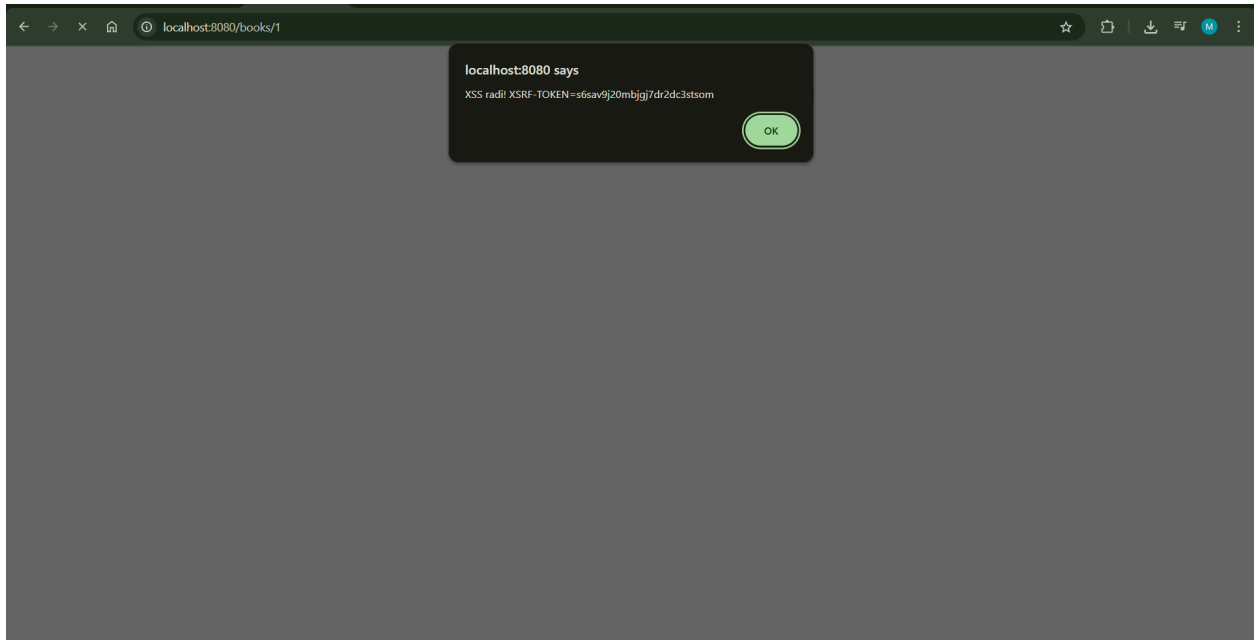
Ovime je ubačen novi korisnik u tabelu **persons**.

Na stranici **Users (persons.html)** korisnik se zatim pojavljuje sa malicioznim e-mail atributom, koji pri učitavanju pokreće **XSS skriptu** i ispisuje kolačiće sesije u konzoli.

Dokaz uspešnog napada:

- novi korisnik se vidi u tabeli (dobra knjiga)

- u **developer tools konzoli** ispisan je sadržaj **XSRF-TOKEN**.
- na alert prozoru pojavio se tekst "**XSS radi! TOKEN=...**"



Predlog odbrane:

Implementirati čuvanje komentara koristeći **PreparedStatement** umesto običnog **Statement**, čime se onemogućava ubacivanje proizvoljnog SQL koda (SQL Injection).

Umesto koda (ranjiva implementacija):

```

public void create(Comment comment) {
    String query = "insert into comments(bookId, userId, comment) values ("
        + comment.getBookId() + ", "
        + comment.getUserId() + ", '"
        + comment.getComment() + "')";
    try (Connection connection = dataSource.getConnection()) {
        Statement statement = connection.createStatement();
        statement.execute(query);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Potrebno je koristiti sledeću implementaciju (bezbedna verzija):

```

public void create(Comment comment) {
    String query = "insert into comments(bookId, userId, comment) values (?, ?, ?)";
    try (Connection connection = dataSource.getConnection()) {
        PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setInt(1, comment.getBookId());
        preparedStatement.setInt(2, comment.getUserId());
        preparedStatement.setString(3, comment.getComment());
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

Razlika je u tome što se u PreparedStatement vrednosti prosleđuju kroz metode (`setInt`, `setString`) i tretiraju se kao obični podaci, a ne kao deo SQL upita. Na taj način napadač ne može da „ubaci“ dodatne SQL komande kroz polje komentara.

Napad: Prikazivanje kolačića korisnika

Metod napada:

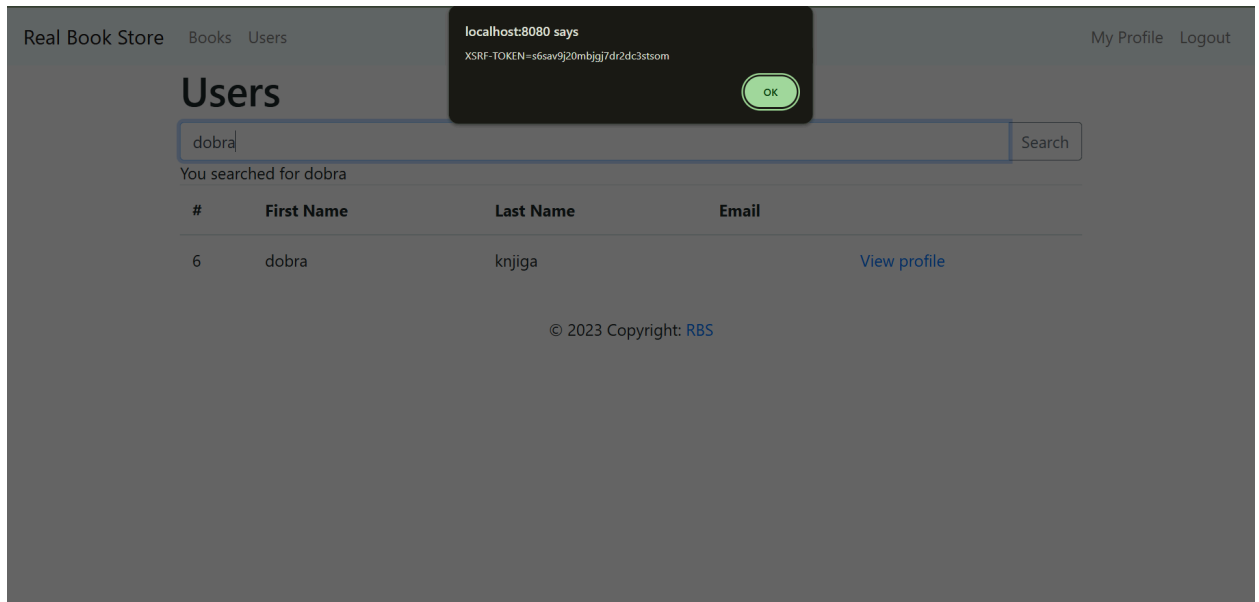
Na stranici *Persons* aplikacije postoji ranjivost koja omogućava XSS napad.

U polje za e-mail (koje se puni kroz SQL injection) ubacili smo sledeći payload:

```

```

Nakon što se izvrši pretraga korisnika, pregledač interpretira HTML i pokreće **alert** sa vrednošću kolačića sesije, što potvrđuje uspešan XSS napad.



Predlog odbrane:

Kako bi se sprečila injekcija HTML/JavaScript koda, potrebno je da se prilikom prikazivanja vrednosti u tabeli ne koristi **innerHTML**, već **textContent**. Na taj način se uneti sadržaj tretira isključivo kao tekst, a ne kao izvršiv kod.

Umesto koda:

```
tdElement.innerHTML = person.email; itd
```

Potrebno je koristiti:

```
persons.forEach(function(person) {  
    const tableRowElement = document.createElement(  
        let tdElement = document.createElement("td");  
        tdElement.textContent = person.id;  
        tableRowElement.appendChild(tdElement);  
        tdElement = document.createElement("td");  
        tdElement.textContent = person.firstName;  
        tableRowElement.appendChild(tdElement);  
        tdElement = document.createElement("td");  
        tdElement.textContent = person.lastName;  
        tableRowElement.appendChild(tdElement);  
        tdElement = document.createElement("td");  
        tdElement.textContent = person.email;  
        tableRowElement.appendChild(tdElement);  
        tdElement = document.createElement("td");  
        tdElement.textContent = '<a href="/persons/' +  
            tableRowElement.appendChild(tdElement);  
  
        tableContent.appendChild(tableRowElement);  
});  
  
document.getElementById('searchContainer').className  
document.getElementById('searchTerm').textContent =
```

Real Book Store Books Users My Profile Logout

Users

Search

You searched for dobra

#	First Name	Last Name	Email
6	dobra	knjiga	View profile

© 2023 Copyright: RBS

Nakon što je izvršena izmena koda (zamena `innerHTML` sa `textContent`), pokušaj XSS napada sa unosom `` više se ne izvršava. Kod se sada prikazuje samo kao običan tekst, bez pokretanja JavaScript-a. Time je ranjivost uspešno uklonjena, a funkcionalnost aplikacije je ostala nepromenjena.

Cross-Site Request Forgery (CSRF)

Napad

Aplikacija u početnoj verziji nije imala nikakvu zaštitu od CSRF napada. To znači da je bilo moguće izvršiti **neautorizovane promene podataka** tako što bi korisnik, dok je ulogovan, bio prevaren da klikne na stranicu koja u pozadini šalje zahtev prema aplikaciji.

Za demonstraciju je iskorišćen folder `csrf-exploit` u okviru projekta. Unutar fajla `index.html` nalazi se skripta sa funkcijom `exploit()`, koja prilikom klika na element *"Click here!"* šalje **POST zahtev** ka endpointu `/update-person` iz klase `PersonsController.java`.

U zahtevu se prosleđuju parametri:

- `id = 1`
- `firstName = "Batman"`
- `lastName = "Dark Knight"`

Users tabela (pre napada) korisnik sa `id=1` ima vrednosti Bruce Wayne:

Real Book Store

Books

Users

My Profile

Logout

Users

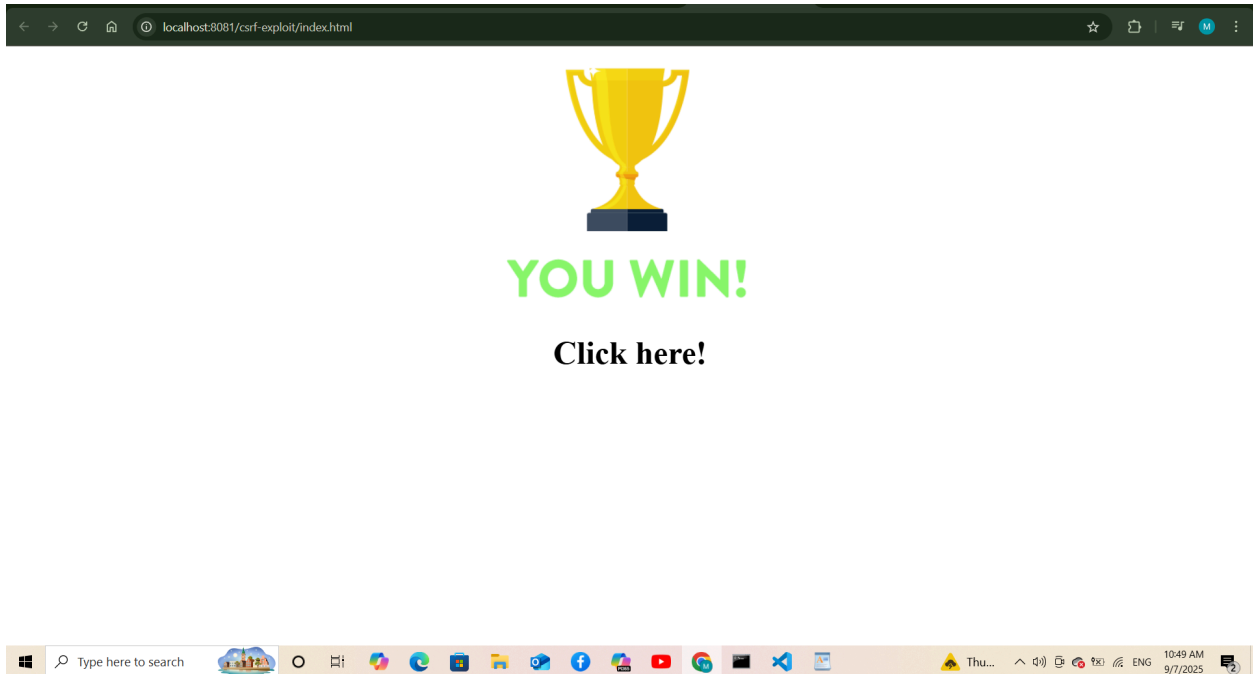
Search...

Search

#	First Name	Last Name	Email	
1	Bruce	Wayne	notBatman@gmail.com	View profile
2	Sam	Vimes	night-watch@gmail.com	View profile
3	Tom	Riddle	theyGotMyNose@gmail.com	View profile
4	Quentin	Tarantino	qt5@gmail.com	View profile
6	dobra	knjiga		View profile

© 2023 Copyright: [RBS](#)

CSRF exploit stranica prikazan je izgled lažne stranice sa porukom "You Win!":



Users tabela (posle napada) isti korisnik je promenjen u *Batman Dark Knight*, što potvrđuje uspešan CSRF napad.

A screenshot of a web application interface. The header shows 'Real Book Store' with links for 'Books' and 'Users', and a user profile section with 'My Profile' and 'Logout'. The main heading is 'Users'. Below it is a search bar with the text 'Search...'. A table lists four users. The first user, 'Batman Dark Knight', has an email 'notBatman@gmail.com'. The other users are 'Sam Vimes', 'Tom Riddle', and 'Quentin Tarantino'. Each row has a 'View profile' link. At the bottom, there is a copyright notice: '© 2023 Copyright: RBS'.

Odbrana

Odbrana je implementirana korišćenjem **CSRF tokena** koji se generiše i čuva u sesiji svakog korisnika (`CsrfHttpSessionListener`).

- U metodama `GET /persons/{id}` i `/myprofile` token se dodaje u **model** i zatim u formu kroz hidden input:

```
<input type="hidden" name="csrfToken" th:value="${CSRF_TOKEN}">
```

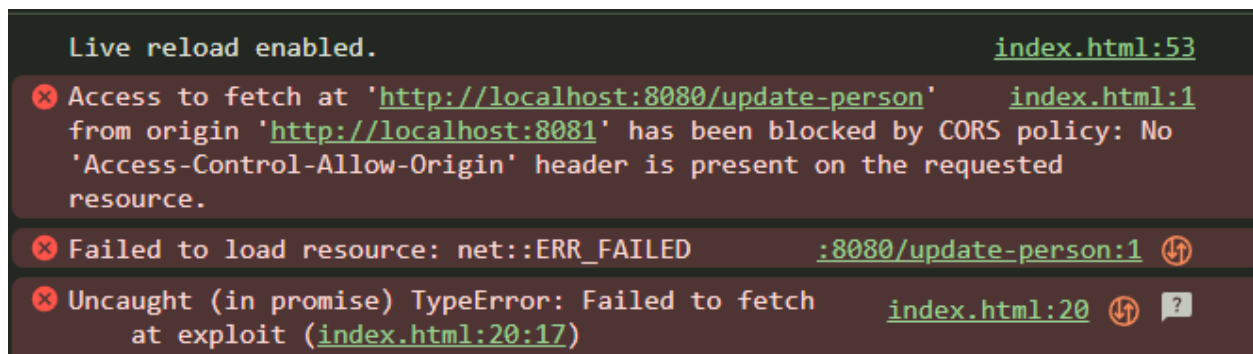
Mehanizam:

```
@GetMapping("/persons/{id}")
@PreAuthorize("hasAuthority('VIEW_PERSON')")
public String person(@PathVariable int id, Model model, HttpSession session) {
    model.addAttribute("CSRF_TOKEN", session.getAttribute("CSRF_TOKEN"));
    model.addAttribute("person", personRepository.get("" + id));
    return "person";
}
```

```
@PostMapping("/update-person")
@PreAuthorize("hasAuthority('UPDATE_PERSON')")
public String updatePerson(Person person, HttpSession session, @RequestParam("csrfToken") String csrfToken) throws AccessDeniedException {
    String csrf = session.getAttribute("CSRF_TOKEN").toString();
    System.out.println(csrf + " " + csrfToken + "kraj");
    if (!csrf.equals(csrfToken)) {
        throw new AccessDeniedException("Forbidden");
    }
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    System.out.println(username + " " + person.getFirstName());

    if(!username.equalsIgnoreCase(person.getFirstName())){
        throw new AccessDeniedException("Forbidden");
    }
    personRepository.update(person);
    return "redirect:/persons/" + person.getId();
}
```

Napad vise nije uspesan :)



Autorizacija

Autorizaciju sam uradila pomoću Spring Security-ja i metodskih anotacija `@PreAuthorize`, a zaštitu od CSRF-a sam dodala tokenom koji se čuva u sesiji i šalje kroz formu. Da bi metoda autorizacija radila, u klasi bezbednosne konfiguracije uključila sam `@EnableMethodSecurity`.

Permisije su primenjene kroz `@PreAuthorize` na kontrolerskim metodama, u skladu sa tabelom zadatka:

- **VIEW_PERSON** – pregled detalja osobe:

```
@GetMapping("/persons/{id}")
@PreAuthorize("hasAuthority('VIEW_PERSON')")
public String person(@PathVariable int id, Model model, HttpSession session) {
    model.addAttribute("CSRF_TOKEN", session.getAttribute("CSRF_TOKEN"));
    model.addAttribute("person", personRepository.get(id));
    return "person";
}
```

VIEW_MY_PROFILE – pregled sopstvenog profila:

```
@GetMapping("/myprofile")
@PreAuthorize("hasAuthority('VIEW_MY_PROFILE')")
public String self(Model model, Authentication authentication, HttpSession session) {
    String csrf = session.getAttribute("CSRF_TOKEN").toString();
    model.addAttribute("CSRF_TOKEN", session.getAttribute("CSRF_TOKEN"));
    User user = (User) authentication.getPrincipal();
    model.addAttribute("person", personRepository.get(user.getId()));
    return "person";
}
```

UPDATE_PERSON – izmena detalja osobe (isto važi i za brisanje):

```

@PostMapping("/update-person")
@PreAuthorize("hasAuthority('UPDATE_PERSON')")
public String updatePerson(Person person, HttpSession session, @RequestParam("csrfToken") String csrfToken) throws AccessDeniedException {
    String csrf = session.getAttribute("CSRF_TOKEN").toString();
    System.out.println(csrf + " " + csrfToken + "kraj");
    if (!csrf.equals(csrfToken)) {
        throw new AccessDeniedException("Forbidden");
    }
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    String username = authentication.getName();
    System.out.println(username + " " + person.getFirstName());

    if (!username.equalsIgnoreCase(person.getFirstName())){
        throw new AccessDeniedException("Forbidden");
    }
    personRepository.update(person);
    return "redirect:/persons/" + person.getId();
}

```

U klasi `WebSecurityConfig` omogućen je mehanizam methodske bezbednosti:

```

@Configuration
@EnableWebSecurity
@EnableMethodSecurity

```