Aplikacija za dijeljenje datoteka sa različitim pravim pristupa

Viduka, Ivan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:043820

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA DIJELJENJE DATOTEKA S RAZLIČITIM PRAVIMA PRISTUPA

Diplomski rad

Ivan Viduka

Osijek, 2022

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 28.06.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ivan Viduka
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1177R, 13.10.2020.
OIB studenta:	48971408826
Mentor:	Izv. prof. dr. sc. Ivica Lukić
Sumentor:	Izv. prof. dr. sc. Mirko Köhler
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv.prof.dr.sc. Zdravko Krpić
Član Povjerenstva 1:	Izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Aplikacija za dijeljenje datoteka sa različitim pravim pristupa
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Izraditi web stranicu za dijeljenje dokumenata između studenata. Skripte imaju različita prava pristupa, od javnog do privatnog među dorešenim korisnicima ili grupama. Datoteke se mogu sortirati po različitim paramterima kao što je godina studija, tip datotke i slično. Komentiranje datoteka imaju samo prijavljeni korisnici te ih dijeliti na društvenim mrežama. Omogućiti spremanje skripti na cloud. Rezervirano za: Ivan Viduka
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	28.06.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 20.07.2022.

Ime i prezime studenta:	Ivan Viduka
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1177R, 13.10.2020.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za dijeljenje datoteka sa različitim pravim pristupa**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivica Lukić

i sumentora Izv. prof. dr. sc. Mirko Köhler

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1 Zadatak diplomskog rada 1
2. PREGLED PODRUČJA TEME 2
3. SIGURNOSNE PRIJETNJE U IZRADI WEB APLIKACIJE 4
3.1 SQL injection 4
3.2 Probijanje autentikacije (engl. Broken authentication) 9
3.3 Cross-site scripting (XSS) 13
3.4 CSRF 16
3.5 Propusti u kontroli pristupa 20
3.6 Propusti u kriptiranju podataka 24
3.7 Neovlaštena promjena direktorija 26
3.8 Uskraćivanje resursa iskorištavanjem regularnih izraza (ReDoS) 30
3.9 Falsificiranje zahtjeva na server (engl. Server-side request forgery - SSRF) 32
3.10 Udaljeno izvođenje programskog koda (engl. Remote code execution - RCE) 35
4. RAZVOJNO OKRUŽENJE I FUNKCIONALNI ZAHTJEVI 38
4.1 Korišteni alati i tehnologije 38
4.1.1 Razvojni okvir Laravel 38
4.1.2 Composer 38
4.1.3 Artisan 38
4.1.4 Bootstrap 38
4.2 Funkcionalni zahtjevi na web aplikaciju 39
4.3 Ostale postavke projekta 40
5. REALIZACIJA ZADATKA 41
5.1 Prevencija sigurnosnih prijetnji 41
5.1.1 Prevencija SQL injection napada 41
5.1.2 Prevencija probijanja autentikacije 43
5.1.3 Prevencija XSS napada 45
5.1.4 Prevencija CSRF prijetnje 46
5.1.5 Prevencija propusta u kontroli pristupa 49
5.1.6 Prevencija propusta u kriptiranju podataka 51
5.1.7 Prevencija neovlaštene promjene direktorija 51
5.1.8 Prevencija ReDoS napada 52
5.1.9 Prevencija SSRF napada 53
5.1.10 Prevencija RCE napada 53
5.2 Korištenje aplikacije 53

6. ZAKLJUČAK 65
LITERATURA 66
SAŽETAK 67
ABSTRACT 68

1. UVOD

Pohranjivanje i dijeljenje datoteka postaje sve češće korištena internetska usluga. Potreba za konstantom raspoloživosti određenih informacija, zajedničkim korištenjem datoteka ili suradnjom pri njihovoj izradi te oslobađanjem lokalnog memorijskog prostora samo su neki od razloga korištenja ove usluge. Znatno povećanje ovog načina pohranjivanja i dijeljenja informacija u poslovnom svijetu uzrokovano je globalnom pandemijom i sve većim brojem zaposlenika koji rade od kuće. Dijeljenje podataka i suradnja s ostalim djelatnicima mora ostati na učinkovitoj razini, što se provodi koristeći različite usluge skladištenja podataka. Spremanje datoteka na udaljenim lokacijama, poput oblaka računala, omogućava korisniku pristup tim datotekama s različitih lokacija. Ovime se oslobađa prostor na lokalnom uređaju, olakšava se pristup datotekama i njihovo dijeljenje s drugim korisnicima te omogućava lakšu zaštitu pristupa. Ostvarivanje sigurnog korištenja jedan je od najvećih izazova aplikacija i usluga za dijeljenje podataka. Povećanje broja korisnika i sve veća ovisnost poduzeća o ovakvom obliku poslovanja dovodi do povećanja posljedica u slučaju sigurnosnih propusta.

U ovom diplomskom radu izradit će se web aplikacija čija je glavna svrha pohranjivanje i dijeljenje podataka. U teorijskom dijelu rada navest će se najčešće sigurnosne prijetnje web aplikaciji, koja im je svrha i način na koji djeluju. Zatim će se u praktičnom dijelu unutar navedene web aplikacije prikazati potrebni koraci za izbjegavanje ili smanjenje uspješnosti opisanih napada. Također, implementirat će se funkcionalnosti poput filtriranja datoteka te interakcije korisnika aplikacije putem komentara.

Nakon uvoda, drugo poglavlje donosi pregled trenutnog stanja i postojećih aplikacija na navedenom području. U trećem poglavlju daje se teorijska podloga, u kojoj se ukratko opisuju najčešće sigurnosne prijetnje. Četvrto poglavlje opisuje funkcionalnosti navedene aplikacije te tehnologije i alate korištene za njezinu izradu. Peto poglavlje prikazuje implementaciju stranice, s naglaskom na programska rješenja u sprječavanju ranije opisanih prijetnji.

1.1 Zadatak diplomskog rada

U diplomskom radu potrebno je navesti i objasniti najčešće sigurnosne prijetnje web aplikacija te kakve posljedice mogu imati ako nisu sanirane. Nadalje, bit će opisana postojeća web rješenja za pohranjivanje i dijeljenje datoteka te navesti funkcionalnosti aplikacije koja će biti implementirana u ovom radu. Potrebno je ostvariti programsko rješenje koristeći razvojni okvir Laravel, u kojem će se implementirati sigurnosne prevencije za navedene prijetnje. Na kraju, treba se prikazati upotreba navedene aplikacije i korištenje njezinih funkcionalnosti.

2 PREGLED PODRUČJA TEME

Postoji veliki broj dostupnih aplikacija koje obavljaju funkcionalnosti obuhvaćene u ovom radu. Najpoznatiji primjeri su različite varijacije usluga oblaka računala od velikih kompanija – Google Drive, Microsoft One Drive i Amazon Drive. Navedene kompanije pružaju određenu količinu prostora besplatno, koji se može koristiti za različite dokumente i fotografije, suradnju i praćenje projekata te verzioniranje datoteka. Pravo na memorijski prostor ostvaruje se registracijom u aplikacije navedenih kompanija, gdje je moguće pronaći i druge usluge – strojno učenje, analiza podataka, oglašavanje i slično. Google Drive je najpopularniji za skladištenje podataka jer pruža veće količine prostora za pohranu (15GB prilikom registracije). Omogućava jednostavno verzioniranje datoteka i dijeljenje s drugim korisnicima. Microsoft One Drive je također popularan zbog pohrane, ali prvenstveno u poslovnom smislu, zbog lakše integracije većeg broja zajedničkih korisnika i raznih sigurnosnih dodataka. Nudi mogućnost automatizirane sinkronizacije datoteka za Windows korisnike, pogotovo za datoteke vezane uz Office 365 uslugu. Amazon Drive nema poslovni način rada, ali svojim članovima omogućava neograničenu pohranu fotografija. Također, postoje aplikacije specijalizirane samo za pohranu. Najpoznatiji primjeri su DropBox, iCloud i Mega. Moguće je kombinirati količinu podatkovnog prostora s različitim sigurnosnim zahtjevima te sinkronizirati podatke s lokalnim uređajem. Navedene aplikacije mogu se koristiti u privatne i poslovne svrhe, a korisnik može odabrati već složene pakete usluga ovisno o svojim zahtjevima. Najveća prednost koju pruža Mega je sigurnost podataka i njihova enkripcija prilikom prijenosa. Također, omogućava sinkronizaciju datoteka i jednostavni pristup datotekama s mobilnih uređaja putem aplikacije. DropBox osigurava brzu sinkronizaciju i podržava verzioniranje datoteka. Osim toga, nudi i online verzije MS Officea, čime se omogućava direktna izmjena pohranjenih datoteka. Tu istu uslugu nudi iCloud, ali s programima koji su specifični za Apple proizvode. iCloud je prisutan na svim Apple uređajima, zbog čega je jednostavan za upotrebu i lako dostupan svim korisnicima. Također, velika količina podataka sinkronizira se automatski, jer iCloud poznaje strukturu datotečnog sustava.

Sve je veći broj projekata u kojima sami korisnici pokušavaju kreirati vlastiti oblik aplikacije, poput primjera [1]. U navedenom primjeru kreirana je web aplikacija za poduzeća koja žele testirati znanje svojih djelatnika u području za koje su djelatnici specijalizirani. Testovi se provode u obliku pitanja s višestrukim izborom odgovora, a pitanja su temeljena na području koje djelatnik treba poznavati. Rezultati testiranja dostupni su korisniku s administrativnim

ovlastima. Aplikacija omogućava pohranu djelatnika, ispitivanja koja se trebaju provesti te rezultata navedenih ispitivanja. Na temelju dobivenih informacija, poduzeće može procijeniti trenutnu razinu znanja svojih zaposlenika, usporediti ju s prethodnim rezultatima te ocijeniti ispunjavaju li uvjete za određenu poziciju. Također, aplikacija ima široko područje primjene, jer se jednostavno može personalizirati pitanja za točno određeno područje unutar poduzeća ili industrije. Korisnik s ovlastima administratora samo mora dodati pitanja kojima želi provjeriti karakteristična znanja svojih djelatnika. Aplikacija je namijenjena većem broju poduzeća, gdje se svako poduzeće promatra kao klijent koji plaća licencu za korištenje aplikacije. Razlog povećanja broja aplikacija ovog oblika je želja poduzeća za vlastitom verzijom platforme, koja ispunjava njihove specijalizirane zahtjeve i prilagođena je njihovom radu. Osim toga, aplikacija ovakvog tipa često spada pod projekte programera koji započinju raditi u nekom programskom jeziku ili razvojnom okruženju, iz razloga što obuhvaća veliki broj funkcionalnosti koje se koriste u raznim aplikacijama i projektima.

Primjer kreiran u ovom radu predstavljat će platformu za dijeljenje datoteka među studentima. Svaki korisnik moći će pohraniti datoteku te odlučiti hoće li ona biti javno dostupna ili ne. Javne datoteke će se moći preuzimati te pretraživati po određenim parametrima. Osim toga, aplikacija će imati određene karakteristike društvene mreže, jer će datoteke sadržavati opis i postojat će opcija ostavljanja komentara. Naglasak će biti na sigurnosnom dijelu aplikacije, poput zaštite pristupa i različitih razina korištenja aplikacije. Sigurnost podataka predstavlja sve veći izazov, zbog povećanog broja korisnika i platformi te same količine podataka koja se sprema. Propusti u zaštiti aplikacije mogu dovesti do povrede privatnosti korisnika i dijeljenja njihovih informacija, kao i informacija o djelatnicima i poduzeću kada se radi o poslovnoj upotrebi. U ovoj implementaciji pokušat će se prikazati rješenje za svaki potencijalni napad naveden u teorijskom dijelu diplomskog rada. Postoji veliki broj mogućih napada te su za ovaj rad odabrani najčešće isprobani napadi, kao i napadi s potencijalno najvećim posljedicama. Veliki broj navedenih sigurnosnih rizika nalazi se na OWASP popisu napada, koji se ažurira svake godine i sadrži informacije o novim i popularnim oblicima napada.

3 SIGURNOSNE PRIJETNJE U IZRADI WEB APLIKACIJE

U ovom poglavlju navest će se najčešće sigurnosne prijetnje na koje se mora obratiti pozornost prilikom izrade web aplikacije. Prijetnje obrađene u ovom radu odabrane su pomoću OWASP popisa najvećih prijetnji u prethodnoj godini te konzultacijama s mentorom. Za odabrane prijetnje dat će se njihov opis, objasniti postupak izvođenja napada i koji propusti dovode do njega te navesti preventivne metode kojima se osigurava uklanjanje ili minimiziranje uspješnosti napada. Osim prijetnji obrađenih u ovom radu, moguće je pronaći dodatne oblike napada u literaturi [2]. Također, općenite smjernice i obrasci za kreiranje sigurne aplikacije dane su u izvoru [3].

3.1 SQL injection

SQL injection, skraćenog naziva SQLi, predstavlja napad u kojem se koristi maliciozan SQL kod kako bi se manipuliralo bazom podataka. Kreira ih se tako da se utječe na upite koje aplikacija šalje svojoj bazi podataka. Ovi napadi mogu se koristiti za pristup informacijama koje ne bi trebale biti dostupne, nedozvoljenu izmjenu podataka, brisanje podataka iz baze ili uklanjanje same baze te slične napade. Podaci kojima se pristupa uglavnom se odnose na liste korisnika ili klijenata, njihove privatne informacije te osjetljive podatke vezane uz poslovanje samog pružatelja usluge. Osim neodobrenog pristupa ili brisanja tablica iz baze, jedan od opasnijih slučajeva je dobivanje administrativnog pristupa nad samom bazom, pri čemu osoba koja izvršava napad dobiva kompletan uvid u sve spremljene podatke. Najčešće mete napada su web stranice, na kojima se maliciozni SQL postavlja na mjestima gdje se očekuje unos određenih podataka od strane običnog korisnika, s namjerom spremanja tih informacija u bazu. SQL je standardiziran jezik za pristup i upravljanje bazama podataka, koji se koristi za dohvaćanje informacija koje će se prikazati korisniku, spremanje i ažuriranje korisnikovih unosa te brisanje zapisa koji više nisu potrebni. Dodatne informacije o samom napadu i zašto se on izvodi dostupne su u literaturi [4].

SQL injection napadi mogu se podijeliti na tri kategorije: In-band SQLi, Inferential SQLi i Out-of-band SQLi.

In-band SQLi predstavlja najčešći oblik napada, u kojem osoba koristi isti komunikacijski kanal za slanje malicioznog koda i primanje nedozvoljenih rezultata. Dva načina kojima se najčešće izvodi ova kategorija su napadi bazirani na greškama te napadi korištenjem Union naredbe.

Napadi bazirani na greškama oslanjaju se na poruku pogreške koju šalje server pri neispravnom unosu podataka. Cilj je iz dobivene poruke iščitati informacije o strukturi same baze podataka, koje se kasnije koriste za kreiranja napada i dobivanje informacija. Do ovog oblika napada dolazi kada se zaborave isključiti poruke o greškama, koje su jako korisne u fazi razvoja aplikacije, ali daju previše informacija i ne bi trebale biti dostupne običnim korisnicima. Prikaz napada dan je u primjeru 3.1.

```
//UNOS KORISNIKA
1 ORDER BY 4

// GENERIRANI UPIT (ODABIRE 3 STUPCA).
SELECT name, description, price FROM products WHERE id=1 ORDER BY 4

//VRAĆENA GREŠKA
ORA-01785: ORDER BY item must be the number of a SELECT-list expression.
```

Primjer 3.1. *Kreiranje neispravnog unosa s ciljem dobivanja informacija u grešci*

Iz vraćene greške saznajemo da u tablici postoje samo tri stupca, jer nije moguće sortirati po četvrtom. Vrijednost broja iza ORDER BY naredbe kreće od 2 i povećava se sve dok se ne dobije navedena greška, na temelju čega se saznaje ukupan broj stupaca tablice.

Napadi bazirani na korištenju Union naredbe su oblik napada u kojima se pri unosu podataka koristi SQL operator Union, zaslužan za kombiniranje rezultata dvije ili više SELECT naredbe. Dobiveni rezultat se vraća kao dio HTTP odgovora, a omogućava korisniku pristup informacijama iz tablica koje ne bi trebale biti dostupne i nisu predviđene za prikaz pri normalnom radu aplikacije. Kreiranje takvog napada dano je u primjer 3.2.

```
//UNOS KORISNIKA NAMIJENJEN ZA DOBIVANJE INFORMACIJA O KORISNICIMA
1 AND 1=2 UNION SELECT username, password, FROM members

//GENERIRANI UPIT
SELECT name, description, price FROM products WHERE category=1 AND 1=2 UNION SELECT
username, password, FROM members
```

Primjer 3.2. *Kreiranje napada pomoću UNION naredbe*

U navedenom primjeru kombiniraju se dva rezultata uz pomoć UNION operacije, ali pri unosu se namjerno uvodi neistinit uvjet *AND 1=2*. S time se osigurava da ne postoji dohvaćanje prvih rezultata (koji su primarni cilj ispravnog korištenja aplikacije), već da se samo dohvate rezultati drugog upita, u kojima se traži korisničko ime i lozinka svih korisnika.

Inferential SQLi odnosi se na napade u kojima se podaci i upiti šalju direktno na server te se promatraju odgovori servera i njegovo ponašanje na same upite. Pomoću ovih napada dobivaju se dodatne informacije o strukturi baze podataka. Također se naziva i izvođenjem napada naslijepo, jer se napadi ne izvode s web stranice na bazu i odgovori servera nisu unutar istog komunikacijskog kanala. Ovaj oblik napada traje duže i složeniji je za izvest, jer se podaci ne prenose sa stranice koja je namijenjena interakciji s bazom i potrebno je na temelju dobivenih odgovora samostalno rekonstruirati građu baze podataka. Unatoč tome, ovi napadi su jednako opasni kao i ostale kategorije. Dva načina izvođenja ove kategorije su napad temeljen na Boolean vrijednostima te napad temeljen na vremenu odziva.

Napad temeljen na Boolean vrijednostima odnosi se na napad u kojem se šalje upit na bazu podataka koji zahtjeva dva različita odgovora, ovisno je li upit uspješan ili neuspješan. Ovisno o rezultatu upita, HTTP odgovor će biti drugačiji. Na temelju dobivenog odgovora, osoba koja izvodi napad zna radi li se o uspješnom upitu ili ne, unatoč tome što ne dobiva nikakve podatke iz baze u odgovoru. Ovaj oblik napada je uglavnom spor, jer se moraju pogađati sve moguće kombinacije strukture same baze i njezinih tablica. Primjer izvođenja Boolean napada prikazan je na primjeru 3.3.

```
//UNOS NETOČNE TVRDNJE U URL STRANICE
http://newspaper.com/items.php?id=2 and 1=2
//GENERIRANI UPIT
SELECT title, description, body FROM items WHERE ID = 2 and 1=2
//UNOS TOČNE TVRDNJE U URL STRANICE
http://newspaper.com/items.php?id=2 and 1=1
//GENERIRANI UPIT
SELECT title, description, body FROM items WHERE ID = 2 and 1=1
```

Primjer 3.3. *Primjer unosa za izvođenje Boolean napada*

Prilikom prvog primjera, iz baze se neće vratiti ništa, jer se radi o netočnom uvjetu (očekivano ponašanje). U slučaju da se pri postavljanju ispravnog uvjeta, kao u drugom primjeru, dobije drugačiji odgovor od baze, znači da se ne provodi provjera unesenih podataka i da je stranica ranjiva na SQL injection napade. Nakon toga se mogu izvoditi ciljani napadi, kako bi se dohvatili podaci iz baze.

Napad temeljen na vremenu odziva funkcionira na sličan način. Šalje se upit na bazu, koji od baze zahtjeva da čeka određeni vremenski period prije slanja odgovora. Na temelju vremena potrebnog za slanje odgovora, može se zaključiti je li upit na bazu bio uspješno izvršen ili ne. Isto kao i prethodni napad, uspješnost se može odrediti iako baza ne šalje nikakve podatke u odgovoru, ali je konstruiranje i izvođenje napada poprilično sporo. Korištenje ovakvog napada prikazano je na primjeru 3.4.

```
//UNOS KORISNIKA
1-SLEEP(15)

//KREIRANI UPIT SA SLEEP FUNKCIJOM
SELECT * FROM products WHERE id=1-SLEEP(15)
```

Primjer 3.4. *Kreiranje upita pomoću SLEEP funkcije*

Korištenjem SLEEP funkcije, koja kao parametar prima broj sekundi koliko program treba mirovati, može se detektirati je li stranica ranjiva na napad. Ukoliko odgovor servera dođe nakon 15 sekundi čekanja, stranica je podložna malicioznim napadima i manipulacijom nad bazom.

Out-of-band SQLi nije jako čest, jer se temelji na određenim dopuštenjima koja moraju biti omogućena na serveru korištenom od strane aplikacije. Preduvjeti su da server može kreirati DNS ili HTTP zahtjeve, kojima donosi potrebne informacije. Uglavnom se koristi kao alternativa za ranije navedene kategorije.

Osim navedenih primjera, SQLi napadi mogu se izvesti manipulacijom URL putanje. Neki od mogućih slučajeva prikazani su na primjeru 3.5.

```
http://www.mycloud.com/files?id=55 or 1=1
SELECT * FROM Files WHERE id = 55 OR 1=1

http://www.mycloud.com/files?id=55; DROP TABLE Users
SELECT * FROM Files WHERE id = 55; DROP TABLE Users

https://webshop.com/products?category=Gifts --
SELECT * FROM products WHERE category = 'Gifts'-- AND ostali uvjeti
```

Primjer 3.5. *Dodatni primjeri SQLi napada*

Prvi primjer ponovno dodaje tvrdnju koja je uvijek istinita u uvjet, te time omogućuje dohvaćanje svih zapisa iz baze, umjesto jednog koji bi trebao biti dohvaćen pri ispravnom korištenju stranice. U drugom primjeru koristi se simbol ; koji označava kraj naredbe. Nakon toga unosi se nova naredba, koja briše tablicu s informacijama svih korisnika aplikacije. Zadnji primjer prikazuje korištenje simbola -- koji u SQL-u označavaju komentar. To omogućava da se zanemari ostatak naredbe, koji sadrži dodatne uvjete i ograničenja, te vraćanje informacija iz tablice koje ne bi trebale biti dostupne pri normalnom korištenju.

Posljednja prijetnja koja će se prikazati vezano uz ovaj oblik napada je SQLi drugog reda. Odnosi se na maliciozni kod koji se unosi u bazu, ali s namjernom kasnijeg izvođenja. Prvenstveni cilj ovog napada je proći provjere prije spremanja u bazu te spremljeni kod iskoristiti u budućnosti. Podaci koji su spremljeni u bazu smatraju se sigurnima te je moguće da se više ne izvršavaju provjere nad njima. Testiranje i otkrivanje takvih napada je znatno otežano, ali i samo izvođenje napada je kompliciran. Takav napad zahtjeva poznavanje organizacije baze podataka te interakcija koje se odvijaju između baze i aplikacije za vrijeme njezine upotrebe. Jedan od potencijalnih napada je putem registracije. Ukoliko osoba posjeduje račun s korisničkim imenom *ImeKorisnika123*, a ne postoje ograničenja za korištenje znakova u korisničkom imenu, moguće je registrirati novog korisnika s imenom *ImeKorisnika123* --. Kada se navedeno ime spremi u tablicu, smatra se da je sigurno za korištenje. Zatim se nakon prijave u aplikaciju odlazi na postavljanje nove zaporke, ali s obzirom da je -- oznaka za komentar, promijenit će se lozinka prvog korisnika (dobit će se neovlašteni pristup). Izgled izvedene naredbe u bazi podataka sličan je prethodnoj vrsti napada, a može se izvesti koristeći unos nalik onome u primjeru 3.6.

```
//DIO KOJI JE PODEBLJAN SE NE IZVODI ZBOG UPOTREBE OZNAKE ZA KOMENTAR
UPDATE users SET password='novaLozinka' WHERE username=’ ImeKorisnika123 ' -- ‘ and
password=’staraLozinka''
```

Primjer 3.6. *Primjer izvođenja SQLi napada drugog reda*

Ovaj oblik napada može se spriječiti ograničavanjem dozvoljenih simbola i znakova na one koji su sigurni te validacijom svih podataka, a ne samo onih prije unošenja u bazu.

Prvi korak u prevenciji SQLi napada je sanitizacija i validacija unesenih podataka. Njihovo korištenje osigurava identificiranje nedozvoljenih unosa. Provode se na način da se uspostave kriteriji koje uneseni podaci moraju ispuniti te ih se implementira u kodu pri provjeri korisničkog unosa. S obzirom da se ne može zaustaviti sve oblike napada, često se koristi

vatrozid za web aplikacije (engl. web application firewall - WAF). WAF se oslanja na velike popise koji se konstanto ažuriraju, a sadrže adrese s kojih su prepoznati napadi i liste potencijalno opasnih korisničkih unosa. Ukoliko se s neke od tih adresa primijeti zahtjev koji je sličan nekom s navedene liste, zahtjev se blokira.

Najbolji način sprječavanja SQLi napada je korištenje parametriziranih upita i pripremljenih izjava u SQL-u. Primjenjuju se u slučajevima kada se unutar upita koristi unos kojim se može manipulirati ili koristiti za potencijalno opasne radnje – preporuča se u svim slučajevima korištenja korisničkih unosa. Naredba koja se koristi u upitu mora uvijek biti napisana samostalno, bez korisničkih unosa. Oni se kasnije dodaju u obliku parametara, nakon prevođenja SQL naredbe. Primjeri korištenja parametara pri radu s bazu dan je u programskom kodu 3.7.

```
<?php
   $stmt = $dbh->prepare (
      "INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
   $stmt->bindParam(':name', $name);
   $stmt->bindParam(':value', $value);
?>

<?php
   $stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
   $stmt->execute([$_GET['name']]);
?>
```

Programski kod 3.7. *Korištenje parametrizacije za korisnički unos*

Varijable $name i $value predstavljaju unose korisnika te se dodaju u naredbu na njihova predviđena mjesta, nakon što se bazu obavijesti o kakvoj naredbi je riječ (u ovom slučaju operacija INSERT).

Korištenjem ovih funkcija sprječava se izmjena planirane naredbe. Ukoliko se u drugom primjeru za vrijednost parametra *name* upiše *Name OR 1=1* -- , neće doći do uvjeta koji je uvijek istinit i pretvaranja ostatka naredbe u komentar zbog znakova --, već će se u bazi tražiti korisnik s imenom *Name OR 1=1* --. Upit je već ranije poslan na bazi i ona zna koju operaciju treba očekivati, zbog toga ne dolazi do manipulacije nad izrazom.

3.2 Probijanje autentikacije (engl. Broken authentication)

Probijanje autentikacije označava skup različitih napada u kojima je cilj preuzeti jedan ili više korisničkih računa te dobiti ovlasti koje imaju njihovi stvarni vlasnici. Autentikacija se može smatrati ugroženom kada osoba koja izvodi napad ovlada tuđim računom pomoću probijanja zaporke, preuzimanja korisnikovih sesija ili ključeva, skupljanja informacija o korisničkom

računu ili ostalih detalja koji omogućavaju oponašanje stvarnog vlasnika od strane napadača. Ovakvi napadi najčešće se provode na dva načina: upravljanje vjerodajnicama i upravljanje sesijama. Strategije samog napada mogu varirati od masivnih pokušaja dobivanja pristupa do ciljanja točno određenog korisničkog računa. Vjerojatno najčešći oblik napada je umetanje vjerodajnica (engl. Credential stuffing).

Umetanje vjerodajnica napad je u kojem se koriste ranije prikupljene liste pristupnih podataka, poput korisničkih imena i zaporke, u različitim kombinacijama kako bi se ostvario pristup željenom korisničkom računu ili organizaciji. Navedene liste podataka mogu biti kupljenje ili rezultat nekog ranijeg napada na drugu organizaciju. Razlog zbog kojeg je ovaj oblik napada toliko uspješan je u tome što velika većina korisnika upotrebljava iste zaporke za veći broj računa. Osoba koja izvodi napad predaje listu pristupnih podataka u botnet – niz uređaja povezanih mrežom, od kojih svaki pokreće jedan ili više programa koji se izvode samostalno. U navedenoj mreži se istovremeno isprobavaju različite kombinacije pristupa na većem broju stranica. Kada na nekoj od njih pronađu uspješnu kombinaciju, dobiva se neovlašteni pristup korisničkom računu. Taj račun se dalje koristi za prodaju informacija ili korištenja, prikupljanje privatnih informacija o organizaciji (ako se radi o poslovnom računu) ili trošenje resursa za kupovinu u web trgovinama.

Ove napade moguće je spriječiti na više načina. Jedan je promjena samog oblika autentikacije, odnosno nekorištenje zaporki. Umjesto toga, korisnik prijavu izvršava putem biometrije ili uređaja koji je jedinstven i pripada samo njemu. Također, moguće je periodično tražiti ponovnu potvrdu takve prijave, kako bi se dodatno podigla razina sigurnosti. Jedan od najučinkovitijih načina sprječavanja ovih napada je višestruka autentikacija (engl. Multi-Factor Authentication - MFA). Temelji se na tome da korisnik mora koristiti još jedan način autentikacije kako bi potvrdio prijavu, uz korisničko ime i zaporku. To se može implementirati na više načina, poput otiska prsta, upisivanja jednokratnog koda poslanog na mobilni uređaj ili email poslan na račun korisnika. Napadač koji ima samo listu imena i zaporki nema pristup tim informacijama.

Slična vrsta napada iz ove kategorije je pogađanje ispravne kombinacije na velikom broju pokušaja (engl. Brute force). Temelji se na tome da se izračuna najveći broj kombinacija koji mogu tvoriti zaporku te konstantnoj izmjeni znakova i brojeva. Uglavnom se mijenjaju u obliku često korištenih fraza ili obrazaca. Ovaj napad je jako učinkovit ukoliko su korištene kratke ili jednostavne kombinacije. Još jedna od posljedica ovog napada je veliko povećanje prometa stranice i njezino potencijalno rušenje. S obzirom da se radi o jednostavnijem obliku

napada, lakše ga je i zaustaviti. Najjednostavniji način je osigurati složenu zaporku od strane korisnika. Prilikom registracije, korisnik mora kreirati lozinku minimalne duljine, koja sadrži kombinaciju velikih i malih slova, brojeva, posebnih znakova i simbola. Također, moguće je ograničiti maksimalan broj prijava, samog korisnika ili IP adrese s koje dolazi prijava, nakon čega se zabranjuje mogućnost prijave na određeni period i šalje se obavijest administratoru.

Prilikom spremanja korisničkih zaporki u bazu, česta je praksa korištenje hash funkcija. One su zadužene za spremanje zaporki u obliku dugog niza znakova, koji osiguravaju da napadač i dalje ne zna zaporku nakon što dobije pristup tablici. Međutim, ovaj pristup ne garantira potpunu zaštitu od brute-force napada. Moguće je masovno unošenje često korištenih riječi dok ne dođe do poklapanja s hashiranim oblikom, jer iste zaporke imaju isti zapis nakon hash funkcije. Kako bi se to spriječilo, dodaje se nasumični podatak na zaporku (engl. salt), koji osigurava da se svaka zaporka, makar bila identična zaporci drugog korisnika, obradi i spremi u jedinstveni niz znakova.

Druga kategorija napada u probijanju autentikacije naziva se upravljanje sesijama. Iako se ne zasniva na pristupnim podacima, također spada u ovaj oblik napada jer aplikacije koriste sesije i pristupne podatke u istu svrhu – utvrđivanje identiteta korisnika. U slučaju dobivanja neovlaštenog pristupa nad sesijama, moguće je oponašati stvarnog korisnika na isti način kao u ranije navedenim napadima. Web sesije predstavljaju niz interakcija između korisnika i stranice u određenom vremenu, koje se spremaju na web server. Uglavnom se spremaju vrijednosti koje korisniku olakšavaju korištenje stranice, poput informacija u obrascu, artikala stavljenih u košaricu za kupnju, prijašnje pretrage u tražilici ili posjećene web stranice i slično. S obzirom da se navedene vrijednosti spremaju na serveru, na korisničkoj strani sprema se identifikator od navedene sesije, koji mu omogućava dohvaćanje tih podataka sa servera. Identifikatori se mogu spremiti u obliku kolačića ili kao parametri URL putanje. Osim navedene upotrebe, sesija označava i naziv za vremenski period od prvog otvaranja aplikacije do prestanka njezinog korištenja.

Postoji više načina za izvođenje napada upravljanja sesijama. Najjednostavniji primjer je oduzimanje sesija, koji se temelji na tome da stvarni vlasnik izvede prijavu na stranici, ali se zaboravi odjaviti nakon korištenja. Ukoliko je moguće pristupiti računalu ili nekom drugom uređaju koje je korišteno za prijavu, ostvaruje se nastavak korištenja aplikacije s ranije navedenom sesijom. Još jedan sličan primjer je zapisivanje identifikatora sesije u URL stranice. Iako to omogućava prijenos ranije navedenih podataka, poput vrijednosti obrasca i sadržaja košarice, predstavlja opasnost ukoliko se koristi nesigurna mreža ili je URL javno

dostupan i vidljiv. Potrebno je samo preuzeti navedeni identifikator i moguće je koristiti aplikaciju poput prijavljenog korisnika.

Napad koji se može izvesti, ukoliko je učinjen propust od strane programera, je fiksiranje sesije. Ispravna praksa pri radu s aplikacijama je davanje novog identifikatora korisniku nakon prijave u aplikaciju, odnosno poništavanje onoga koji je dobio prilikom samog otvaranja stranice. Ukoliko napadač pronađe inicijalni identifikator, a novi se dodijeli nakon prijave, nema nikakve koristi od njega (ne može ga koristiti za pristup aplikaciji). Međutim, ako identifikator ostane isti i samo se navede da označava korisnika koji je prijavljen i ima pristup, osoba koja izvodi napad dobiva sve ovlasti bez pokušaja otkrivanja novog identifikatora.

Osim navedene izmjene identifikatora, moguće je dodatno utjecati na sigurnost sesije. Preporuča se ne koristiti identifikator sesije unutar URL putanje, već ih slati putem kolačića kreiranih od pouzdanih izvora. Na taj način se izbjegava mogućnost prepisivanja putanje i pogađanja tuđih sesija. Kada se sesija spominje u kontekstu vremenskog perioda korištenja aplikacije, potrebno je upravljati njezinim trajanjem. Sesije koje se koriste u aplikacijama nižeg rizika, poput streaming platformi ili portala, mogu trajati duži vremenski period kako se korisnik ne bi morao ponovno prijavljivati svaki put kada ih želi koristiti. Ako se radi o osjetljivim podacima, poput bankovnih aplikacija, odjavljivanje korisnika trebalo bi se napraviti nakon svega par minuta bez aktivnosti.

Od preostalih oblika napada, često se koriste pogađanje lozinki i lažno predstavljanje (engl. phishing). Pogađanje lozinki razlikuje se od umetanja vjerodajnica jer se ne koriste neovlašteno nabavljene liste pristupnih podataka, već popis često korištenih lozinki. Za razliku od brute-force napada, izbjegava blokiranje IP adrese zbog prečestih pokušaja na način da isprobava jednu zaporku s popisa uz sve poznate email adrese ili korisnička imena. Na taj način se ne izvodi tri ili pet pokušaja prijave za jednog korisnika, kolika je maksimalna granica u većini slučajeva, već se izvodi po jedna za svaki račun. Detaljnija objašnjenja i dodatne primjere napada moguće je vidjeti u literaturi [5].

Lažno predstavljanje temelji se na neopreznosti korisnika. Napad se uglavnom izvodi na način da se korisniku pošalje email u kojem se neovlaštena osoba predstavlja kao povjerljiv izvor i direktno od korisnika traži unos njegovih podataka. Ovim napadom može se ciljati veća skupina korisnika, u kojoj se šalje općenitiji oblik poruke, ili točno određeni korisnik, kada se poruka može više prilagoditi njemu da izgleda vjerodostojnije. Zaustavljanje ovih napada

zahtjeva određene korake i od korisnika i od pružatelja usluga. Navedene poruke potrebno je što ranije detektirati i spriječiti da dođe do korisnika, kako do potencijalne prijetnje uopće ne bi došlo. Međutim, ukoliko to ne bude uspješno, traži se od korisnika da prepoznaju opasnost. Za to je potrebno provesti edukaciju korisnika kako bi znali prepoznati kada se radi o lažnom predstavljanju, makar izgledalo kao pouzdani izvor. Također, korištenje višestruke autentikacije osigurava da čak i prilikom uspješno izvršenog napada neovlašteni korisnik nema pristup računu – jer mu fale ostali koraci prijave.

3.3 Cross-site scripting (XSS)

Cross-site scripting predstavlja potencijalnu opasnost u web sigurnosti, koja omogućava da osoba koja izvodi napad izmijeni inače sigurne stranice s kojima korisnik ima interakciju koristeći određene propuste. Uspješni XSS napadi omogućavaju napadaču da se predstavi kao stvarni korisnik, sa svim njegovim ovlastima i mogućnostima korištenja stranice. Najčešće se koristi zlonamjerni programski kod u jeziku JavaScript, koji se šalje korisniku. Ukoliko napadnuti korisnik ima veće ovlasti, moguće je uspostaviti kontrolu nad stranicom i ostvariti pristup osjetljivim podacima. U XSS napadima, sam napad izvodi se s klijentske strane, uglavnom u obliku zlonamjerne skripte. Unatoč tome, radi se o problemu koji se sprječava u backend dijelu programa, jer se izvodi na programskom kodu koji pruža ili obrađuje backend. Do napada najčešće dolazi kada se na stranici prikazuje sadržaj dobiven od korisnika ili nekog nepouzdanog izvora, a nije provedena sanitizacija, validacija ili kodiranje unosa. Zlonamjeran kod uglavnom cilja funkcionalnosti koje može izvršiti korisnikov web preglednik te ugrožava njegovu interakciju sa stranicom prikazujući opcije za preuzimanje, dodatni medijski sadržaj ili dodatke na stranici. Korisnikov preglednik ne može znati radi li se o nesigurnoj skripti, već ju samo izvodi. Na ovaj način može se ostvariti pristup korisnikovim kolačićima, sesijama ili tokenima, a u nekim slučajevima i prepisati sadržaj stranice.

Radi li se o stranici podložnoj takvom obliku napada može se zaključiti na jednostavan način, koristeći JavaScript funkcije poput *alert()* i *print()*. Ukoliko se navedene funkcije uspješno izvedu, na stranici se može izvesti XSS. Također je moguće izvesti napade koristeći HTML ili Flash kod, kao i sve oblike koda izvedive u pregledniku. Postoje tri kategorije XSS napada: odražavajući XSS (engl. Reflected XSS), spremljeni XSS (engl. Stored XSS) i XSS temeljen na strukturi stranice (engl. DOM-based XSS). Dodatne podjele i informacije o napadima dostupne su unutar literature [6].

Odražavajući XSS najjednostavniji je oblik napada. Odnosi se na napade u kojima se skripta odmah vraća s web servera, u obliku poruke s navedenom greškom, rezultata pretraživanja ili

nekog drugog oblika napada koji sadrži informacije odgovora na poslani zahtjev. Do njega dolazi kada aplikacija prima podatke iz HTTP zahtjeva te ih odmah koristi u odgovoru, bez da ih se provjeri. Izvor napada može biti e-pošta ili nekakva web stranica s već pripremljenim HTTP zahtjevom. Nakon izvođenja tog zahtjeva i reflektiranja napada od servera, preglednik ga izvršava na korisničkoj strani. Napad je prikazan u primjeru 3.8.

```
// URL PUTANJA S POTENCIJALNOM OPASNOSTI
https://website.com/status?message=<script> type= 'text/javascript'> alert('xss');</script>
```

Primjer 3.8. *Izvođenje XSS napada izmjenom URL putanje*

Navedeni maliciozni kod prikazuje obavijest s tekstom xss. Na stranici se ispisuje poruka *<script type='text/javascript'>alert('XSS');</script > not found*. S obzirom na navedeni ispis, zna se da stranica ne blokira XSS napad. Napadač može kreirati ovakve maliciozne stranice, primjerice za dohvaćanje kolačića iz preglednika, koje šalje korisnicima na email adresu. Kada korisnik pritisne navedeni link, skripta se izvodi u pregledniku i vraća napadaču željene informacije.

Spremljeni XSS također se naziva i XSS drugog reda. Odnosi se na napade u kojima stranica prihvaća podatke iz nepouzdanih izvora ili korisnika i sprema ih na serverima stranice (baze podataka, zapisi aplikacije, forumi i slično). Korisnik dohvaća spremljeni napad prilikom upita na bazu te se on izvodi unutar preglednika. Nesigurni podaci mogu se prenijeti unutar samog HTTP zahtjeva ili na druge načine – prikazi SMTP poruke u webmail aplikaciji, prikazivanje objava s društvenih mreža na stranicama za online trgovinu i slično. Potencijalni napad prikazan je u primjeru 3.9.

```
//UNOS U OBRAZAC U POLJE ZA KOMENTAR
Komentar na objavu <script src= "http://malicious-site.com/get-info.js">  </script>
```

Primjer 3.9. *Izvođenje XSS napada pomoć komentara na stranici*

Ukoliko se ovaj unos prihvati u bazu, a nema nikakve zaštite ili kodiranja pri ispisivanju, svaki puta kada se pristupi stranici na kojoj je prikazan taj komentar, izvodi se zlonamjerna skripta. Za razliku od prethodnog oblika napada, korisnik ne mora otići na navedeni link, već se on automatski izvodi pri učitavanju stranice.

XSS temeljen na strukturi stranice mogu se izvesti kada aplikacija sadrži JavaScript kod na klijentskoj strani, koji se koristi za obradu podataka. Ukoliko se obrada ne izvede na siguran

način, moguće je zlonamjerno korištenje aplikacije, poput ispisivanja novog sadržaja na stranici. Funkcija ranjiva za napad i sami unos napada dani su primjerom 3.10.

```
//OBRADA PODATAKA U JAVASCRIPT KODU STRANICE
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = 'You searched for: ' + search;
//IZVOĐENJE NAPADA
You searched for: <img src=0 onerror= alert(document.cookie);>
```

Primjer 3.10. *Izvođenje XSS napada iskorištavanjem propusta u programskom kodu*

Ukoliko napadač ima pristup polju za unos podataka, jednostavno može unijeti vrijednost sličnu ovoj u navedenom primjeru. S obzirom da dolazi do greške pri učitavanju slike, maliciozni kod se automatski izvodi. U ovom primjeru se kolačići ispisuju na zaslon, ali moguće ih je predati kao parametar u php skripti (koju je postavljena od strane izvršitelja napada). Na taj način se dohvaćaju informacije o korisniku bez da on to primijeti.

Rezultati izvođenja XSS napada mogu biti različiti. Većina se izvodi s ciljem dohvaćanja korisničkih informacija iz preglednika, odnosno uspostavom kontrole nad sesijom. Također je moguće otkrivanje korisnikovih privatnih informacija, instalacija trojanskih ili drugih vrsta zlonamjernih programa, preusmjeravanje korisnika na druge stranice ili izmjena sadržaja željene stranice. Posljedice samog napada variraju ovisno o vrsti stranice i ovlastima napadnutog korisnika. Čak i napadi na informativne stranice koje ne zahtijevaju prijavu mogu imati veliki utjecaj, ukoliko se izmijene vijesti na portalima, prikažu netočne informacije o cijeni ili vrijednosti dionica te promijene upute za korištenje lijekova i tehnologije.

Otkrivanje XSS prijetnji u aplikaciji može biti komplicirano. Najbolji način je odraditi sigurnosni pregled svih stranica aplikacije i pronaći gdje se sve koristi prikaz podataka dobivenih iz HTTP zahtjeva. Potrebno je unositi podatke u svaku pristupnu točku aplikacije te provjeravati HTTP odgovore. Također je potrebno provjeriti unose u URL putanju. Provjera JavaScript koda stranice mora se provesti detaljno, kroz sve funkcije. Iako se radi o dugotrajnom testiranju, i dalje nije moguće garantirati da su sve pristupne točke osigurane.

Osim navedenih provjera, uspješnost napada može se smanjiti na nekoliko načina. Kao i u drugim primjerima, sanitizacija ulaznih podataka i njihovo filtriranje s obzirom na dopuštene vrijednosti uklanja veliki broj prijetnji. Također, kodiranje unesenih podataka prilikom

prikaza na stranici sprječava da se zlonamjerni kod tumači kao aktivan sadržaj. Ovo se postiže korištenjem posebnih funkcija, poput *htmlentities()* i *htmlspecialchars()* u php-u te *escape()* u JavaScriptu. Njima se osigurava da se sadržaj baze prikazuje na stranici u onom obliku u kojem je zapisan. Za sprječavanje XSS napada u HTTP odgovorima koji ne bi trebali imati HTML ili JavaScript sadržaj, u zaglavlju se koriste oznake *Content-Type* i *X-Content-Type-Options*. Pomoću njih, web preglednik zna kako tumačiti odgovor na ispravan način.

Za dodatnu zaštitu koristi se pravilnik zaštite sadržaja (engl. Content security policy - CSP). To je mehanizam unutar web preglednika koji ublažava utjecaj XSS napada. Njime se ograničava način na koji se učitava različiti sadržaj stranice (HTML, CSS, JavaScript i slično). Dostupan je u većini web preglednika.

3.4 CSRF

Cross-site request forgery, ili skraćeno CSRF, prijetnja je koja omogućava napade koji korisnika navode da nesvjesno izvede malicioznu aktivnost, umjesto one koja je namijenjena na stranici aplikacije u kojoj je trenutno prijavljen. Ovisno o složenosti, uspješan napad može natjerati korisnika na različite aktivnosti – izmjenu email adrese ili zaporke, novčane transakcije, pružanje osobnih informacija i slično. Ovakav oblik napada zahtjeva određenu razinu socijalnog inženjeringa, odnosno manipulacije ljudima koja omogućava da oni nesvjesno izvedu opasnu radnju ili odaju privatne podatke. To se najčešće odnosi na slanje uvjerljive elektroničke poruke ili poveznice, koji u sebi sadržavaju maliciozni zahtjev na server. S obzirom da se radi o prijavljenom korisniku, nemoguće je razaznati je li zahtjev legitiman ili ne. Do toga dolazi jer većina zahtjeva iz preglednika sa sobom šalje sve dostupne informacije, poput kolačića, informacija o sesiji i IP adrese. CSRF napadi ciljaju funkcionalnosti koje izazivaju promjenu stanja na serveru (npr. promjena lozinke, korisničkog imena ili nekakvog iznosa) jer se odgovor servera šalje prijavljenom korisniku i napadač nema nikakve koristi od njega. Više detalja dostupno je na izvoru [7].

Preduvjet za izvođenje napada je poznavanje stranice i načina na koji ona funkcionira. Također, mora postojati relevantna aktivnost koja će se izvesti – aktivnost koja će napadaču donijeti određenu korist, poput promjene zaporke drugih korisnika ili izmjene prava pristupa na vlastitom računu. Nadalje, aplikacija svojim sesijama mora upravljati putem kolačića. Točnije, mora imati kolačić koji sadrži identifikaciju korisnika za tu sesiju. Ne postoji drugi način za verificiranje korisničkih upita, a da ovaj oblik napada bude uspješan. Na kraju, ne smiju postojati dodatni parametri zahtjeva kojima napadač nema pristup i ne može ih

pogoditi. Na primjer, ukoliko se pri zahtjevu za promjenu zaporke traži da se unese njezina trenutna vrijednost, a te informacije nisu nigdje dostupne, nemoguće je izvršiti napad.

Primjer izvođenja uspješnog napada prikazan je sljedećim zahtjevom. Ukoliko aplikacija omogućuje izmjenu email računa prijavljenog korisnika te je napadač upoznat s načinom funkcioniranja stranice, može navesti korisnika da nesvjesno pošalje zahtjev prikazan na primjeru 3.11.

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE
email=name@normal-user.com
```

Primjer 3.11. *Maliciozan zahtjev za promjenu email adrese*

Ovaj zahtjev ispunjava sve uvjete CSRF napada:

- napadač poznaje strukturu stranice i gdje može izvesti potencijalni napad
- napadač ima koristi od izvođenja ovog napada, jer nakon promjene adrese može zatražiti novu lozinku putem email poruke i na taj način potpuno ovladati računom
- aplikacija koristi kolačić za sesiju kako bi identificirala koji korisnik šalje zahtjev (ne može se prepoznati da ga ne šalje on, već napadač) i ne postoje drugi mehanizmi za verifikaciju korisnika
- napadač lako može odrediti potrebne vrijednosti u zahtjevu kako bi napad bio uspješan

Napadač korisniku šalje link na svoju stranicu, čiji kod izgleda približno primjeru 3.12.

```
<html>
   <body onload="document.forms[0].submit()>
        <form action="https://vulnerable-website.com/email/change"
          method="POST">
            <input type="hidden" name="email"
              value="name@evil-user.net" />
       </form>
   </body>
</html>
```

Primjer 3.12. *Forma za izvođenje CSRF napada*

S obzirom da se u ovom primjeru zahtjeva koristi POST HTTP metoda, napad se mora izvesti u obliku forme. Forma se podnosi putem klika na gumb za podnošenje forme, ali taj dio se automatizira pomoću JavaScript naredbe u početnoj *body* oznaci. Na taj način se ne oslanja na korisnika da u potpunosti izvede napad i smanjuje vjerojatnost da posumnja u legitimnost stranice. Nakon što korisnik klikne na link, događa se sljedeće:

- šalje se HTTP zahtjev na web stranicu, koji sadrži vrijednost novog email računa (ovog puta je to adresa napadača)
- kolačići potvrđuju da se radi o prijavljenom korisniku i povlače njegove podatke
- zahtjev se izvršava te je email adresa za korisnikov račun promijenjena

Izgradnja stranice za uspješan napad postaje teža s povećanjem broja parametara potrebnih za uspješan napad. U slučajevima gdje se koristi GET metoda, za izvođenje napada nije potrebno kreiranje posebne stranice. Napad se može izvesti na samoj stranici gdje je korisnik prijavljen, koristeći atribute HTML elemenata. Gornji primjer u GET metodi mogao bi se izvesti na način prikazan primjerom 3.13.

```
<html>
    <body>
      <img src="https://vulnerable-
           website.com/email/change?email=name@evil-user.net"
            width="0" height="0" border="0">
    </body>
</html>
```

Primjer 3.13. *Izvođenje CSRF napada pomoću GET metode*

Navedena slika mogla se poslati i putem elektroničke pošte, a korisnik ju ne bi primijetio zbog njezinih dimenzija. Preglednik bi izvršio zahtjev bez da korisnik primi ikakvu obavijest o tome. Stranice na kojima se CSRF može izvesti na ovakav način najčešće su mete napada. Razlog je jednostavan – napadač ne mora kreirati novu stranicu i temeljiti plan na korisničkom propustu, već se pouzda u stranicu koju korisnik svakako planira posjetiti i na koju mora biti prijavljen.

Jedina razlika između napada putem GET i POST metode je u tome kako ga korisnik nesvjesno izvodi. GET napade moguće je izvesti direktno na stranici, pomoću atributa elementa, a POST napad zahtjeva izvršavanje u obliku forme. Napadi pomoću metoda PUT ili DELETE većinom su blokirani u modernijim preglednicima, jer preglednici zahtijevaju da skripte s jedne stranice mogu pristupati skriptama s druge jedino kada se radi o stranicama koje imaju isto podrijetlo.

Postoji niz koraka koji se mogu poduzeti protiv CSRF napada i izgledati učinkovito, ali zapravo ne rješavaju problem. Jedan od njih je korištenje tajnog kolačića, odnosno kolačića s atributom koji osigurava da se koristi HTTPS protokol. Ukoliko se on ne koristi, zahtjev se odbija. Problem je u tome što će se prilikom izvođenja zahtjeva svi kolačići, pa čak i oni tajni, prenijeti u zahtjevu na server. S obzirom da se prenose identifikacijski kolačići, zahtjev će se svakako izvršiti, jer aplikaciji nema način da potvrdi radi li se o stvarnom korisniku ili ne. Također, HTTPS protokol nije garancija niti zaštita od CSRF napada. Iako osigurava veću razinu pouzdanosti stranice i treba biti preduvjet za upotrebu stranice, sami protokol ne čini ništa za sprječavanje tih napada. Nadalje, prihvaćanje samo POST oblika zahtjeva ne garantira potpunu zaštitu stranice. Kako je ranije navedeno, iako je otežano u usporedbi s korištenjem GET metode, i dalje je moguće kreirati i izvesti napad. Rješenje koje zaustavlja CSRF napade je nekorištenje sesija za identifikaciju korisnika, odnosno zapisivanje identifikatora u URL stranice. Ovaj način se ne preporuča jer omogućava druge oblike napada, poput ranije navedenog probijanja autentikacije.

Djelomično sigurno rješenje je podjela transakcije na veći broj manjih dijelova. To dodatno otežava izvršavanje napada, ali ga ne sprječava u potpunosti jer je moguće na dovoljnom broju pokušaja detektirati kako izgleda svaki korak transakcije i na taj način konstruirati svoj napad. Slično tome, moguće je korištenje *Referer* opcije u zaglavlju zahtjeva, koja osigurava da je zahtjev poslan s iste domene na koju je registrirana aplikacija. Ovu zaštitu moguće je zaobići dodavanjem meta oznaka u vlastitu HTML stranicu s koje se napad izvodi ili korištenjem domene koja se provjerava na mjestu poddomene vlastite stranice. Obje opcije dane su na primjeru 3.14.

```
<meta name="referer" content="never">
http://vulnerable-website.com.attacker-website.com/csrf-attack
```

Primjer 3.14. *Opcije za izbjegavanje Refer zaštite u zaglavlju*

Najbolji način za prevenciju CSRF napada je kreiranje CSRF tokena u svakom zahtjevu. Token se kreira nasumično i uz različite kombinacije znakova i simbola, kako bi se otežalo pogađanje njihovih vrijednosti. Token mora biti vezan uz korisnikovu sesiju i verificiran prije same obrade poslanog zahtjeva. Njegovo dodavanje u zahtjev utječe na preduvjete uspješnog izvršenja napada – aplikacija se ne oslanja samo na kolačiće kako bi potvrdila korisnika i postoji parametar u zahtjevu koji napadač ne poznaje i nema mu pristup. CSRF token treba se koristiti samo u POST zahtjevima, jer su GET zahtjevi vidljivi u povijesti pretraživanja i svoje

komponente ispisuju u URL, što ih čini lako dostupnima. Primjer 3.15. prikazuje zahtjeva koji koristi CSRF token.

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 68
Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm
csrf=WfF1szMUHhiokx9AHFply5L2xAOfjRkE&email=name@normal-user.com
```

Primjer 3.15. *Zahtjev koji sadrži CSRF token*

Za uspješnu uporabu ovog oblika zaštite od napada, potrebno je osigurati nekoliko koraka u izvedbi. Validacija tokena mora se provoditi u svim slučajevima. Ukoliko se ona provodi samo kada je token prisutan, to omogućava napadaču da ga u potpunosti ukloni iz zahtjeva i time osigura preskakanje validacije. Nadalje, potrebno je provjeriti pripada li navedeni token prijavljenom korisniku, odnosno CSRF token treba vezati uz trenutnu sesiju, a ne ga provjeravati globalno. Ukoliko se samo provjerava postoji li taj token i izostavi se provjera kome on pripada, napadač se može prijaviti na stranicu sa svojim računom, spremiti token koji je generiran za njega i onda ga koristiti u napadu na drugog prijavljenom korisnika. Zahtjevi koji imaju duplicirane vrijednosti tokena, tokene koji ne pripadaju prijavljenom korisniku ili ih uopće ne posjeduju, moraju se blokirati.

Dodatni koraci u prevenciji napada koje korisnik može učiniti su odjava sa stranice kada ona nije u upotrebi, ne pohranjivati lozinke u web preglednik, izbjegavanje korištenja drugih aplikacija dok je korisnik prijavljen na onoj koju koristi aktivno i slično.

3.5 Propusti u kontroli pristupa

Kontrola pristupa odnosi se na različite uloge koje korisnik aplikacije može imati te shodno tome različiti sadržaj i ovlasti koje su mu dostupne pri korištenju stranice. Predstavlja ograničenja na korisnicima i njihovim aktivnostima s ciljem osiguranja da korisnik ne može izvoditi one operacije ili pristupiti resursima koji mu nisu namijenjene. Propusti u kontroli pristupa obično dovode do dohvaćanja informacija koje bi trebale biti skrivene, izmjenama i brisanjima podataka nad kojima napadač nema ovlasti ili dobivanje ovlasti nad funkcionalnostima koje nadilaze korisnikovu trenutnu ulogu. Ovakvi propusti su dosta česti u aplikacijama, jer ovise o ljudskom faktoru pri donošenju odluka i implementaciji poslovne

logike, strukturi projekta, pravnim ograničenjima i samoj implementaciji. Najčešći propusti i razlozi uspješnosti ovih napada nalaze se u literaturi [8]. Unutar web aplikacija, kontrola pristupa temelji se na identificiranju i autentikaciji korisnika te upravljanju sesijama koje sadrže informacije poslane od strane korisnika putem zahtjeva. Implementirana kontrola pristupa zatim mora provjeriti može li navedeni korisnik izvršiti željenu radnju ili ne.

S obzirom na zadatke implementirane kontrole, kontrola pristupa može se podijeliti na tri kategorije: vertikalna kontrola pristupa, horizontalna kontrola pristupa i kontrola pristupa koja ovisi o kontekstu.

Vertikalna kontrola pristupa odnosi se na implementirane mehanizme koji ograničavaju pristup funkcionalnostima i podacima ukoliko korisnik nema ovlasti pristupa toj razini. Sadržaj aplikacije i njezine funkcionalnosti mijenjaju se ovisno o ulozi korisnika. Primjerice, korisnici s ulogom administratora imaju pristup opcijama za promjenu i brisanje računa drugih korisnika, ali te opcije nisu vidljive i dostupne običnom korisniku. Vertikalna kontrola pokriva najveći broj sigurnosnih propusta i često je sinonim za implementaciju sigurnosnih modula u aplikaciji, koji služe za odvajanje dužnosti različitih korisnika i kontrolu prikazanog sadržaja. Potencijalna prijetnja za napad nastaje kada se ne provedu potrebni koraci zaštite. Ukoliko se početna stranica za običnog korisnika i administratora razlikuje na način da korisnik ne vidi poveznice na stranice koje vidi administrator, to i dalje nije dovoljno da se aplikacija smatra sigurnom. Razlozi toga vidljivi su na primjeru 3.16.

```
https://insecure-website.com/admin
https://insecure-website.com/admin/page-name
https://insecure-website.com/document.txt
```

Primjer 3.16. *Primjeri stranica aplikacije kojima se može pristupiti izmjenom URL putanje*

Obični korisnik i dalje može pristupiti stranicama administratora bez poveznica, unoseći adresu stanice u URL putanju. S obzirom da većina stranica ima poseban prikaz za administratora, nije teško pogoditi putanju za pristup toj stranici. Čak i u slučajevima da je početna stranica zaštićena, moguće je da ostale stranice na koje ona pokazuje nisu. To je prikazano u drugom primjeru, gdje osoba koja izvodi napad može unositi često korištena imena stranica u obliku popisa te na dovoljnom broju pokušaja pronaći stranicu koja nije zaštićena, a pristup bi trebao imati samo administrator. Ukoliko se nazivi generiraju nasumično ili u određenom obrascu kombiniranja simbola, šanse za napad se smanjuju, ali i

dalje napad nije zaustavljen. Isto se odnosi na posljednji primjer, u kojem se ne cilja stranica, već neki oblik resursa. To može biti dokument s informacijama ili slika koja bi trebala biti dostupna samo kada je korisnik u ulozi administratora.

Napade pomoću manipulacije putanjom moguće je izvesti i u slučajevima kada parametri putanje služe za determiniranje prava korisnika. S obzirom da takva implementacija sama po sebi predstavlja lošu praksu, napad se može izvesti relativno jednostavno, što je vidljivo u primjeru 3.17.

```
//PARAMETRI NAKON PRIJAVE
https://insecure-website.com/login/home.php?admin=false
https://insecure-website.com/login/home.php?role=2
//PROMJENA PARAMETARA
https://insecure-website.com/login/home.php?admin=true
https://insecure-website.com/login/home.php?role=1
```

Primjer 3.17. *Izmjena parametara vidljivih u URL putanji*

Ukoliko se informacije o prijavi spremaju u putanju, a ne u obliku sesija ili kolačića, lakše ih je promijeniti. U slučaju da nakon uspješne prijave običan korisnik vidi putanju sličnu prvom primjeru, jednostavnom promjenom vrijednosti parametara napadač sebi može dodijeliti veće ovlasti. Navedeni napad moguće je izvesti i pomoću poziva API metode, ukoliko ona nije zaštićena. Primjer 3.18. prikazuje takvu izmjenu vrijednosti.

```
PUT /api/v1/management/users/user1 HTTP/1.1
Host: insecure -website.com

{
“id”:”1”,
“username”:“BasicUser”,
“userRole”:”user”, --“userRole”:”admin”,
“firstName”:”John”,
“lastName”:”Doe”
}
```

Primjer 3.18. *Izmjena vrijednosti unutar PUT metode*

Ukoliko se radi o funkcionalnosti gdje običan korisnik mijenja informacije o svojem profilu, on nema opciju promijeniti svoju ulogu. Ukoliko napadač uspije promijeniti vrijednost tog parametra, korištenjem skrivenog polja ili posrednog poslužitelja na kojem zaustavlja zahtjev,

može svojem profilu dati veće ovlasti. Ukoliko se napad sprječava samo na korisničkom sučelju, ali ne i unutar API poziva, postoji šansa za promjenom razine pristupa.

Horizontalna kontrola pristupa odnosi se na mehanizme koji ograničavaju pristup resursima unutar funkcionalnosti koje korisnik smije koristiti. Primjenjuje se na korisnike koji imaju istu razinu pristupa i ovlasti te koriste isti oblik resursa, ali moraju ispunjavati uvjete kako bi mogli pristupiti točno određenom resursu. Na primjer, svi prijavljeni korisnici u bankovnoj aplikaciji mogu izvesti plaćanje ili vidjeti ispis transakcija na svojem računu. Iako u svojoj ulozi ima pravo na tu funkcionalnost, prijavljeni korisnik ne može ostvariti pristup tuđim transakcijama ili provesti uplatu koristeći račun drugog korisnika. Napadi na razini horizontalne kontrole pristupa mogu sličiti onima s vertikalne razine. Najčešći primjer je manipulacija URL putanjom, poput izmjene prikazane primjerom 3.19.

```
//PUTANJA KOJU KORISNIK VIDI NAKON PRIJAVE
https://insecure-website.com/myaccount?id=123
//PUTANJA NAKON IZMJENE PARAMETRA
https://insecure-website.com/myaccount?id=451
```

Primjer 3.19. *Izmjena parametara URL putanje*

Ukoliko se identifikator korisnika ili nekog resursa vidi u putanji, moguće je njime upravljati. Izmjenom njegove vrijednosti, potencijalno se mogu dohvatiti sadržaji kojima korisnik nema pravo pristupiti, iako ima ovlasti za navedenu funkcionalnost. U primjeru kada se radi o društvenoj mreži, izmjena identifikatora objave može korisnika dovesti na iduću objavu koju je on kreirao, ali mu može omogućiti pristup objavi drugog korisnika, ukoliko postoji sigurnosni propust. Identifikator ne mora imati inkrementalne vrijednosti i moguće je koristiti složeni jedinstveni identifikator za smanjenje uspješnosti napada, kao u ranijim primjerima. Unatoč tome, i dalje se ne treba oslanjati samo na taj oblik osiguranja pristupa. Navedeni problemi mogu biti još veći u slučaju da se u putanji nalazi identifikator korisnika te se promjenom njegove vrijednosti dođe do korisničkog računa s administrativnim pravima pristupa. U ovom slučaju, ozbiljnost napada raste i on iz horizontalnog prelazi u vertikalni. Nakon ostvarivanja pristupa, napadač može upravljati tuđim računima ili promijeniti zaporku administrativnog računa, kako bi sebi osigurao stalni pristup.

Kontrola pristupa koja ovisi o kontekstu ograničava pristup resursima i funkcionalnostima ovisno o trenutnom stanju u kojem se aplikacija nalazi. Odnosi se na radnje koje se izvršavaju

u više koraka i sprječava da korisnik izvrši složenu radnju u neispravnom redoslijedu. Primjer takve kontrole je provedba kupovine u internet trgovini, gdje korisnik ne može više mijenjati sadržaj svoje košarice nakon što je provedeno plaćanje. Sličan postupak vrijedi i pri ažuriranju vrijednosti. U slučaju da administrator mora promijeniti informacije o korisniku, to će učiniti na način da prvo učita trenutne podatke, zatim napravi izmjene i u završnom koraku te izmjene provjeri i potvrdi podnošenje zahtjeva. Ukoliko u određenom koraku izostane zaštita ili validacija, otvara se mogućnost za neodobreni pristup neovlaštenom korisniku.

Od ostalih oblika napada može se izdvojiti korištenje API metoda u kojima nije implementiran nikakav oblik kontrole pristupa, manipulacija metapodatcima kako bi se promijenila razina pristupa (poput izmjena u kolačićima ili tokenima) i korištenje posredničkog poslužitelja ili VPN-a za pristup stranicama koje su blokirane na određenoj geografskoj lokaciji.

Najučinkovitiji način u borbi protiv ovih napada je korištenje dodatnih provjera prije izvršavanja samog zahtjeva. Aplikacija mora biti dobro dizajnirana i na strani korisničkog sučelja sadržaj mora biti kontroliran, ali implementacija verifikacije podataka na backend strani aplikacije osigurava da manipulacija podacima ili putanjama aplikacije ne bude uspješna. Osim toga, resursi koji nisu javno dostupni moraju imati zabranjen pristup. Potrebno je jasno definirati dostupne informacije i ovlasti svake uloge te shodno tome implementirati stranice i validacije zahtjeva. Također, potrebno je testirati sve putanje aplikacije u svim dostupnim korisničkim ulogama, kako bi se provjerila uspješnost implementacije. Nadalje, potrebno je onemogućiti izlistanje direktorija i njegovog sadržaja na serveru, kao i ne spremati sigurnosne kopije podataka na dostupnoj lokaciji servera. Poželjno je ograničiti količinu upita na API i zahtjeve sa stranice, kako bi se smanjila uspješnost automatiziranih napada. Na kraju, moguće je zabilježiti pokušaje neovlaštenog pristupa te o tome obavijestiti administratore ukoliko je broj pokušaja veći.

3.6 Propusti u kriptiranju podataka

Propusti u kriptiranju predstavljaju osjetljive podatke nad kojima nije provedena očekivana razina zaštite informacija ili se zaštita ne provodi uopće. Ovaj problem nekada se odnosio na izlaganje osjetljivih podataka, ali uzrok toga leži u propustima zaštite podataka. Osjetljivi podaci mogu predstavljati bilo koje informacije koje ne bi trebale biti javno dostupne, poput osobnih informacija korisnika, zaporki i identifikatora ili detalja o bankovnom računu. Osjetljive podatke se obrađuje i štiti na različite načine, ovisno o tome koliko su podaci bitni, ali svi trebaju proći kroz barem nekakav oblik sigurnosne zaštite. Ukoliko nije poznato kojoj

razini određeni podaci pripadaju, uvijek se koristi najviša moguća razina koja i dalje podatke čini upotrebljivima. Propusti ne moraju nužno označavati nekorištenje zaštitnih mehanizama, već se za propust smatra bilo kakvo odstupanje od dogovorenih načina obrade za neki tip podatka.

Propusti se mogu dogoditi u različitim fazama rada aplikacije. Primjerice, prenošenje podataka u izvornom obliku putem mreže. Korištenje kriptografskih protokola, poput TLS ili SSL protokola, osigurava se sigurna komunikacija na Internetu. Ukoliko se ne koristi zaštita u komunikacijskim protokolima, ostavlja se velika mogućnost za neovlašteni pristup podacima. Slični problemi nastaju i pri korištenju zastarjelih algoritama kriptiranja ili nesigurnih protokola za komunikaciju. Algoritmi poput MD5, SHA1 i PKCS v1.5 najčešće imaju problem s manjkom nasumičnosti (računalo lako pronađe obrazac za probijanje zaštite) ili imaju manu u radu algoritma koju se može iskoristiti. Pri izostanku TLS-a ili kompleksnog algoritma zaštite podataka, napadač može motriti promet na stranici i presresti zahtjev. S obzirom da su informacije vidljive u stvarnom zapisu ili ih se može jednostavno dešifrirati, napadač može dobiti sve informacije o korisniku. Ima direktan pristup vrijednostima u zahtjevu, kolačićima korisnika, kao i sesijama s podacima za prijavu na korisnički račun.

Osim odabira kvalitetnog algoritma kriptiranja, potrebno ga je ispravno koristiti kako bi podaci bili sigurni. Jedan od propusta je ponovna upotreba nasumičnog dodatka za zaporku. Kako je ranije navedeno, taj dodatak povećava složenost algoritma i osigurava jedinstvenost zapisa. Ukoliko se isti dodatak koristi u više slučajeva, napadač zna da je određeni dio konačne enkripcije predvidiv te se time smanjuje vrijeme potrebno za dešifriranje informacija. Ako baza podataka koristi ovakav pristup za šifriranje zaporki, napadač može doznati prave vrijednosti u relativno kratkom periodu. Za to mu mogu poslužiti liste ranije generiranih hash vrijednosti. Možda i najveća opasnost za dešifriranje podataka nastaje kada se koriste jednostavni ključevi za enkripciju. Ukoliko napadač pristupi navedenim zaštićenim podacima i izvede napad na opisan način, uspješnost napada ovisi o složenosti ključa. Napadač mijenja vrijednosti ključa dok zaštićeni podaci ne poprime smislene vrijednosti. Ovaj pristup je sličan napadima s velikim brojem pokušaja, ali daje bolje rezultate. Ključevi trebaju biti dugački nizovi nasumičnih simbola, u različitim i nesmislenim kombinacijama, a ne poznate i korištene riječi. Također, navedene ključeve potrebno je često mijenjati. Time se osigurava da napadač nema pristup podacima, čak i slučajevima kada uspije saznati stariju vrijednost enkripcijskog ključa. Prilikom zamjene, ključ se postavlja na novu vrijednost i sav sadržaj koji treba biti zaštićen ponovno prolazi kroz proces enkripcije. Ovaj proces treba biti potpuno

automatiziran i sigurnosne kopije moraju postojati dok se proces zamijene ne izvrši u potpunosti. Nakon toga ih je potrebno obrisati ili ažurirati, kako stara vrijednost ključa ne bi predstavljala prijetnju.

Od ostalih opasnosti treba izdvojiti izostanka provedbe enkripcije u vidu sigurnosnih uputa u HTTPS zaglavljima, izostanak validacije certifikata pouzdanosti na serveru s kojim se komunicira, ne korištenje ili korištenje jednostavnih vrijednosti za inicijalizacijski vektor te korištenje vlastitih hash funkcija.

Glavni cilj korištenja kriptografije nije u stvaranju savršenih i neprobojnih šifriranih vrijednosti, već kreiranje vrijednosti koje se ne mogu otkriti u razumnom vremenskom periodu, koristeći trenutno dostupne resurse za obradu. Ukoliko je dešifriranje podataka otežano i stavlja velike zahtjeve na osobu koja ih želi otkriti, vjerojatnost napada se smanjuje. Također, uvijek se trebaju koristiti već razrađeni algoritmi. Kreiranje vlastitog algoritma vremenski je jako zahtjevno, a postoji velika vjerojatnost da i dalje sadrži lako iskoristivu manu. Široko korišteni algoritmi imaju veliku razinu složenosti i samo njihovo korištenje je potvrda da ispravno odrađuju svoju zadaću. Preporuke za osiguravanje baze podataka kao i koraci za njezinu zaštitu opisani su u izvoru [9].

Prevenciju ovih napada može se provesti na više načina. Najsigurniji način je ne spremati osjetljive podatke ako ih se ne koristi. Ukoliko za navedenim podacima nema potrebe, bilo u radu aplikacije ili informacijama o korisniku, ne treba ih tražiti i pohranjivati. Nadalje, podatke koji će se koristiti treba analizirati prije same izrade aplikacije. Potrebno je uočiti koji podaci zahtijevaju višu razinu zaštite, uskladiti pohranu podataka sa zakonskim regulativama te odrediti jasne smjernice za implementaciju kriptografije nad podacima. Inicijalizacijski vektori moraju sadržavati kvalitetne početne vrijednosti, koje ne smiju biti korištene više puta za isti ključ. Također, potrebno je osigurati korištenje kvalitetnih algoritama zaštite i osigurati ranije opisane zamjene enkripcijskih ključeva. Spremljene zaporke trebaju koristiti nasumični dodatak pri kreiranju hash vrijednosti. Svi podaci koji se prenose moraju biti osigurani koristeći TLS protokol ili sličnu inačicu zaštite. Osim navedenih koraka, preporuča se ne spremati osjetljive podatke u priručnu memoriju.

3.7 Neovlaštena promjena direktorija

Neovlaštena promjena direktorija, poznata kao path ili directory traversal, predstavlja napad kojim osoba dobiva pristup zaštićenim datotekama na temelju manipuliranja putanjama u direktoriju. Cilj ovog napada je pristupiti datotekama koje se nalaze izvan direktorija same

aplikacije. Ovo se može odnositi na datoteke koje sadrže izvorni kod stranice ili konfiguracijske datoteke, podatke za prijavu i pristup upravljačkim funkcionalnostima, informacije o operacijskom sustavu i slično. Naziva se još i *dot-dot-slash* napad, jer niz simbola „ ../ “ služi za izmjenu putanje, odnosno vraćanje jednu razinu iznad u stablu direktorija. Većina stranica u ispravnom načinu rada ograničava pristup na direktorij koji sadrži datoteke potrebne zarad aplikacije, a ovaj oblik napada omogućava zaobilaženje tih ograničenja.

Bilo koja web aplikacija složenija od samog prikazivanja sadržaja, potencijalno je ranjiva na promjenu direktorija. Kada se učitavaju ili koriste resursi, poput slika, dokumenata i skripti, postoji rizik da napadač dobije pristup sadržaju koji je zabranjen. Ovaj oblik napada najčešće se događa kod funkcionalnosti učitavanja ili preuzimanja datoteka sa servera koristeći parametar iz zahtjeva. Mogućnosti ovakvog napada dane su u primjeru 3.20.

```
// ORIGINALNA PUTANJA
http://www.website.com/page.php?file=/files/File.txt
<img src="/loadImage?filename=218.png">
// IZMIJENJENA PUTANJA
http://www.website.com/page.php?file=../../etc/passwd
<img src="/loadImage?filename=../../etc/passwd">
http://www.website.com/page.php?file=/var/www/html/private/file.conf
```

Primjer 3.20. *Opcije izvođenja path thraversal napada*

Većina aplikacija smještena je u */var/www* direktoriju. Ukoliko aplikacija uzima parametar iz URL-a bez ikakve provjere, potpuni oblik navedene putanje bit će */var/www/../../etc/passwd*. Svaki niz simbola ../ poništava najbliži direktorij s lijeve strane, tako da je skraćeni oblik putanje samo */etc/passwd*. Ovime je napadaču omogućen pristup datoteci koja sadrži sve lozinke i informacije o korisnicima servera. Isti napad može se izvesti u bilo kojem dijelu aplikacije koji pruža opciju preuzimanja datoteke, uz poznavanje strukture projekta. Zadnji primjer prikazuje još jednu opasnost, a to je korištenje apsolutne putanje. U slučaju poznavanja lokacije određene datoteke, a to je najčešće slučaj kada server vraća povjerljive informacije u porukama o grešci, napadač umjesto korištenja ../ može direktno navesti cijelu putanju kao parametar.

Također, moguće je uključiti skripte ili stranice s druge web aplikacije, najčešće postavljene od strane samog napadača. Ona uglavnom sadrži maliciozni kod, koji se može iskoristiti za drugačiji oblik prijetnje poput ranije opisanog Cross-site scripting napada. Moguće ga je izvesti izmjenom putanje na način sličan primjeru 3.21.

```
http://mysite.com.hr/web-page?page=http://other-site.com/other-page.php/malicius-code.php
```

Primjer 3.21. *Izvođenje napada korištenjem putanje za malicioznu stranicu*

Osim direktne manipulacije putanjom, moguće je mijenjati i druge parametre. Kako bi napad bio uspješan potrebno je poznavanje rada stranice i načina obrade zahtjeva, ali u određenom mjeri ga je moguće izvesti. Najčešća meta ovih napada su vrijednosti unutar kolačića, jer osoba koja izvodi napad ima uvid u njihovo imenovanje i strukturu. Potencijalni napad prikazan je u primjeru 3.22.

```
// PHP SKRIPTA KOJA KORISTI VRIJEDNOSTI IZ KOLAČIĆA ZA UČITAVANJE DATOTEKE
<?php
   $layout = 'homepage.php';
   if ( is_set( $_COOKIE['LAYOUT'] ) ) {
     $layout = $_COOKIE['LAYOUT'];
   }
   include ( "/project/views/layouts/" . $layout);
?>

// IZMJENJENA VRIJEDNOST KOLAČIĆA U  ZAHTJEVU

GET /page.php HTTP/1.0

Cookie: LAYOUT=../../../../etc/passwd

 // ODGOVOR SERVERA

HTTP/1.0 200 OK

Content-Type: text/html

Server: Apache

root:fi3sED95ibqR6:0:1:System Operator:/:/bin/ksh
```

Primjer 3.22. *Izvođenje napada promjenom vrijednosti kolačića*

Ponavljanjem niza ../ osigurava se da *include()* funkcija ne učitava datoteku iz željenog direktorija, već da se vrati do početnog direktorija i učita datoteku s lozinkama. Ukoliko osoba koja izvodi napad nije sigurna koliko razina navedeni sustav sadrži, povećanje navedenog niza osigurava vraćanje u početni direktorij, jer svako pozivanje ../ u njemu

ostavlja korisnika na istoj lokaciji. Iako to olakšava izvođenje napada, i dalje je potrebno poznavati rad same aplikacije, tako da napad ostaje poprilično složen.

Izgradnja aplikacije bez korištenja osjetljivih datoteka u podatkovnom sustavu, poput onih za konfiguraciju sustava ili pristup bazi podataka, nije moguća. Ukoliko ovakvi propusti nisu primijećeni, napadač ima velike šanse ostvariti pristup cijelom podatkovnom sustavu, sistemskim programima i zaštićenim informacijama.

Uspješnost ovakvog oblika napada temelji se na propustima s korisničke strane, ali i pogreškama pri konfiguracijama servera. Dodatni detalji ovog oblika napada dostupni su u radu [10]. S obzirom da je nemoguće kreirati aplikaciju u kojoj korisnik ne može raditi izmjene nad putanjama, potrebno je ispravno odraditi njihovu validaciju.

Najčešća pogreška koja stvara dojam uspješne zaštite je blokiranje ili brisanje ../ niza. Sama ideja je ispravna, jer blokiranje tih simbola onemogućava izlazak iz direktorija predviđenog za rad aplikacije. Međutim, navedeni niz moguće je staviti u putanju i na drugačije načine. Korištenjem URL kodiranja, simboli ../ mogu se zapisati kao niz znakova *%2E%2E%2F* te imati isti učinak. Također, postavljanje očekivane vrijednosti za dohvaćenu datoteku (primjerice putanja mora završavati s datotekom koja ima ekstenziju .pdf) može se zaobići korištenjem takozvanih null bajtova. Oni osiguravaju da se putanja uspješno prekine na željenom mjestu, prije tražene ekstenzije. Primjerice, putanju koja mora završavati sa slikom i traženom ekstenzijom .png na njezinom kraju može se izmijeniti u oblik: *http://www.website.com/page.php?file=../../../etc/passwd00.png*.

Provjera ekstenzije bit će uspješna, a naredba će se izvesti do putanje navedene prije oznake za null bajt. Ukoliko se i ovdje pokuša primijeniti prevencija simbola, ponovno je moguće koristiti URL kodirane vrijednosti - %00 ili 0x00.

S obzirom na sve navedene opasnosti, zaštitu za ovaj napad potrebno je provesti jako pažljivo. Najjednostavniji dio prevencije je konstanto ažuriranje servera i operacijskog sustava na najnovije verzije. Nadalje, potrebno je što manje koristiti unose korisnika za dohvaćanje podataka iz sustava. Također, preporuča se spremanje web servera i datoteka operacijskog sustava na različitim lokacijama. Za putanje koje ipak sadrže korisnički unos podataka, potrebno je što bolje sanitizirati podatke. Potrebno je provjeriti postoji li tražena putanja i ima li korisnik pristup podacima u njoj te nalaze li se njezini parametri na listi dopuštenih putanja ili oznaka. Na kraju, savjetuje se provjera svih dijelova aplikacije koji rade s datotekama, odnosno dopuštaju li izravne ili kodirane pristupe podacima koji bi trebali biti zaštićeni.

3.8 Uskraćivanje resursa iskorištavanjem regularnih izraza (ReDoS)

ReDoS napadi predstavljaju oblik napada koji se temelji na složenosti algoritma validacije izraza. Njegov cilj je uskratiti dostupnost željenog resursa koristeći propuste pri dizajniranju regularnih izraza. Usluga postaje jako spora ili u potpunosti nedostupna za korisnika. Prilikom izvođenja napada, kreira se niz znakova koji je složen na način da vremenski zaokupi validaciju korisničkog unosa. Primarna zadaća regularnih izraza je zaustaviti nedozvoljeni unos, odnosno osigurati da korisnik mora poštovati određena pravila pri unosu podataka. Međutim, pri obavljanju te funkcionalnosti često otvaraju prostor za kreiranje ReDoS napada. Iako nisu očigledna ranjivost, loše dizajnirani obrasci za provjeru regularnih izraza mogu dovesti do velikih problema u aplikaciji.

Temeljna ideja ovog napada je dovesti implementaciju provjere izraza u ekstremne slučajeve, koji dovode do eksponencijalnog rasta vremena obrade. Glavni razlog tome je sam način rada regularnih izraza, odnosno postupak koji se zove *backtracking*. Validacija izraza provodi se u više koraka, gdje se za svaki dio izraza provjerava ispunjava li sve uvjete. U slučaju kada dođe do neispravnog dijela unosa, backtracking vraća validaciju na prethodnu poziciju u kojoj se mogla donijeti drugačija odluka. Ovaj postupak se ponavlja dok se ne ispitaju sve moguće kombinacije i permutacije znakova.

ReDoS napad može se promatrati i kao jednostavnija verzija brute-force napada. Kako je ranije navedeno, brute-force napadi većinom se koriste za ostvarivanje pristupa stranici na temelju velikog broja pokušaja različitih kombinacija podataka. Kao rezultat toga, stranica je opterećena i može doći do uskraćivanja resursa. Međutim, takvi pokušaji se sve češće ograničavaju – vremenski ili maksimalnim brojem pokušaja prijave. ReDoS napade je teže detektirati i spriječiti. Iskorištavaju propuste u samom dizajnu sustava i potreban je manji broj napada kako bi se prouzročilo isto opterećenje na sustav. Utjecaj ovog oblika napada detaljnije je obrađen u radu [11].

Obrasci provjere regularnih izraza koji su podložni ovim napadima najčešće su nesigurni jer sadrže ponavljanje izraza (koristeći simbole + i *) ili im određeni dio unutar izraza ispunjava uvjete provjere za cijeli izraz. Naziv za izraze koji se koriste u ReDoS napadima je *Evil Regex*, a njihovo korištenje prikazano je na primjeru 3.23.

```
//PROVJERE REGULARNOG IZRAZA PODLOŽNE NAPADU

^(a+)+$
^(a+)+[a-zA-Z]$
^([a-zA-Z]+)*
^(a|a?)+

//UNOS ZA USPJEŠAN NAPAD

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
```

Primjer 3.23. *Regularni izrazi s propustima i oblik napada koji ih iskorištava*

U prvom primjeru, izraz unutar zagrada pretražuje pojavljivanje znaka *a* i svih njegovih ponavljanja nakon toga. Nakon izvršavanja takve provjere, provjerava se je li cijeli izraz ponavljanje određenog dijela izraza. Slanjem navedenog unosa, svaki dodatni simbol *a* prolazi kroz provjeru ponavljanja samog izraza i svih njegovih dijelova, što dodatno povećava vrijeme obrade. Izrazi u kojima se koriste vrijednosti znakova a-z i A-Z mogu djelovati sigurnije, ali s obzirom da postoje određeni dijelovi izraza koji su nesigurni, cijeli izraz je nesiguran. Drugi primjer sadrži isti problem s korištenjem ponovljenog simbola na početku, dok se u trećem koristi ponavljanje cijelog izraza – također predstavlja propust ako se unese niz istih simbola. U zadnjem primjeru prihvaća se samostalan simbol *a* ili njegova kombinacija s još jednim simbolom, ali oznaka ponavljanja na kraju daje isti učinak kao prethodni primjeri.

ReDoS napadi mogu se kreirati direktno, u slučaju ne postojanja validacije korisničkog unosa. Primjer navedenog propusta je kada postoje ograničenja u unosu podataka, poput korisničkog imena i zaporke. Ukoliko korisničko ime ne smije biti sadržano unutar zaporke, ali ne postoji sanitizacija za te dvije vrijednosti, moguće je unijeti Evil Regex. Primjer 3.24. prikazuje kreiranje takvog napada.

```
username: ^(a+)+$
password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!
```

Primjer 3.24. *Kreiranje vlastitog ReDoS napada kroz propuste u validaciji*

Napadač pod korisničko ime navodi regularni izraz koji sadrži propust, a u zaporku izraz koji taj propust iskorištava. Provjera je li korisničko ime sadržano u zaporci predstavlja vremenski zahtjevan proces i potencijalni prekid izvođenja aplikacije, jer se osim programske provjere provodi i ona unesena u korisničkom imenu.

ReDoS napadi mogu uzrokovati prestanak rada aplikacije, ali i stvoriti probleme s vatrozidom i radom servera. Regularni izrazi neizostavni su dio u radu aplikacija jer sprječavaju druge napade, ali potrebno ih je kreirati ispravno. Početni korak u zaštiti od ovih napada je korištenje višenitnosti. Kada aplikacija radi na većem broju niti, ostale niti mogu obraditi zahtjeve koji su na čekanju. Nit zadužena za zlonamjeran zahtjev može uvesti vremensko ograničenje te nakon određenog perioda prekinuti proces. Također, provjere regularnih izraza ne bi trebale biti javno dostupne, jer to napadačima olakšava uvid u potencijalne propuste. Preporuča se korištenje već provjerenih regularnih izraza te razdvajanje složenih izraza u veći broj manje kompleksnih provjera. Ponavljanja, preklapanja i različite kombinacije izraza najčešći su razlog uspješnih napada. Na kraju, korisnički unosi moraju biti validirani i filtrirani, kako ne bi postojala mogućnost direktnog unosa Evil Regex izraza.

3.9 Falsificiranje zahtjeva na server (engl. Server-side request forgery - SSRF)

Falsificiranje zahtjeva na serveru je nesigurnost koja omogućava napadaču da putem zahtjeva upućenog sa servera ostvari pristup nedozvoljenoj lokaciji. Nesiguran server koristi se kao pristupna točka te se s njega šalju zahtjevi na druge sustave, kojima nije moguće pristupiti direktno. S obzirom na rast složenosti i povezanosti današnjih aplikacija, povećava se i broj zahtjeva koji se obavljaju na strani servera. Napad se izvodi na način da se pomoću navedenog servera ostvari veza s unutarnjom mrežom organizacije. Ona sadrži osobne i pristupne podatke te usluge koje su zaštićene vatrozidom i nisu dostupne vanjskoj mreži, već se koriste kao resursi unutarnjih komponenti aplikacije. Napadač može iskorištavati server kako bi dohvaćao i izmjenjivao vrijednosti resursa, mijenjao URL adrese na koje se šalju određeni podaci te saznao informacije o samoj konfiguraciji servera. Sve veće korištenje oblaka računala i njegovih usluga otvara pristup velikoj količini podataka, što povećava negativan utjecaj ovakvog oblika napada. Do navedenih napada dolazi kada korisnik ima potpunu ili djelomičnu kontrolu nad zahtjevom koji se šalje serveru, najčešće izmjenom URL putanje treće strane, na koju aplikacija šalje zahtjev.

Zahtjevi između servera uobičajeni su u radu web aplikacija. Najčešće se koriste za dohvaćanje podataka udaljenog resursa ili uključivanje metapodataka s drugih stranica. Takvi zahtjevi u unutarnjoj mreži sami po sebi nisu opasni, ali temelje se na ispravnoj upotrebi i implementaciji. Ukoliko se unos dobiven od korisnika upotrebljava za oblikovanje URL putanje, bilo da se radi o putanji aplikacije ili usluge treće strane, moguće je ostvariti pristup navedenim zaštićenim lokacijama. Primjer 3.25. objašnjava takvu izmjenu.

```
//URL ZA DOHVAĆANJE PODATAKA TREĆE STRANE
https://example.com/feed.php?url=externalsource.com/data
//IZMJENA PUTANJE
https://example.com/feed.php?url=localhost/admin
```

Primjer 3.25. *Promjena URL putanje za pristup zaštićenim lokacijama*

Aplikacija u navedenom primjeru putem vanjske mreže koristi putanju za pristup resursima unutarnje mreže. Unutarnja mreža nije javno dostupna, ali njezini resursi mogu se koristiti u radu same aplikacije. Ukoliko je *url* parametar vidljiv u kreiranju zahtjeva, napadač njegovu vrijednost može izmijeniti i postaviti na *localhost*. Ovime ostvaruje uvid u resurse spremljene na serveru, koje može koristiti za kreaciju budućih SSRF napada. Napadač može pokušati pristupiti navedenoj adresi direktno, ali takav pristup je zaštićen raznim provjerama i autentikacijom. Međutim, kada takav zahtjev dođe s lokalnog uređaja, sigurnosne provjere se ne provode jer zahtjev dolazi iz pouzdanog izvora.

Osim izmjene putanje, moguće je pristupiti željenim resursima putem IP adrese. S obzirom da većina javno nedostupnih resursa posjeduje svoju privatnu IP adresu, često izostaje autentikacija ispravnosti zahtjeva. Ponovno se iskorištava pretpostavka da je komunikacija unutarnje mreže sigurna i iz pouzdanih izvora, zbog čega posjeduje nižu razinu zaštite. Ukoliko postoji lokacija za pristup administrativnim funkcijama, a poznaje se njezina IP adresa, napad je moguće izvesti pomoću modificiranog zahtjeva, kako je prikazano u primjeru 3.26.

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://192.168.0.68/admin
```

Primjer 3.26. *Modifikacija zahtjeva za pristup zaštićenim lokacijama*

Slanje ovakvog zahtjeva zaobilazi provjere i korisniku daje sve informacije dostupne na navedenoj lokaciji. Ukoliko aplikacije blokiraju korištenje određenih adresa u putanji, poput localhosta i 127.0.0.1, moguće je koristiti drugačije oblike zapise. Ovakve zaštite mogu se izbjeći korištenjem alternativne verzije IP adrese ili kodirajući njezinu vrijednost u URL simbole. Također, moguće je kreirati vlastitu domenu koja nije na popisu zabranjenih

vrijednosti, ali automatski preusmjerava na ranije navedene adrese. U suprotnoj implementaciji, kada server dopušta zahtjeve samo s određenog popisa vrijednosti, moguće se služiti istim metodama. Potrebno je napraviti određene izmjene koje su djelomično zahtjevne, poput ugrađivanja zabranjene vrijednosti unutar valjane ili iskorištavanja DNS načina imenovanja, ali konačni cilj i princip izvršavanja napada ostaje isti.

Slična verzija prijašnjih napada je korištenje preusmjeravanja. U slučaju da postoje određene zaštite za URL putanju i izmjene nisu dopuštene, potencijalna opasnost je preusmjeravanje putanje nakon uspješne validacije. Potrebno je kreirati URL koji ispunjava potrebne uvjete, ali na kraju sadrži adresu željenog resursa. Izmjena je pokazana primjerom 3.27.

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://website.net/product/productId=6&path=http://192.168.0.68/admin
```

Primjer 3.27. *Dodavanje parametra u putanju za pristup nedozvoljenim resursima*

Navedeni zahtjev rezultira preusmjeravanjem na navedenu IP adresu i stranicu administratora. Ovakav napad djeluje iz razloga što početni dio putanje doista pripada navedenom API-ju te je navedena domena dozvoljena. Nakon toga, zahtjev odrađuje preusmjeravanje na zaštićeni resurs u unutarnjoj mreži. S obzirom da je početna validacija zadovoljena, daljnjih provjera nema i napadač ima potpuni pristup privatnim lokacijama.

Nakon uvida u strukturu mreže, moguće je napasti servere ili resurse zaobilazeći vatrozid koji ih štiti. Maliciozan zahtjev šalje se na server koji je ranjiv na SSRF napade te se putem unutarnje mreže prosljeđuje na ciljani server. Ovim postupkom je izbjegnuta reakcija vatrozida, jer on štiti samo od vanjskih prijetnji. Ciljani server šalje odgovor serveru kojem napadač ima pristup te on dobiva željene informacije.

Osim navedenih opasnosti, podatke je moguće dohvatiti i s oblaka računala. Većina servisa koja pružaju usluge oblaka računala imaju definirano REST sučelje na određenom mrežnom priključku. Ukoliko napadač uspije pristupiti toj adresi, dobiva uvid u razne informacije. Primjerice, AWS servis pruža sučelje na adresi *http://169.254.169.254/*, na kojem se nalaze konfiguracijske datoteke i određeni autentikacijski ključevi.

Ukoliko napadač kontrolira odredište zahtjeva, može zaobići veliki broj zaštitnih mehanizama. Najčešći rezultat ovih napada je pristup zaštićenim informacijama, ali posljedice mogu biti puno veće u kombinaciji s drugim napadima. Uspješan SSRF napad:

- iskorištava povjerenje u valjanost zahtjeva koji se šalju u unutarnjoj mreži
- izbjegava provjere popisa dopuštenih IP adresa
- izbjegava autentikaciju usluga s obzirom na web hosta
- pristupa resursima i konfiguracijama koje nisu javno dostupne
- pristupa podacima s web servera (korištenje *file://* umjesto *https://* unutar URL putanje)
- skenira mrežne priključke na unutarnjoj mreži
- prikuplja IP adrese koje se koriste na serveru

Ne postoji jedinstveni pristup za rješavanje SSRF napada. Razlog tome je što se aplikacije implementiraju na različite načine i u različite svrhe. Također, nemoguće je navesti sve moguće varijante prijetnji. Napadač se uvijek može osloniti na drugačiji oblik zapisa dijelova putanje ili propuste u samoj validaciji. Trenutno popularni i učinkoviti načini prevencije dostupni su u radu [12]. Preporuča se izbjegavanje korisničkih unosa pri formiranju URL putanje, čak i u slučajevima kada je implementirana validacija. Veliki broj prijetnji može se riješiti navođenjem pouzdanih IP adresa i imena hostova (DNS nazivi adrese). Na ovaj način je lakše validirati zahtjeve i automatski odbaciti one koji ne dolaze s odobrenih adresa. Također, potrebno je onemogućiti nepotrebne URL izraze. Ukoliko aplikacija radi isključivo na HTTP i HTTPS protokolu, izrazi poput *file://* i *ftp://* ne smiju biti dostupni, kako bi se napadaču smanjile potencijalne opcije. Na kraju, preporuča se implementacija sigurnosnih provjera i autentikacije na unutarnjoj mreži. Iako se u većini slučajeva radi o pouzdanim zahtjevima i primjena ovog načina rada može otežati korištenje legitimnom korisniku, povećava se sigurnost aplikacije i eliminira se najveća prednost koju posjeduje napadač.

3.10 Udaljeno izvođenje programskog koda (engl. Remote code execution - RCE)

Udaljeno izvođenje programskog koda je napad u kojem se ostvaruje pristup naredbama na udaljenom računalu ili serveru, neovisno o njegovoj fizičkoj lokaciji. Napadi se mogu izvoditi preko lokalne mreže, mreže širokog područja ili cijelog interneta. Do samih napada najčešće dolazi kada korisnik uređaja preuzme zlonamjeran program pri korištenju web aplikacije, poslan od strane napadača. Ovaj oblik napada omogućava izvođenje malicioznog koda bez

direktnog pristupa uređaju te njegova ozbiljnost može varirati od zlonamjernih skripti do preuzimanja potpune kontrole nad uređajem.

Pronalazak nesigurnosti prvi je korak RCE napada. Osoba koja izvodi napad pregledava sva računala u mreži (ili na internetu, ovisi na kojoj razini se napad izvodi) i traži nesigurnosti koje mogu biti iskorištene. To se može odnositi na obranu same mreže, sigurnosni propust u web aplikaciji ili previd u implementaciji operacijskog sustava. Nakon pronalaska nezaštićenog uređaja ili propusta, napadač ostvaruje pristup na udaljeno računalo i izvodi programski kod. Maliciozna skripta razlikovat će se s obzirom na cilj napada, ovisno dohvaćaju li se datoteke s korisnikovog računala ili se traže informacije o bankovnom računu te je li cilj onemogućiti korisniku pristup uslugama ili samo nadzirati njegovo korištenje. Maliciozni kod kreira se u programskom jeziku korištenom za izvođenje aplikacije preko koje se ostvaruje pristup. Server interpretira navedeni kod i izvodi ga u navedenom jeziku – najčešće PHP, Python i Java. Funkcije ranjive na ovaj oblik napada te dodatni primjeri dostupni su u literaturi [13]. Kao i u većini ranije objašnjenih napada, glavni propust je izostanak ili loše provedena validacija korisničkog unosa. Maliciozan kod se nakon neuspjele validacije šalje serveru na izvođenje, čime započinje napad. Ukoliko takav propust postoji na aplikaciji s velikim brojem korisnika, uspješnost napada i njegove posljedice značajno rastu. Napad je uglavnom pisan u obliku naredbi ili skripti koje se izvode u terminalu. Skripte namijenjene izvođenju u terminalu sadrže veći broj naredbi, koje se pri njezinom pokretanju izvode slijedno. Nakon što je skripta učitana u aplikaciju, prilikom interpretacije se pokreće njezino izvođenje.

Kako je ranije navedeno, RCE napadi mogu se značajno razlikovati, ovisno o tome koji propust se iskorištava i što je cilj napada. Tri najčešće korištena oblika su: napad korištenjem deserijalizacije, napad preopterećivanjem međuspremnika i napad korištenjem neispravnog tipa podatka.

Serijalizacija podataka predstavlja proces u kojem se složene strukture podataka, poput polja ili objekata, pretvaraju u jasniji oblik koji se može prenositi mrežom u nizu podataka. Kada se podaci prenesu na željeno odredište, procesom deserijalizacije ponovno se spremaju u početan oblik. Cilj napadača je izmijeniti serijalizirani niz podataka tako da u njega ubaci maliciozan kod. S obzirom da se takvi podaci šalju u obliku niza znakova, postoji mogućnosti da se određeni dio prilikom deserijalizacije interpretira kao naredba koja se može izvesti.

Preopterećivanje međuspremnika nije usko vezan za RCE napad, već je često korišten način iskorištavanja propusta. Temelji se na propustu koji omogućava napadaču izmjenu vrijednosti podataka koje su spremljene u memoriji. Osim što se može koristiti za brisanje podataka ili opterećivanje uređaja, u RCE napadima služi za umetanje malicioznog koda na uređaj. S obzirom da se programski kod u izvođenju također sprema unutar memorije, spremanje malicioznog koda na ispravnu lokaciju osigurava njegovo pokretanje bez znanja korisnika.

S obzirom da veliki broj aplikacija koristi unesene podatke u funkcijama programskog koda, postoji šansa za kreiranje RCE napada. Prilikom izvođenja programskog koda, potrebno je potvrditi da su objekti koji se koriste ispravnog tipa. Ukoliko izostane takva provjera, napadač može umetnuti neispravne unose koji osiguravaju da se jedan dio njih interpretira kao naredba namijenjena za izvođenje.

Unatoč svim navedenim ciljevima izvođenja RCE napada, do naglog porasta njegove upotrebe dolazi iz novijeg razloga – rudarenje podataka. Rudarenje podataka i kriptovaluta doživjelo je značajni porast unazad nekoliko godina i otvara velike mogućnosti za ostvarivanje profita. Uspostavom kontrole nad udaljenim uređajima, napadač može iskoristiti tuđe resurse za obradu podataka. Iako ovakav napad nema za cilj prikupiti korisnikove podatke, ima značajan utjecaj na trošenje energije i računalnog sklopovlja.

Zaštita od ovakvih napada poprilično je općenita i teško je direktno utjecati na samu prijetnju. Napadi su izrazito raznovrsni i mogu doći iz različitih izvora. Najvažniji korak u prevenciji napada je konstanto ažuriranje sigurnosti. Iako je stalno obnavljanje programa i sklopovlja zahtjevno i popraćeno vremenskim periodom u kojem se ne pruža usluga, njime se osigurava korištenje najnovijih postupaka u sprječavanju uspješnih napada. Preporuča se korištenje vatrozida i redovito skeniranje mreže, kako bi se detektirale potencijalne prijetnje i spriječilo izvođenje detektiranih napada. Također, stalno nadziranje sustava povećava mogućnost uočavanja anomalija u njegovom radu i potencijalnih prijetnji. Potrebno je imati pripremljeno rješenje u slučaju detekcije napada, kako bi se napad što prije okončao i sustav ponovno osposobio za rad. Preporuča se korištenje softverskih rješenja koja onemogućavaju preopterećivanje memorije te detaljna validacija korisničkog unosa prije upotrebe.

4 RAZVOJNO OKRUŽENJE I FUNKCIONALNI ZAHTJEVI

U ovom poglavlju navest će se razvojno okruženje u kojem je aplikacija kreirana i alati korišteni pri njezinoj realizaciji. Također, navest će se sve funkcionalnosti web aplikacije za pohranu i dijeljenje datoteka. Funkcionalnosti će se razlikovati s obzirom na razinu pristupa koju sadrži korisnik, a njihova implementacija bit će prikazana u idućem poglavlju.

4.1 Korišteni alati i tehnologije

4.1.1 Razvojni okvir Laravel

Razvojni okvir Laravel temeljen je na programskom jeziku PHP. Osnovna zadaća mu je olakšati kreiranje web aplikacija korištenjem razumljive sintakse i organiziranjem koda u obliku obrasca MVC (engl. Model-View-Controller). Model predstavlja dio aplikacije koji sadrži poslovnu logiku, pogled sadrži korisničko sučelje i sadržaj koje će se prikazati korisniku, dok kontroler povezuje te dvije komponente i omogućava njihovu komunikaciju. Laravel povećava skalabilnost aplikacije i olakšava njezino testiranje te kreira strukturu projekta koja odvaja javne i privatne resurse aplikacije. Također, olakšava korištenje PHP naredbi pri kreiranju sadržaja na HTML stranici pomoću Blade predloška. Uz jednostavne oznake, moguće je ispitivati uvjete i prolaziti kroz petlje bez pisanja potpune sintakse na stranici. Više detalja dostupno je u službenoj dokumentaciji [14].

4.1.2 Composer

Composer je alat koji se koristi unutar Laravela, a služi za upravljanje programskim bibliotekama i ovisnostima programa. Olakšava instaliranje vanjskih biblioteka te ažuriranje zahtjeva i komponenata potrebnih za normalan rad aplikacije.

4.1.3 Artisan

Artisan je sučelje naredbenog retka uključeno u Laravel. Omogućava korištenje jednostavnih naredbi u komandom prozoru, koje olakšavaju migraciju podataka u bazu, pokretanje aplikacije, kreiranje komponenata poput modela ili kontrolera, upravljanje priručnom memorijom i slično.

4.1.4 Bootstrap

Boostrap je CSS okvir otvorenog koda, koji se koristi za stvaranje responzivnih stranica te olakšava njihov dizajn. Izrađen je koristeći HTML, CSS i JavaScript, što omogućava prilagodbu sadržaja stranice ovisno o veličini i rezoluciji uređaja.

4.2 Funkcionalni zahtjevi na web aplikaciju

Prije izrade aplikacije, moraju se definirati zahtjevi za njezino korištenje. Aplikacija će sadržavati dva tipa korisnika – studenta i administratora. Administrator će imati sve ovlasti koje ima i običan korisnik, uz određene dodatke. Zahtjevi obje vrste korisnika dani su u nastavku.

1. Zahtjevi studenta (običnog korisnika):

- Korisnik se može registrirati
- Korisnik se može prijaviti u aplikaciju pomoću email adrese i zaporke
- Korisnik može promijeniti zaporku, ukoliko ju je zaboravio, putem email adrese na koju dobiva poveznicu za oporavak
- Korisnik se može odjaviti iz aplikacije
- Korisnik može pohranjivati datoteke u aplikaciju i dodati opis
- Korisnik može odlučiti je li datoteka javno dostupna ili ne
- Korisnik može mijenjati razinu privatnosti svojih datoteka
- Korisnik može obrisati svoje datoteku
- Korisnik može vidjeti i preuzeti javne datoteke ostalih korisnika
- Korisnik može dijeliti svoje privatne datoteke s ciljanim korisnicima putem njihove email adrese, slanjem poveznice za preuzimanje
- Korisnik može ostaviti komentar vezano uz javnu datoteku drugog korisnika
- Korisnik može „ocijeniti“ datoteku pomoću Like/Upvote funkcionalnosti
- Korisnik može sortirati javno prikazane datoteke s obzirom na naziv datoteke, broj pozitivnih ocjena, vrijeme nastajanja, email autora i veličinu datoteke
- Korisnik može filtrirati prikazane javne datoteke na zaslonu unoseći željeno ime datoteke u tražilicu
- Korisnik može pretraživati javno dostupne datoteke s obzirom na predmet, godinu i smjer studija te naziv datoteke i email adresu autora

2. Dodatni zahtjevi administratora:

- Administrator može vidjeti popis svih korisnika
- Administrator može promijeniti razinu pristupa svakog korisnika
- Administrator može obrisati korisnika
- Administrator može vidjeti broj pohranjenih datoteka svakog korisnika

- Administrator može obrisati bilo koju javno dostupnu datoteku ili komentar
- Administrator može vidjeti statističke informacije (broj korisnika, broj javnih i privatnih datoteka)

4.3 Ostale postavke projekta

Projekt će se implementirati na Linux operacijskom sustavu, koristeći LAMP okruženje. Za bazu podataka pri razvoju aplikacije koristit će se phpMyAdmin lokalna baza te će se aplikacija pokretati na lokalnom serveru. Baza će sadržavati tablice za korisnike, datoteke, dobivene ocjene i komentare datoteka te tablicu fakultetskih kolegija s informacijama o svakom od njih (na kojoj godini se izvodi, za koje smjerove i slično).

5 REALIZACIJA ZADATKA

U ovom poglavlju prikazat će se programsko rješenje zadatka. Objasnit će se poduzete mjere za sprječavanje svakog od ranije opisanih napada i određeni dodatni koraci koji se mogu upotrijebiti u budućnosti. Također, prikazat će se korištenje same aplikacije i njezinih funkcionalnosti navedenih u prethodnom poglavlju.

5.1 Prevencija sigurnosnih prijetnji

Osim ranije navedenih izvora, za realizaciju sigurnosnog segmenta programskog rješenja korišteni su izvori [15] i [16].

5.1.1 Prevencija SQL injection napada

SQL injection prijetnje moguće je minimizirati ispravnim korištenjem Laravel razvojnog okvira. Većina metoda koje su zadužene za komunikaciju s bazom podataka već sadrži ugrađene prevencije za ovaj oblik napada. Metode za izgradnju upita na bazu koriste PDO parametriziranje – način rada objašnjen na primjeru 3.7. Ovim postupkom se prvo kreira željeni upit na bazu, a tek nakon toga se dodaju parametri, čime se osigurava da korisnik svojim unosom ne može promijeniti inicijalan upit. Ovaj način rada primjenjuje se na metode za izgradnju upita na bazu te na metode koje koriste Eloquent modeli unutar aplikacije - modeli koji predstavljaju tablice iz baze podataka i olakšavaju interakciju s njima. Pri izradi ove aplikacije korišteni su Eloquent modeli, jer olakšavaju upotrebu Eager loadinga. Eager loading je koncept u kojem se uz određeni model učitavaju svi ostali modeli koji su s njime povezani. Primjerice, pri dohvaćanju korisnika aplikacije, automatski se dohvaćaju i sve datoteke koje mu pripadaju. Isto se može primijeniti i za dohvaćanje datoteka, pri čemu bi se istovremeno povukle sve informacije vezane uz kolegij kojem datoteka pripada. Ovim načinom rada rješava se problem N+1 upita, tako što se broj upita koji se izvršava na bazu minimizira te se u manje upita učitava veća količina podataka. Navedeni koncept je teže kreirati putem metode za izgradnju upita, zbog čega se ona uglavnom ne koristi u ovom radu. Njezina upotreba se preporuča u situacijama kada je potrebno poboljšati performanse aplikacije, jer se njezini upiti izvode brže od onih kreiranih Eloquent modelom. Korištenje navedenih metoda za komunikaciju s bazom prikazani su na primjeru 5.1.

```
//KORIŠTENJE ELOQUENT MODELA
$query = File::with('user:id,email', 'subject:id,subject_name')->where('is_public', 1)
->where('user_file_name', 'LIKE', '%' . $request->user_file_name . '%')
->where('subject_id', $request->subject_id)
// KORIŠTENJE IZGRADNJE UPITA
$statisticInfo = DB::select('SELECT COUNT(DISTINCT users.id) AS users, (SELECT COUNT(files.id)
FROM files WHERE  files.is_public = 1) AS public_files, (SELECT COUNT(files.id) FROM files WHERE
files.is_public = 0) AS private_files FROM users, files');
```

Primjer 5.1. *Korištenje Eloquent modela i izgradnje upita*

U navedenom primjeru prikazano je korištenje Eloquent modela u aplikaciji. Korišteni upit je pojednostavljen zbog lakšeg prikaza, a cijela funkcija dostupna je unutar datoteke DatabaseAction.php, pod nazivom *buildSearchQuery()*. U toj funkciji upotrebljava se unos korisnika kako bi se pronašle datoteke koje ispunjavaju njegove uvjete pretrage. S obzirom da je cilj ne ograničiti kreiranje upita korisnika, izostavlja se sanitizacija ulaza. Zbog toga je izrazito bitno koristiti sigurne metode pri kreaciji upita. Metode poput *select()*, *where()*, *orderBy()* i drugih automatski provode parametrizaciju izjava te prvo kreiraju upit, a zatim dodaju parametre dobivene od korisnika. Ovime se osigurava da se potencijalno unošenje malicioznog programskog koda ne izvršava, već da ga se tretira kao normalan unos. Na isti način su osigurane i metode koje se koristi prilikom izgradnje upita. Drugi dio primjera prikazuje korištenje *select()* metode za dohvaćanje statističkih informacija. U ovom slučaju nema prijetnje za SQLi napad jer se ne koristi nikakav korisnički unos pri izgradnji upita, ali *select()* metoda svakako pripada pod zaštićene metode.

Jedina potencijalna opasnost za izvođenje SQLi napada je korištenje DB::raw ili DB::statement metode. Navedene metode koriste se kada programer želi kreirati vlastiti upit, koji je u većini slučajeva poprilično složen i teško ga je kreirati uz ranije navedene metode. Najčešća primjena je za grupiranje podataka nakon COUNT() ili AVG() funkcije, filtriranje podataka po godinama umjesto datumima, korištenje CASE() funkcije i slično. Prilikom korištenja takvih izraza, preporuča se ne dodavati informacije unesene od strane korisnika. Ukoliko to nije moguće izbjeći, potrebno je koristiti alternativne metode, poput *selectRaw()* i *whereRaw()* te provesti parametrizaciju korisničkih unosa. Prilikom izrade aplikacije, DB::raw metoda korištena je minimalno te bez dodavanja korisničkog unosa. Prikaz upotrebe

u aplikaciji, kao i prikaz njezine sigurne upotrebe pri dodavanju korisničkog unosa dani su na primjeru 5.2.

```
// KORIŠTENJE DB::RAW METODE U APLIKACIJI
$query = File::with('user:id,email', 'subject:id,subject_name')
        ->join('subjects', 'files.subject_id', '=', 'subjects.id')
        ->join('users', 'files.user_id', '=', 'users.id')
        ->select(DB::raw('files.id AS file_id, files.*, subjects.*, users.email'))
        ->where('is_public', 1);
// PRIMJER SIGURNOG DODAVANJA KORISNIČKOG UNOSA
DB::table('orders')
            ->whereRaw('price > ?)', [$request->price])
            ->get()
```

Primjer 5.2. *Korištenje DB::raw metode*

Pri izradi aplikacije, DB::raw metoda koristi se u ranije navedenoj funkciji *buildSearchQuery()*, kako bi se dodijelio alias jednom od stupaca. S obzirom da se povezuju tri tablice pri kreiranju upita, dolazi do preklapanja vrijednosti za stupac *id*. Pomoću *raw()* metode se osigurava pamćenje vrijednosti identifikatora datoteke, koja se kasnije koristi pri kreiranju sadržaja na korisničkom sučelju. Nakon toga, dan je primjer korištenja *whereRaw()* metode, u kojoj se upitnik unutar izjave zamjenjuje s vrijednosti predanom u dodatnom parametru metode. Moguće je dodavati veći broj elemenata, ali se mora paziti na njihov redoslijed.

Kao posljednji korak prevencije, preporuča se postavljanje vrijednosti APP_DEBUG unutar .env datoteke na false prilikom objavljivanja aplikacije. Ukoliko dođe do greške, a ta opcija je uključena, moguće je dobiti uvid u strukturu same baze podataka te upita koji se izvodi. Time se znatno olakšava kreiranje i izvođenje napada na bazu.

5.1.2 Prevencija probijanja autentikacije

Za sprječavanje probijanja autentikacije, glavni fokus stavljen je na zaštitu zaporki. Prilikom registracije korisnika, unesena zaporka validira se kroz određeni niz pravila. Mora sadržavati minimalno 8 znakova i to u kombinaciji velikih i malih slova, brojeva i simbola. Također, provjerava se je li unesena zaporka na popisu zaporki za koje je poznato da su prikupljene u ranijim napadima. Provjera se služi pozivom metode s Have I Been Pwned API-ja, koji sadrži

navedeni popis i pretražuje nalazi li se zaporka u njemu. Nadalje, prilikom spremanja zaporke u bazu koristi se Hash funkcija ugrađena u Laravel, koja od navedene zaporke kreira nasumičan niz znakova. Ovime se osigurava da u slučaju ostvarivanja neovlaštenog pristupa bazi podataka napadač nema jasan prikaz pristupnih podataka. Na kraju, ograničava se broj mogućih prijava korisnika, kako bi se spriječili automatizirani napadi. To se ostvaruje dodavanjem posredničkog sloja (engl. middleware) na putanju zaduženu za obradu zahtjeva prilikom prijave. Navedeni sigurnosni koraci prikazani su na primjer 5.3. Posrednički sloj zadužen za prijavu postavlja ograničenje na 8 zahtjeva za prijavu unutar 2 minute. Navedene vrijednosti moguće je mijenjati s obzirom na vrstu aplikacije i razinu sigurnosti koju zahtjeva. Isto ograničenje može se koristiti i na drugim putanjama, kako bi se smanjila uspješnost DDoS napada ili smanjilo opterećenje na aplikaciju.

```
// PRAVILO ZA VALIDACIJU ZAPORKE PRILIKOM REGISTRACIJE
'password' => ['required', 'confirmed', Password::min(8)->mixedCase()->letters()->numbers()->symbols()->uncompromised()]
// HASHIRANJE ZAPORKE PRILIKOM SPREMANJA U BAZU
    User::create([
      'first_name' => $request->first_name,
      'last_name' => $request->last_name,
      'email' => $request->email,
      'password' => Hash::make($request->password),
      'is_admin' => 0,
    ]);
// OGRANIČAVANJE BROJA POKUŠAJA NA PUTANJI ZA PRIJAVU KORISNIKA
Route::post('custom-login', [AuthController::class, 'customLogin'])->middleware("throttle:8,2");
```

Primjer 5.3. *Prevencija probijanja autentikacije pomoću zaporke*

Potencijalne prijetnje vezane uz ostvarivanje pristupa sesijama uglavnom se ne otklanjaju u programskom rješenju. Glavni korak je njihovo izostavljanje u putanjama aplikacije, kao i korištenje složenih vrijednosti za sesije, generiranih od strane Laravela. Validacija sesija na temelju IP adrese izbjegava se zbog mogućnosti dinamičke promjene njezine vrijednosti i vraćanja netočnih rezultata provjere. U slučaju slanja sesija, preporuča se odobravanje HTTPS metoda za sve zahtjeve, kako bi se osigurala enkripcija podataka. Upravljanje trajanjem sesije,

enkripcijom prije slanja i sličnim informacijama moguće je unutar config/session.php datoteke.

Nakon uspješne prijave korisnika, generiraju se nove vrijednosti za sesije, čime se izbjegava njihovo fiksiranje, spomenuto u poglavlju 3.2. Ovim postupkom osigurava se da napadač ne može iskoristiti sesije korisnika prije prijave u aplikaciju za lažno predstavljanje. Prikaz generiranja novih sesija i određene vrijednosti iz session.php datoteke korisne za prevenciju napada nalaze se na primjeru 5.4.

```
// GENERIRANJE NOVIH SESIJA NAKON USPJEŠNE PRIJAVE
  if (Auth::attempt($credentials)) {
        $request->session()->regenerate();
        return redirect("/");
   }
// KORISNE VRIJEDNOSTI UNUTAR session.php DATOTEKE
'lifetime' => env('SESSION_LIFETIME', 120),       // TRAJANJE SESIJE U MINUTAMA
'expire_on_close' => false,        // BRISANJE SESIJE ODMAH NAKON ZATVARANJA PREGLEDNIKA
'encrypt' => false, // KRIPTIRANJE VRIJEDNOSTI PRILIKOM SPREMANJA U SESIJU
'http_only' => true, // OSIGURAVA DA JE KOLAČIĆ SESIJE DOSTUPAN SAMO U HTTP METODAMA
```

Primjer 5.4. *Prevencija probijanja autentikacije pomoću sesije*

Sigurnost po pitanju probijanja autentikacije može se dodatno povećati korištenjem autentikacije korisnika u više koraka ili implementacijom CAPTCHA autentikacije na putanjama koje predstavljaju rizik za brute-force napade.

5.1.3 Prevencija XSS napada

XSS napadi problem su vezan uz prikaz podataka. Samo spremanje zlonamjernog oblika programskog koda u bazu podataka ne predstavlja prijetnju ako se taj programski kod ne može izvršiti. Laravel sprječava ovaj oblik napada pomoću Blade predloška za prikaz sadržaja stranice. Sav sadržaj napisan unutar oznake „{{ }}“ interpretirat će se kao običan tekst, bez mogućnosti ikakvog izvođenja funkcionalnosti. Unatoč tome, dobra je praksa provoditi sanitizaciju i validaciju ulaza gdje god je to moguće. U ovom programskom rješenju, većina korisničkog unosa prolazila je kroz provjeru niza pravila koje mora zadovoljiti. S obzirom da je u određenim slučajevima potrebno ostaviti slobodu korisniku za unos, poput primjerice pisanja komentara, sama validacija nije dovoljna za potpunu sigurnost. Radi toga se sav

sadržaj iz baze podataka na stranici prikazuje pomoću {{ }} operatora. Primjeri različitih validacija ulaza i korištenja navedenog operatora prikazani su na primjeru 5.5.

```
// SLOŽENIJE PROVJERE KORISNIČKOG UNOSA
'first_name' => ['required', 'regex:/^[a-ŽA-Ž. -]+$/', 'max:50'],
'subject_id' => 'required|not_in:0|exists:subjects,id',
'email' => ['required', 'email', 'max:100', 'unique:users',
            'regex:/^[A-Ža-Ž0-9\.]*@(ferit|etfos).hr$/'],
// OGRANIČENA PROVJERA UNOSA ZA KOMENTAR
'comment' => 'required|max:1000',
// KORIŠTENJE BLADE OPERATORA ZA PRIKAZ PODATAKA IZ BAZE
<p class="card-text mt-2"> {{$comment->comment_text}} </p>
<td class="table-text">
    <div> {{ $file->subject->subject_name}} </div>
</td>
```

Primjer 5.5. *Validacije različitih unosa i ispis vrijednosti iz baze na stranicu*

Osim navedenih opcija u Laravelu, moguće je koristiti PHP funkcije. Savjetuje se korištenje *htmlspecialchars()* funkcije, koja određene simbole iz korisničkog unosa kodira u HTML entitete. Primjerice, simboli &, < i > interpretirat će se kao vrijednosti &amp, &lt, &gt. Ovom funkcijom također se osigurava da se korisnički unos tumači kao običan tekst. Također, moguće je koristiti *strip_tags()* funkcija, koja uklanja HTML oznake iz varijable. Primjena navedenih funkcija dana je primjerom 5.6.

```
// PHP FUNKCIJE
$text = "<h1> This is some <b>bold</b> header</h1>.";
echo htmlspecialchars($str);     // ISPIS: <h1> This is some <b>bold</b> header</h1>.
echo strip_tags($str);    // ISPIS: This is some bold header.
```

Primjer 5.6. *Korištenje PHP funkcija za prevenciju zlonamjernog unosa*

5.1.4 Prevencija CSRF prijetnje

Kako je ranije objašnjeno, CSRF napadi mogu se izvesti bez korisnikovog znanja te napadaču omogućiti pristup korisnikovim informacijama. S obzirom da aplikacija u ovom radu sadrži različite putanje za objavljivanje novih datoteka i njihovo ažuriranje, izmjenu razine pristupa

korisnika i slično, postoji velik broj potencijalnih prijetnji za ovaj oblik napada. Kako bi se spriječile potencijalne opasnosti, potrebno je za svaki POST, PUT i DELETE zahtjev provjeriti sadrži li u sebi tajnu vrijednost iz sesije, kojoj maliciozna stranica ne može pristupiti. Unutar Laravel razvojnog okvira je navedena implementacija poprilično jednostavna. Laravel automatski generira CSRF token za svaku aktivnu sesiju prijavljenog korisnika. Navedenim tokenom provjerava se je li korisnik koji šalje navedeni zahtjev zaista prijavljeni korisnik. S obzirom da je token pohranjen unutar sesije korisnika te se automatski mijenja svakom izmjenom sesije, napadač mu ne može direktno pristupiti. Unutar svake HTML forme koja sadrži POST, PUT ili DELETE metodu, potrebno je dodati oznaku za CSRF token. Moguće ga je dodati Blade oznakom ili koristeći {{ }} operator. Ukoliko token nije postavljen unutar forme ili se njegova vrijednost ne poklapa s vrijednošću generiranom za prijavljenog korisnika, stranica će vratiti poruku s greškom „419 – Page Expired“. Prikaz njegovog korištenja unutar programskom rješenja dan je na primjeru 5.7., unutar kojeg se nalazi forma za registraciju novog korisnika aplikacije. Poruke koje se ispisuju u slučaju neispravne validacije izostavljene su zbog bolje preglednosti koda. Korištenjem prikazane oznake, prilikom svakog otvaranja stranice generira se novi i jedinstveni token. Kada se šalje zahtjev, aplikacija vrši provjeru pomoću VerifyCsrfToken.php middlewarea. Ovime se osigurava da je token postavljen unutar zahtjeva jednak onom koji je generiran za trenutno prijavljenog korisnika. U određenim slučajevima kada se ne radi putem HTML forme, primjerice korištenje Ajax poziva, moguće je pohraniti token u zaglavlju. Oznaka <meta name="csrf-token" content="{{ csrf_token() }}"> unutar zaglavlja stranice imat će jednak učinak kao ranije navedeni zapis unutar forme.

U određenim slučajevima potrebno je isključiti CSRF validaciju za jednu ili više putanja. Primjerice, za određene putanje unutar API-ja može se zanemariti CSRF validacija te umjesto nje koristiti drugačiji oblici provjere korisnika i zahtjeva. Kako bi se to ostvarilo, potrebno je unutar VerifyCsrfToken.php navesti rute koje se isključuju iz provjere, s obzirom da se ona automatski provodi na svim rutama. S obzirom da Laravel zahtjeva korištenje navedenih oblika prevencije kako bi se aplikacija mogla koristiti, ostvarivanje pristupa korisničkim informacijama ili njihova izmjena putem CSRF napada svedena je na minimum.

```
// PRIKAZ FORME ZA REGISTRACIJU NOVOG KORISNIKA

<form action="{{ route('register.custom') }}" method="POST">
      @csrf // ili {{ csrf_field() }}

     // ekvivalent <input type="hidden" name="_token" value="{{ csrf_token() }}" />


      <div class="form-group mb-3">
         <input type="text" placeholder="First Name" id="first_name"

                  class="form-control" name="first_name" value="{{old('first_name')}}"

                  required autofocus>
      </div>

      <div class="form-group mb-3">
         <input type="text" placeholder="Last Name" id="last_name"

                   class="form-control" name="last_name" value="{{old('last_name')}}"

                  required autofocus>
       </div>

       <div class="form-group mb-3">
          <input type="text" placeholder="Email" id="email_address" class="form-control"
                 name="email" value="{{old('email')}}" required autofocus>
       </div>

       <div class="form-group mb-3">
          <input type="password" placeholder="Password" id="password"

                   class="form-control" name="password" required>
      </div>

      <div class="form-group mb-3">
          <input type="password" placeholder="Confirm Password"

                  id="password_confirmation" class="form-control"
                  name="password_confirmation" required>
     </div>

    <div class="d-grid mx-auto">
          <button type="submit" class="btn btn-dark btn-block">Register</button>
    </div>

</form>
```

Primjer 5.7. *Korištenje CSRF tokena unutar forme*

5.1.5 Prevencija propusta u kontroli pristupa

Najčešći propust koji dozvoljava provođenje ovog oblika napada je prikazivanje jedinstvenog identifikatora unutar URL putanje. Ovime se dozvoljava uvid u unutarnju strukturu baze podataka, čime se povećava rizik za buduće napade. Zbog toga se unutar aplikacije koristi jedinstveno ime datoteke, generirano koristeći UUID funkciju. Navedena funkcija kreira univerzalni jedinstveni niz znakova za svaku od datoteka, što omogućava korištenje tog niza unutar URL-a (ne postoji šansa za konflikt i postojanje više datoteka s istim imenom). Unatoč toga, potrebno je implementirati dodatne provjere na backend strani aplikacije. Prvo, potrebno je osigurati putanje koje zahtijevaju da je korisnik prijavljen u aplikaciji. Također, potrebno je zaštiti putanje namijenjene za administratora. Običan korisnik im ne smije moći pristupiti te je potrebno zaštiti da mu se navedene opcije ne prikazuju na stranici (koristeći Blade metode), ali i da je pristup zabranjen direktnim unosom putanje. Nadalje, na stranici zaduženoj za prikaz detalja mora se osigurati da je nemoguće pristupiti privatnim datotekama manipulacijom URL putanje. Isto vrijedi i za putanje namijenjene za preuzimanje datoteka. Navedene provjere provode se unutar kreiranih datoteka za posrednički sloj ili middleware: CheckAdmin.php, CheckAdminOrOwner.php, CheckOwner.php i CheckOwnerOrPublic.php. Nakon toga se odgovarajuće provjere dodaju na željenu rutu, čime se osigurava njihovo provođenje prije pokretanja funkcionalnosti. Izgled jedne od middleware datoteka te njezina primjena na putanju, kao i ograničavanje funkcionalnosti na stranici pomoću Blade metoda prikazani su na primjeru 5.8. Prikazana je provjera koja se koristi za preuzimanje datoteka. Iz putanje se uzimaju informacije iz baze podataka o datoteci koja se preuzima. U slučaju da je datoteka javna ili datoteci pristupa njezin vlasnik, omogućava se daljnje izvođenje namijenjene funkcionalnosti. U suprotnom se na stranici prikazuje poruka s greškom o neovlaštenom pristupu. Navedeni middleware se registrira unutar Kernel.php datoteke, čime se omogućava njegovo korištenje na putanjama. Za upotrebu registrirane provjere, potrebno je na putanji primijeniti metodu *middleware()* i navesti koja provjera se želi provesti.

Također, prikazano je korištenje generiranog UUID-a u putanji. Prilikom navođenja putanje, naglašava se da se u putanji želi ispisati generirano ime datoteke umjesto jedinstvenog identifikatora iz baze podataka. Nadalje, prikazana je provjera unutar same strukture stranice, gdje se osigurava da su poveznice za putanje administratora ne prikazuju ako prijavljeni korisnik nema navedene ovlasti. Unutar Blade datoteke moguće je koristiti već ugrađene provjere za prijavljenog i neprijavljenog korisnika - @auth i @guest. Za složenije aplikacije, moguće je koristiti Gate i Policy funkcionalnosti. U slučajevima kada postoji veći broj

provjera, koje je teško pisati unutar @if Blade metode, preporuča se njihovo korištenje zbog bolje organizacije programskog koda. Gate se koristi za jednostavne odluke autorizacije određenih funkcionalnosti, koji se navodi unutar *boot()* funkcije. Uvijek prihvaćaju korisnika kao prvi parametar, dok su ostali opcionalni. Policy je klasa koja sadrži informacije o autorizaciji svih funkcionalisti vezanih za određeni model. Pravila definirana unutar Policya mogu se koristiti i putem Blade metode @can, u svrhu prilagođavanja sadržaja stranice.

```
// PROVJERA UNUTAR MIDDLEWARE DATOTEKE
   public function handle(Request $request, Closure $next) {
      $file = $request->route()->parameters()['file'];
      if (($file->user_id != auth()->user()->id) && !$file->is_public) {
         abort(403, 'Forbidden access');
      }
      return $next($request);
   }
// PRIMJENA NAVEDENE ZAŠTITE NA PUTANJI I KORIŠTENJE UUID-a UNUTAR URL-a
Route::get('download/{file:generated_file_name}', [FileController::class, 'downloadFile'])
      ->middleware('owner-public')->name("file.download");
// ZAŠTITA UNUTAR STRUKTURE STRANICE
@if(auth()->user()->is_admin)
      <li class="nav-item">
           <a class="nav-link" href="{{ route('admin') }}">Admin</a>
      </li>
      <li class="nav-item">
           <a class="nav-link" href="{{ route('statistics') }}">Statistics</a>
      </li>
@endif
      <li class="nav-item">
          <a class="nav-link" href="{{ route('signout') }}">Logout</a>
      </li>
```

Primjer 5.8. *Provjere u middleware datoteci i korištenje Blade metoda*

5.1.6 Prevencija propusta u kriptiranju podataka

Pri izradi aplikacije, praćene su smjernice iz službene dokumentacije Laravela za zaštitu podataka. Zaporke prolaze kroz *Hash()* funkciju prije pohranjivanja u bazu podataka te se koristi ranije navedeni UUID umjesto jedinstvenog identifikatora iz baze podataka u URL putanjama. Također, datoteke koje se spremaju u aplikaciji pohranjuju se u privatan direktorij. Laravel pruža mogućnost korištenja dva algoritma kodiranja – Bcyrpt i Argon2. Idući koraci su prebacivanje lokalne pohrane na sigurnu pohranu u oblak računala (primjerice Amazonov AWS S3 paket). Također, preporuča se omogućavanje HTTPS protokola i prebaciti aplikaciju na port 443, koji osigurava korištenje SSL-a. Ukoliko aplikacija posjeduje SSL certifikat, potrebno je konfigurirati rad aplikacije na način da ga se primjenjuje.

5.1.7 Prevencija neovlaštene promjene direktorija

Glavna prijetnja za izvođenje ovog oblika napada nalazi se u funkcionalnosti preuzimanja datoteke. Napad je uspješan u slučajevima kada se korisnički unos primjenjuje za naziv datoteke koja se preuzima. Ovdje se otvara mogućnost izvođenja dot-dot-slash napada. Ovaj propust je izbjegnut generiranjem nasumičnog imena za datoteku. Prilikom preuzimanja datoteke, ne koristi se direktno navedena putanja, koju može izmijeniti korisnik. Umjesto toga, na temelju putanje za preuzimanje datoteke pronalaze se njezine informacije pohranjene u bazi podataka. Prilikom kreiranja projekta, Laravel kreira javno dostupan i privatan dio aplikacije. Datotekama unutar javnog dijela aplikacije moguće je pristupiti direktno, dok se datoteke i direktoriji u privatnom dijelu nevidljivi web serveru i nemaju direktan pristup. S obzirom da se datoteke aplikacije pohranjuju unutar storage direktorija, koji pripada privatnom dijelu, automatski su zaštićeni od neovlaštenog pristupa. Također, konfiguracije servera ne dopuštaju izvođenje napada primjenom simbola ../ unutar putanje. U slučajevima kada se upotrebljava korisnički unos za kreiranje putanje, potrebno je koristiti funkcije poput *basename()*. S obzirom da se ne upotrebljava korisnički unos u aplikaciji, nije potrebno koristiti ovu funkciju, ali je svejedno dodana. Razlog tome je slučaj kada napadač ostvari pristup bazi podataka. Ukoliko unutar baze napadač izmijeni vrijednost generiranog naziva datoteke u neki od oblika napada (primjerice *../../.env*), klikom na poveznicu za preuzimanje može izaći iz trenutnog direktorija i pristupiti zaštićenoj datoteci. Prikaz korištenja navedene funkcije u aplikaciji dana je na primjeru 5.9., nakon čega slijedi njezino objašnjenje i potencijalna alternativa.

```
// FUNKCIJA ZA PREUZIMANJE DATOTEKE
    public function downloadFile(File $file)
    {
        $download_link = storage_path('app/user-files/' . basename($file->generated_file_name));
        if (file_exists($download_link)) {
            return response()->download($download_link);
        }
    }
```

Primjer 5.9. *Osiguravanje ispravne putanje prilikom preuzimanja datoteke*

storage_path() funkcija omogućava pristup datotekama u storage direktoriju te se unutar nje navodi željena putanja. U ovom programskom rješenju koristi se ranije navedena funkcija *basename()*, koja iz navedenog parametra miče sve dijelove koji predstavljaju putanju. Ukoliko se vrijednosti u bazi podataka nisu neovlašteno mijenjale, aplikacija će raditi normalno jer tamo nisu spremljene nikakve vrijednosti putanje. Međutim, ukoliko se zapis promijeni u oblik *../../.env*, navedena putanja se sklanja i funkcija preuzimanja ne uspijeva, jer se navedena datoteka ne nalazi unutar trenutnog direktorija. Ukoliko se želi zadržati određene informacije putanje, poput poddirektorija, preporuča se korištenje alternativne funkcije – *realpath()*. Navedena funkcija prvo provjerava postoji li navedena datoteka na traženoj putanji. Ukoliko ne postoji, vraća neistinitu vrijednost. Ukoliko se datoteka nalazi na traženoj putanji, vraća se njezina potpuna putanju i izbacuju se oznake za promjenu direktorija, ukoliko ih putanja sadrži. Također, na navedenu funkciju se dodaje posrednički sloj CheckOwnerOrPublic.php, čime se osigurava dodatna provjera prije samog izvođenja funkcije.

5.1.8 Prevencija ReDoS napada

Za prevenciju ReDoS napada u validaciji su se koristili regularni izrazi preporučeni u službenoj dokumentaciji i na forumima vezanim uz Laravel projekte. Provodi se validacija svih korisničkih unosa, postavljaju se ograničenja na dostupne simbole (primjerice za ime i prezime) te se korisnički unosi ne povezuju međusobno, kao u primjeru 3.24. Na nekim mjestima su korištena ograničenja za duljinu unosa, kako bi se smanjila potrebna obrada. Određeni postupci validaciji unosa prikazani su na primjeru 5.10.

```
'first_name' => ['required', 'regex:/^[a-ŽA-Ž. -]+$/', 'max:50'],
'email' => ['required', 'email', 'max:100', 'unique:users',
'regex:/^[A-Ža-Ž0-9\.]*@(ferit|etfos).hr$/'],
'user_file_name' => ['required', 'regex:/^[a-ŽA-Ž0-9._ -]+$/', 'max:70'],
```

Primjer 5.10. *Primjeri validacije nekih ulaza*

5.1.9 Prevencija SSRF napada

Kako je ranije navedeno, prevencija SSRF napada nema univerzalni pristup. Temeljni korak je minimizirati uporabu korisničkog unosa u putanjama aplikacije, što je ostvareno. Prilikom objave aplikacije, preporuča se korištenje popisa dozvoljenih ili blokiranih IP adresa i domena, na temelju kojih se odlučuje hoće li se korisnički zahtjev uopće obraditi. Osim korištenja ažurnih popisa adresa, potrebno je osigurati da je server na kojem se pokreće aplikacija također ažuran i implementira najnovije sigurnosne procedure.

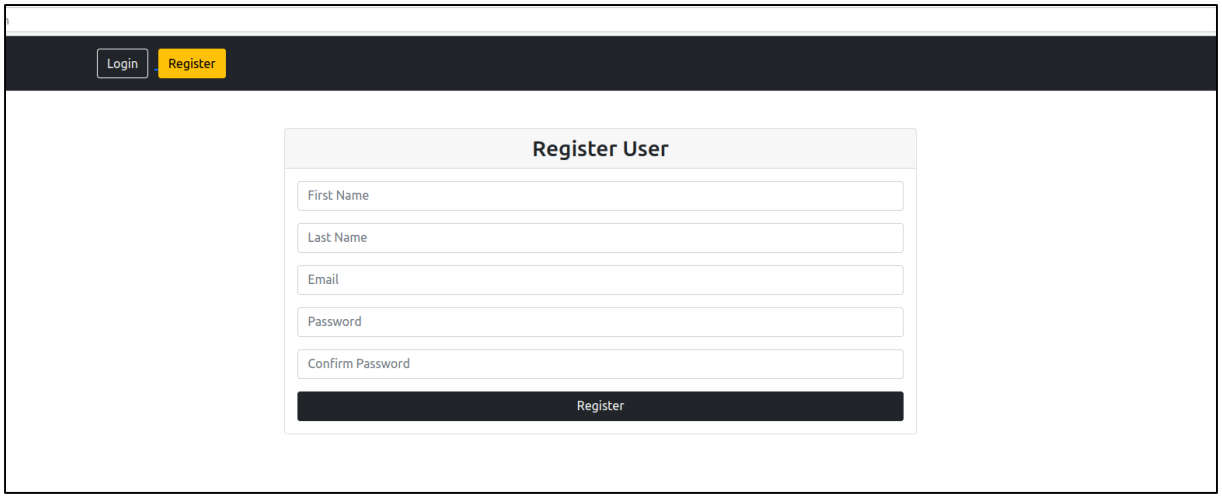
5.1.10 Prevencija RCE napada

Preventivne mjere za ovaj oblik napada poduzete su još u ranijim verzijama Laravela. Na verziji 5.6 uočeni su propusti prilikom deserijalizacije vrijednosti X-XSRF-TOKEN-a. Za izvršavanje napada je bilo potrebno poznavanje generiranog ključa aplikacije, koji nije javno dostupan, ali su ipak implementirane promjene za sprječavanje iskorištavanja propusta. Također, verzija 8.4.3 je imala propust u slučaju da je debug način rada ostao uključen. U toj situaciji, napadač je mogao izvoditi vlastiti programski kod prilikom poruke o pogrešci, bez obzira što nema ovlasti za to. Aplikacija kreirana u ovom radu koristila je Laravel 8.83.8, a idući korak prevencije ovog oblika napada je prelazak na novu verziju – Laravel 9.10. Time se osigurava da je aplikacija ažurirana na najnovije sigurnosne postavke vezane uz poznate sigurnosne prijetnje. Jednostavan korak za dodatnu zaštitu je onemogućavanje debug načina rada, jer je veliki dio prijetnji ovisan o takvom pristupu. Također, u svim funkcijama aplikacije definiran je očekivani tip parametara, čime se onemogućava unošenje neispravnih objekata.

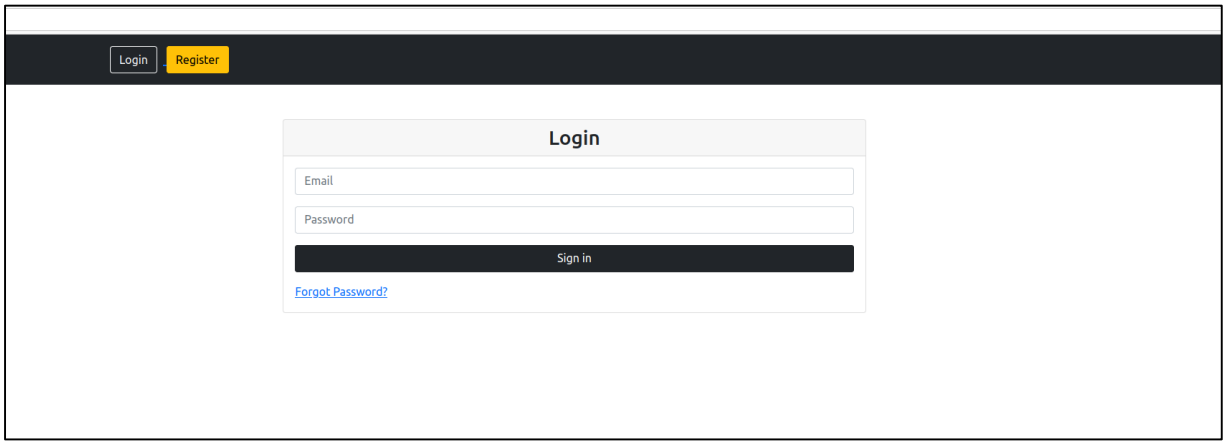
5.2 Korištenje aplikacije

U posljednjem poglavlju prikazat će se upotreba web aplikacije. Kako bi korisnik mogao pristupiti aplikaciji, mora izvršiti registraciju i prijavu. Stranice za te funkcionalnosti prikazane su slikama 5.1. i 5.2. Prilikom registracije, korisnik mora unijeti jedinstvenu email adresu, koja završava domenom *etfos.hr* ili *ferit.hr*. Također, zaporka mora zadovoljiti uvjete navedene u poglavlju 5.1.2. kako bi korisnik bio uspješno registriran. Prijava se izvršava

unošenjem jedinstvene email adrese i zaporke, a u slučaju da korisnik izvrši osam neuspjelih prijava unutar dvije minute, onemogućava mu se ponovna prijava i prikazuje se zaslon s obavijesti da je izvršeno previše pokušaja.



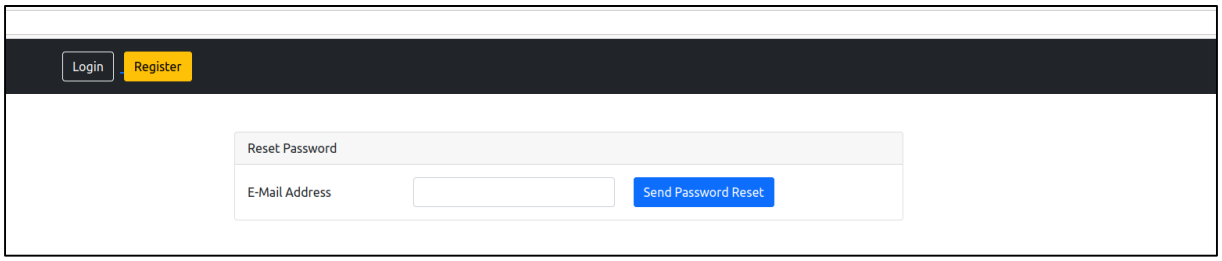
Slika 5.1. *Zaslon za registraciju korisnika*



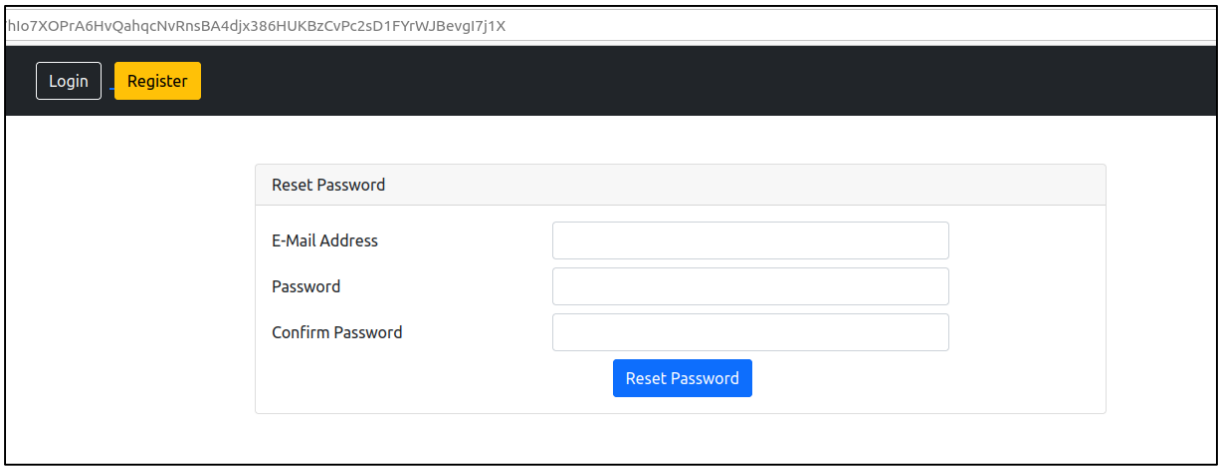
Slika 5.2. *Zaslon za prijavu korisnika*

U slučaju da je korisnik zaboravio zaporku, ili ju samo želi izmijeniti, može odabrati poveznicu *Forgot Password*. Time ga se odvodi na zaslon prikazan slikom 5.3. Tamo mora unijeti email adresu, koja je već iskorištena za registraciju korisnika, na koju će se poslati mail s poveznicom za oporavak lozinke. Prilikom kreiranja zahtjeva za oporavak, u tablicu se navode vrijednosti unesene email adrese i jedinstvenog tokena. Klikom na poveznicu koju korisnik dobiva na email adresu, odvodi ga se na stranicu prikazanu slikom 5.4. Na njoj je potrebno unijeti email adresu i novu zaporku, koja mora ispunjavati iste uvjete kao prilikom registracije. Skupa s novom zaporkom i email adresom, u zahtjevu se šalje i navedeni token, vidljiv u URL putanji. Ovime se osigurava da nije moguće izmijeniti tuđu zaporku, odnosno

da vrijednosti tokena i email adrese moraju odgovarati onima iz tablice. Tek nakon provjere tih vrijednosti ažurira se vrijednost zaporke.



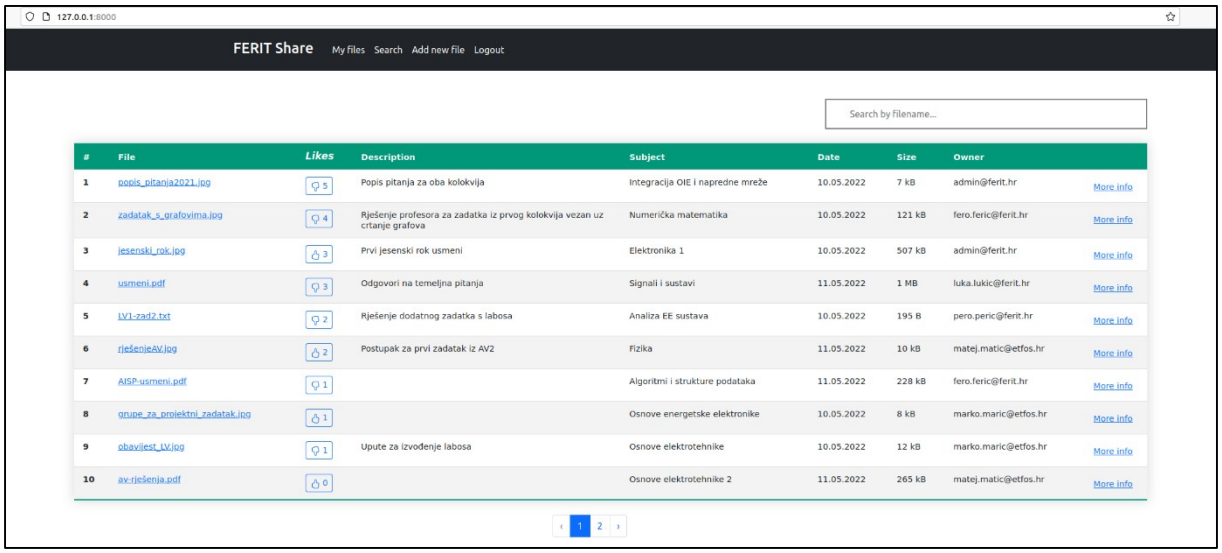
Slika 5.3. *Forma za unos email adrese čiju lozinku treba promijeniti*



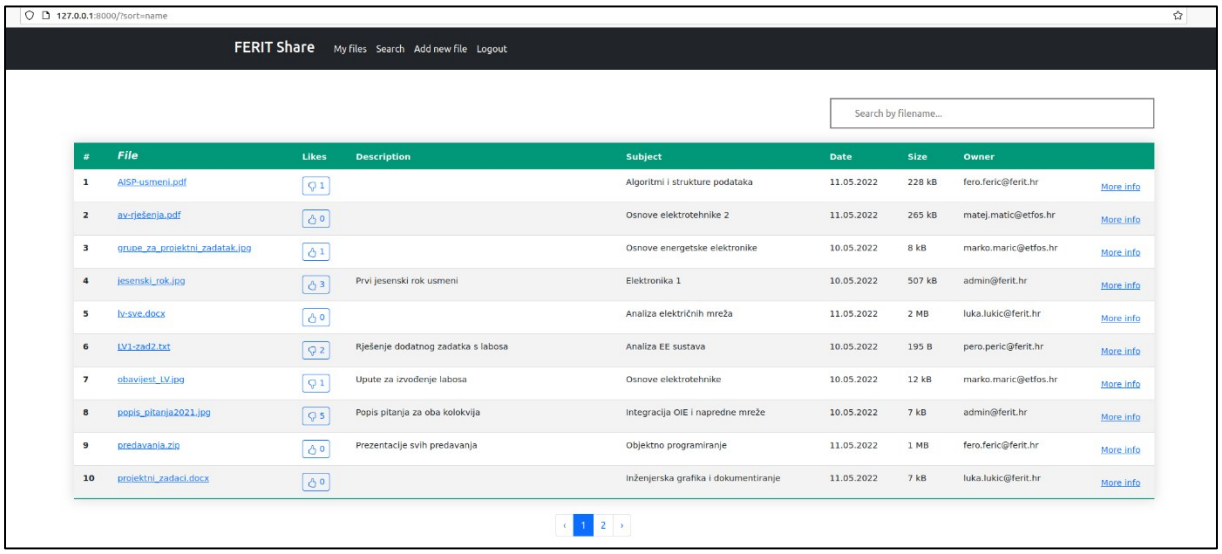
Slika 5.4. *Forma za unos nove zaporke*

Nakon uspješne prijave, korisnik se preusmjerava na početni zaslon gdje se nalaze sve javno dostupne datoteke. Datoteke su poredane s obzirom na broj pozitivnih ocjena korisnika, ali je moguće odabrati više načina sortiranja. Datoteke je moguće sortirati po imenu, broju pozitivnih ocjena, vremenu dodavanja, veličini i imenu autora. Datoteke se sortiraju klikom na naziv stupca, a ponovni klik mijenja smjer sortiranja (primjerice s najnovijih datoteka na najstarije). Navedeni zasloni prikazani su na slikama 5.5., 5.6. i 5.7. Na zaslonu je moguće vidjeti naziv datoteke te broj pozitivnih ocjena zajedno s opcijom dodavanja ili ukidanja vlastite pozitivne ocjene. Nadalje, prikazuje se njezin opis dodan od strane vlasnika, predmet kojem je datoteka namijenjena te njezina veličina i email adresa vlasnika. Također, postoji opcija za više informacija o datoteci, koja vodi na zaslon gdje su prikazane dodatni detalji, kao i komentari korisnika aplikacije vezani uz samu datoteku. Navedena funkcionalnost bit će opisana kasnije u radu. Datoteke prikazane na početnom zaslonu moguće je preuzeti klikom na njihov naziv. Zaslon koji vidi administrator identičan je zaslonu običnog korisnika, uz dodatnu opciju uklanjanja javno dostupne datoteke. Također, ima dodatne poveznice u

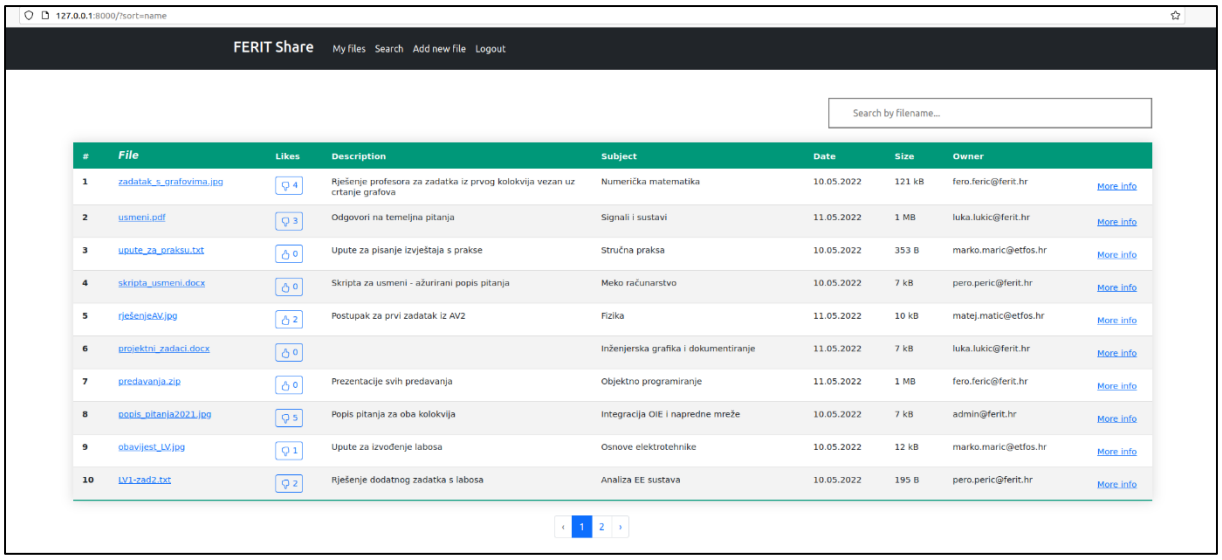
izborniku na vrhu stranice, namijenjene putanjama kojima može pristupiti samo korisnik s ovlastima administratora. Navedeni zaslon prikazan je na slici 5.8.



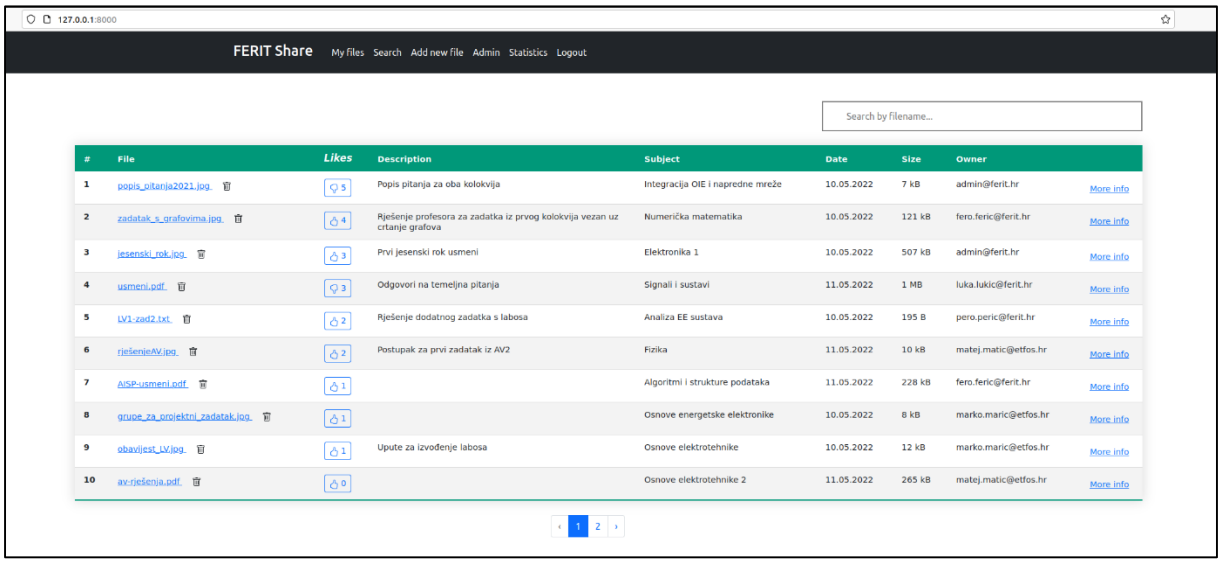
Slika 5.5. *Početni zaslon aplikacije s najbolje ocijenjenim datotekama*



Slika 5.6. *Početni zaslon aplikacije gdje su datoteke poredane po nazivu od A do Ž*

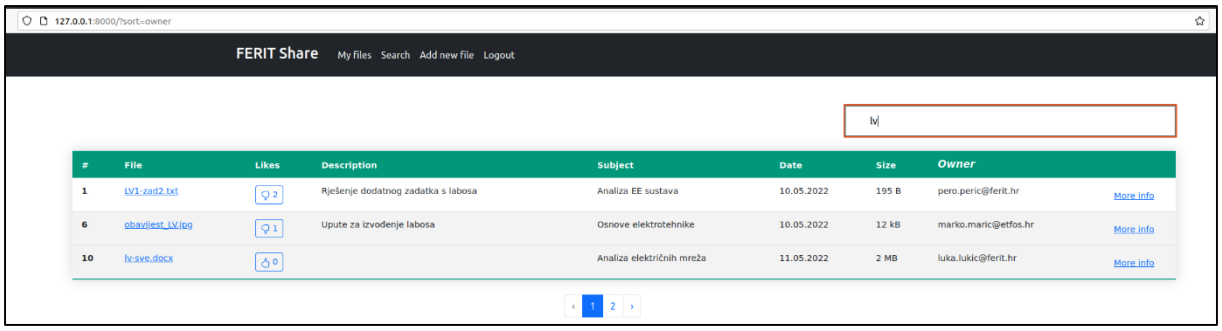


Slika 5.7. *Početni zaslon aplikacije gdje su datoteke poredane po nazivu od Ž do A*

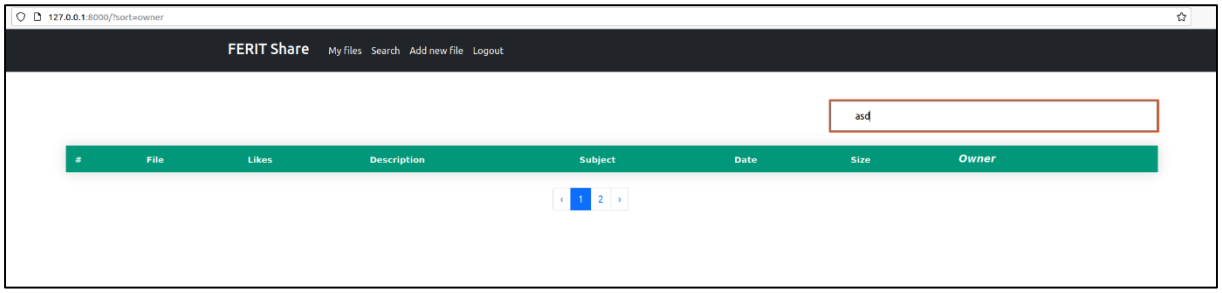


Slika 5.8. *Početni zaslon korisnika s ovlastima administratora*

Također, postoji opcija filtriranja datoteka na početnoj stranici. Unošenjem imena datoteke unutar polja iznad tablice, sakrivaju se one datoteke koje ne sadrže traženu riječ u svojem nazivu. Navedena funkcionalnost ostvarena je pomoću JavaScript funkcije, koja prolazi kroz popis i provjerava nazive prikazanih datoteka. Rezultati pretraživanja vidljivi su na slikama 5.9. i 5.10. Uneseni tekst nije osjetljiv na velika i mala slova.

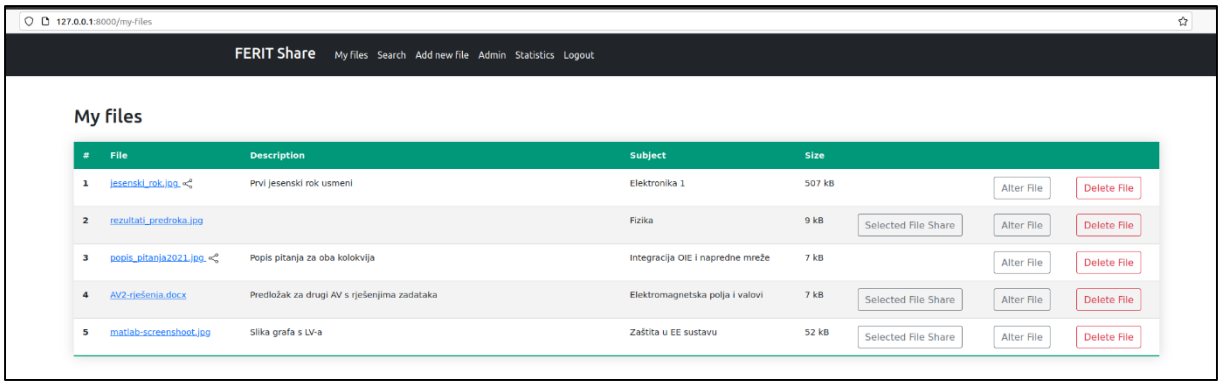


Slika 5.9. *Datoteke na početnom zaslonu koje sadrže „lv“ u svojem nazivu*

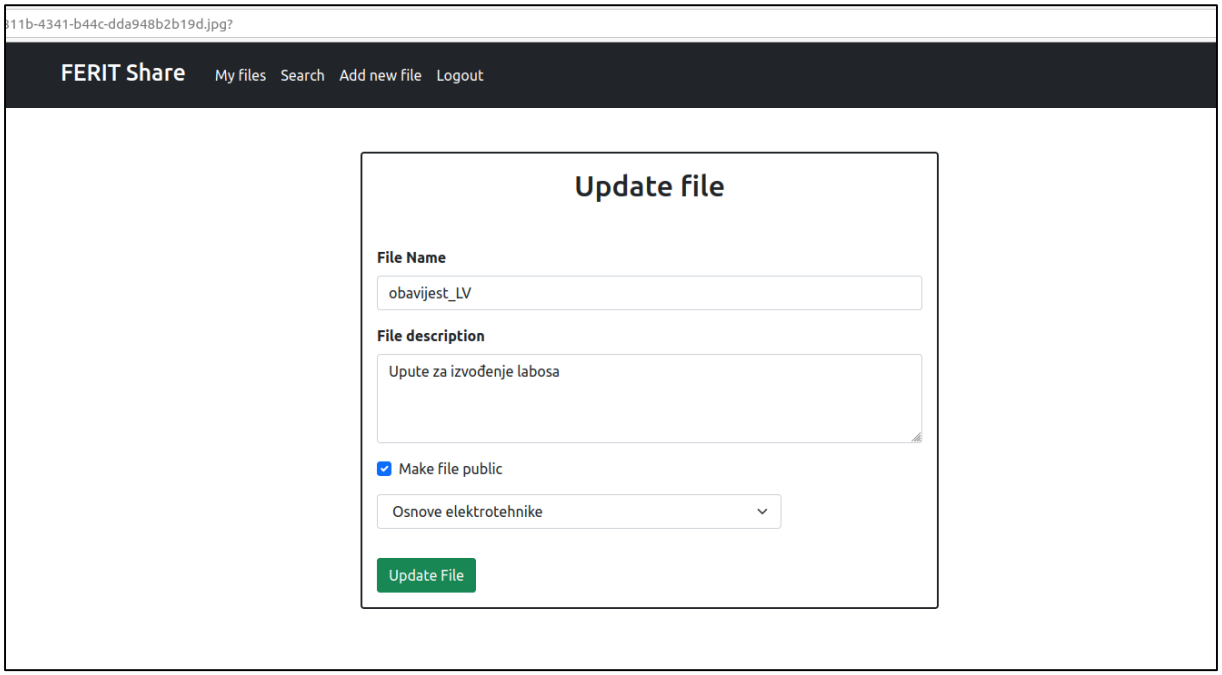


Slika 5.10. *Rezultat kada niti jedna datoteka ne ispunjava uvjet*

Odabirom opcije *My files* u izborniku, korisnik odlazi na zaslon s vlastitim datotekama. Tamo se nalaze njegove javne i privatne datoteke, uz ostale informacije dostupne i na početnom zaslonu. Ukoliko je datoteka označena kao javna, pored naziva sadrži ikonicu koja označava dijeljenje. Na ovom zaslonu nalaze se i dvije dodatne funkcionalnosti – izmjena datoteke i njezino brisanje. Datoteke koje su privatne imaju posebnu funkcionalnost, a to je njihovo dijeljenje sa specifičnim korisnikom. Odabirom izmjene, korisnik se preusmjerava na zaslon gdje može ažurirati trenutne vrijednosti naziva datoteke, njezinog opisa i predmeta kojem pripada te promijeniti njezinu privatnost. Unutar putanje za odlazak na navedenu stranicu koristi se jedinstveno ime datoteke, generirano prilikom njezinog pohranjivanja u bazu podataka. Ukoliko korisnik pokuša izmijeniti vrijednosti datoteke kojoj nije vlasnik (primjerice kopiranjem generiranog imena datoteke iz poveznice za preuzimanje), prikazuje mu se poruka o neovlaštenom pristupu. Nakon potvrde izmjena, korisnik se vraća na zaslon sa svojim datotekama. Prilikom brisanja datoteke, uklanja se njezin zapis u bazi podataka, kao i sama datoteka iz pohrane. Izgled navedenih zaslona prikazan je slikama 5.11. i 5.12.



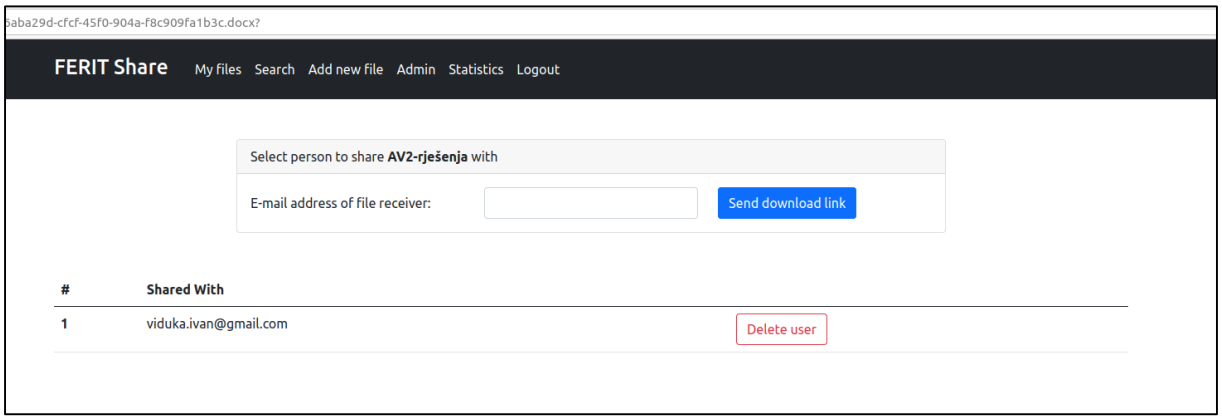
Slika 5.11. *Zaslon s datotekama koje pripadaju korisniku*



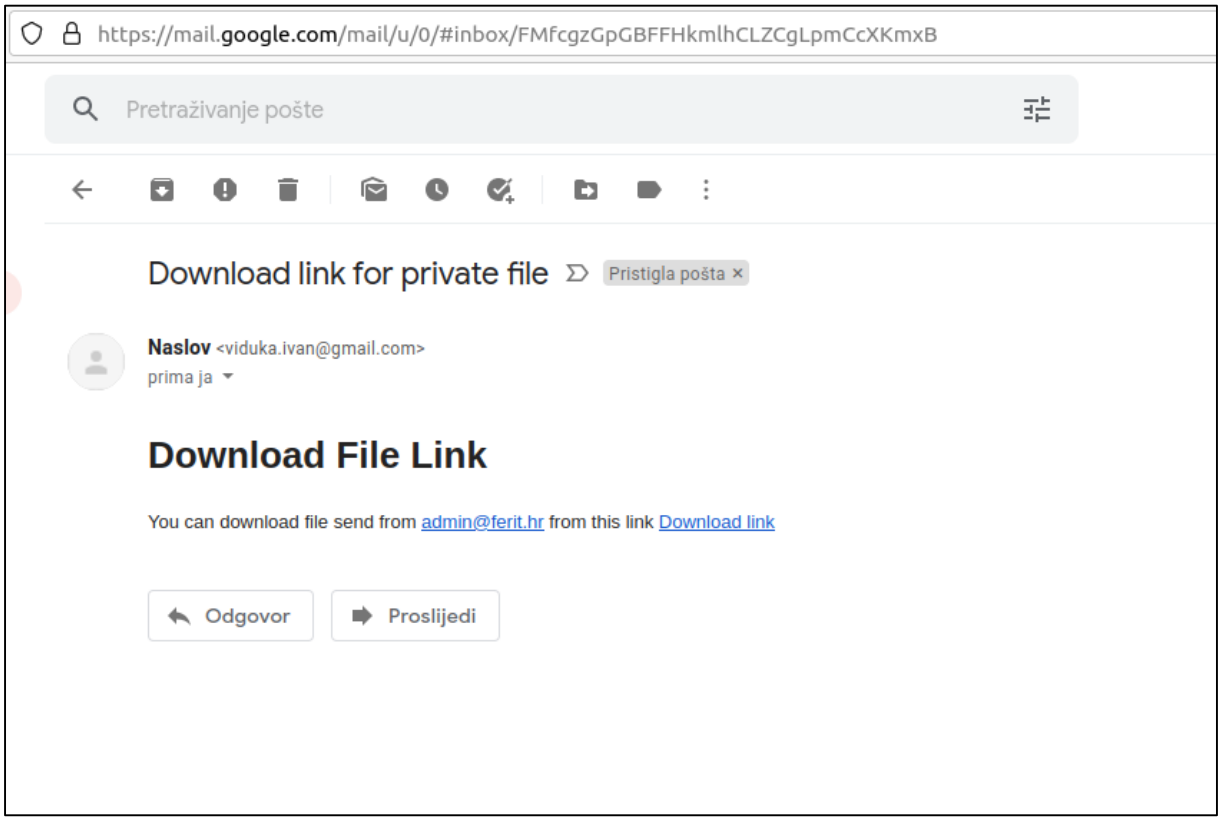
Slika 5.12. *Zaslon za ažuriranje informacija o datoteci*

Prilikom odabira funkcionalnosti za dijeljenje privatne datoteke s drugim korisnikom vrijede ista ograničenja kao pri ažuriranju vrijednosti datoteke – koristi se jedinstveno ime i vrši se provjera je li prijavljeni korisnik vlasnik datoteke. Odabirom navedene opcije, korisnik se preusmjerava na stranicu koja sadrži formu za unos email adrese. Unesena adresa mora pripadati osobi koja je korisnik aplikacije. Nakon unošenja podataka, na navedenu adresu se šalje email koji sadrži poveznicu za preuzimanje datoteke. Također, na stranici se prikazuje ispis korisnika s kojima je datoteka već podijeljena. Ukoliko se pokuša poslati poveznica korisniku koji već ima pristup, dobiva se obavijest da je datoteka već podijeljena te se ne šalje novi email. Vlasnik datoteke može obrisati osobe s navedenog popisa čime im se onemogućava preuzimanje. Također, prilikom preuzimanja datoteke provjerava se je li trenutno prijavljeni korisnik onaj kome je poveznica namijenjena. Prilikom slanja poveznice, u tablicu se pohranjuje jedinstveni generirani naziv datoteke i email osobe kojoj se daje

pristup. Prije preuzimanja se provjerava jesu li obje vrijednosti ispravne, odnosno nalaze li se zajedno u retku tablice. Ukoliko korisnik kojem nije namijenjena datoteka pokuša otvoriti poveznicu za preuzimanje, dobiva grešku o neovlaštenom pristupu. Prikaz stranice za dijeljenje dan je na slici 5.13. dok je email s poveznicom prikazan na slici 5.14.



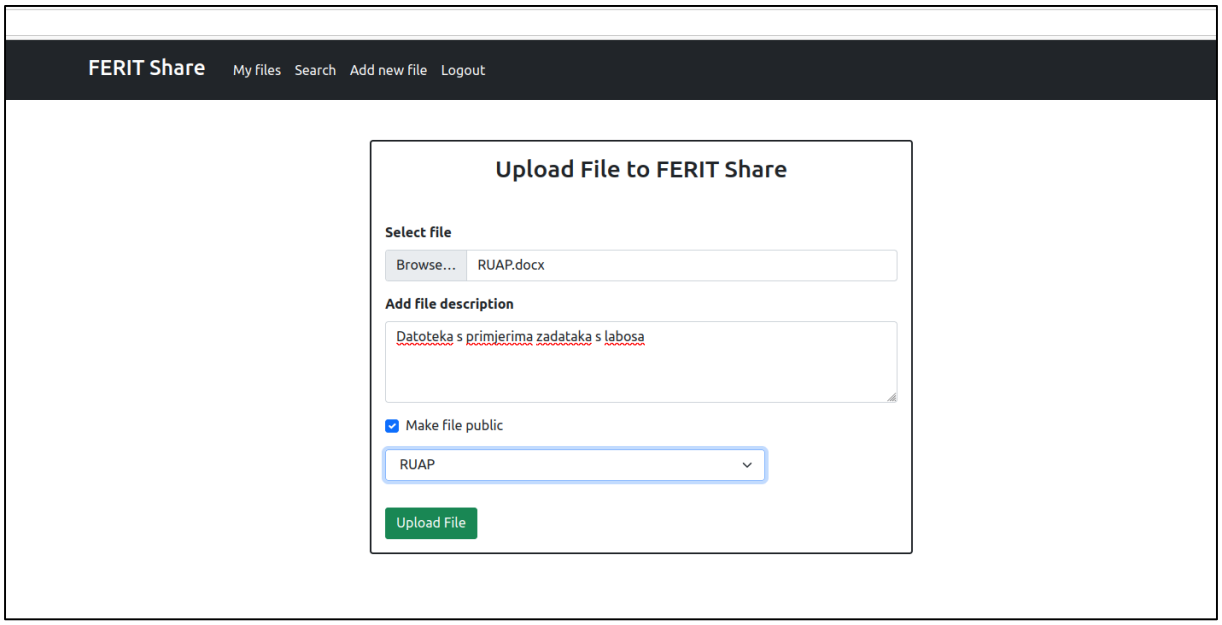
Slika 5.13. *Forma za unos email adrese s ispisom već postojećih dijeljenja datoteke*



Slika 5.14. *Email koji sadrži poveznicu za preuzimanje privatne datoteke drugog korisnika*

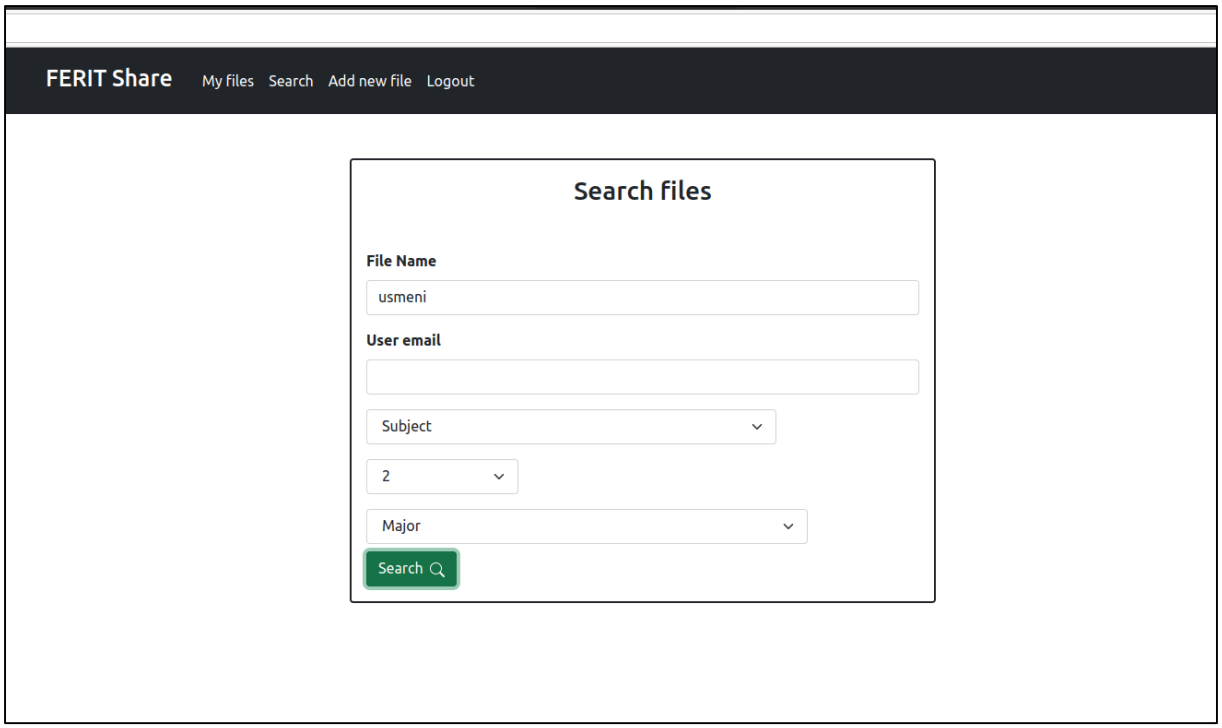
Stranica za dodavanje datoteke slična je stranici za ažuriranje vrijednosti. Njoj se pristupa preko izbornika, putem poveznice *Add new file*. Sadrži polja za unos datoteke i odabir predmeta, koja su obvezna, te polje za opis sadržaja datoteke i odabir o njezinoj privatnosti. U

aplikaciji je postavljeno ograničenje za veličinu datoteke, a maksimalna vrijednost je 10MB. Prikaz zaslona nalazi se na slici 5.15.

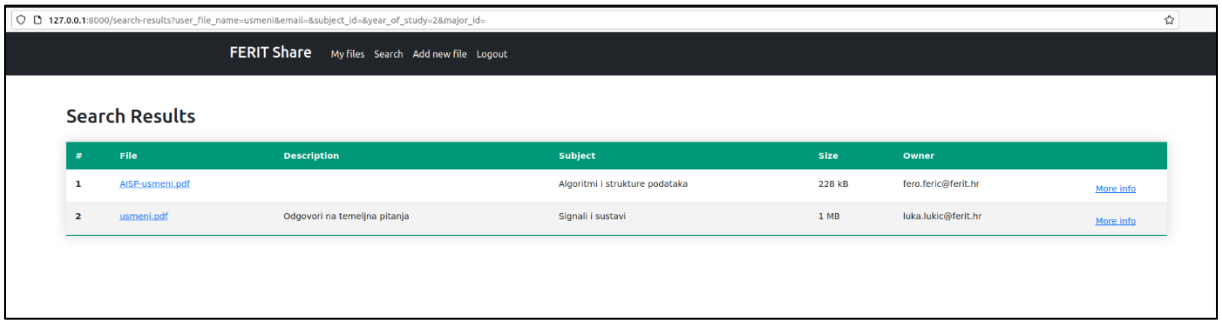


Slika 5.15. *Zaslon za dodavanje datoteke*

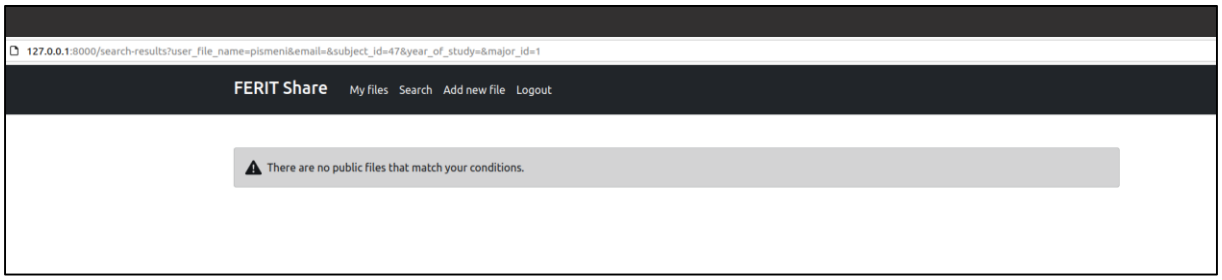
Odabirom opcije *Search* unutar izbornika, korisnika se odvodi na stranicu za pretraživanje datoteka po željenim parametrima. Korisnik može pretraživati datoteke s različitim kombinacijama vrijednosti za njihov naziv, osobu kojoj pripadaju te godinu studija, studijski program i predmet za koji su datoteke namijenjene. Ukoliko se unese kombinacija uvjeta koju ne ispunjava niti jedna datoteka, korisnik dobiva obavijest da takva datoteka ne postoji. U suprotnom, prikazuje se popis svih datoteka koje ispunjavaju uvjet, u tablici nalik onoj s početnog zaslona. Prikaz pretrage i njezinih rezultata dani su slikama 5.16. i 5.17., dok je poruka za neuspjelo pretraživanje prikazana slikom 5.18. Za navedenu pretragu traže se datoteke koje u sebi sadrže riječ usmeni i pripadaju predmetima na drugoj godini studija.



Slika 5.16. *Forma za unos željenih parametara pretrage*



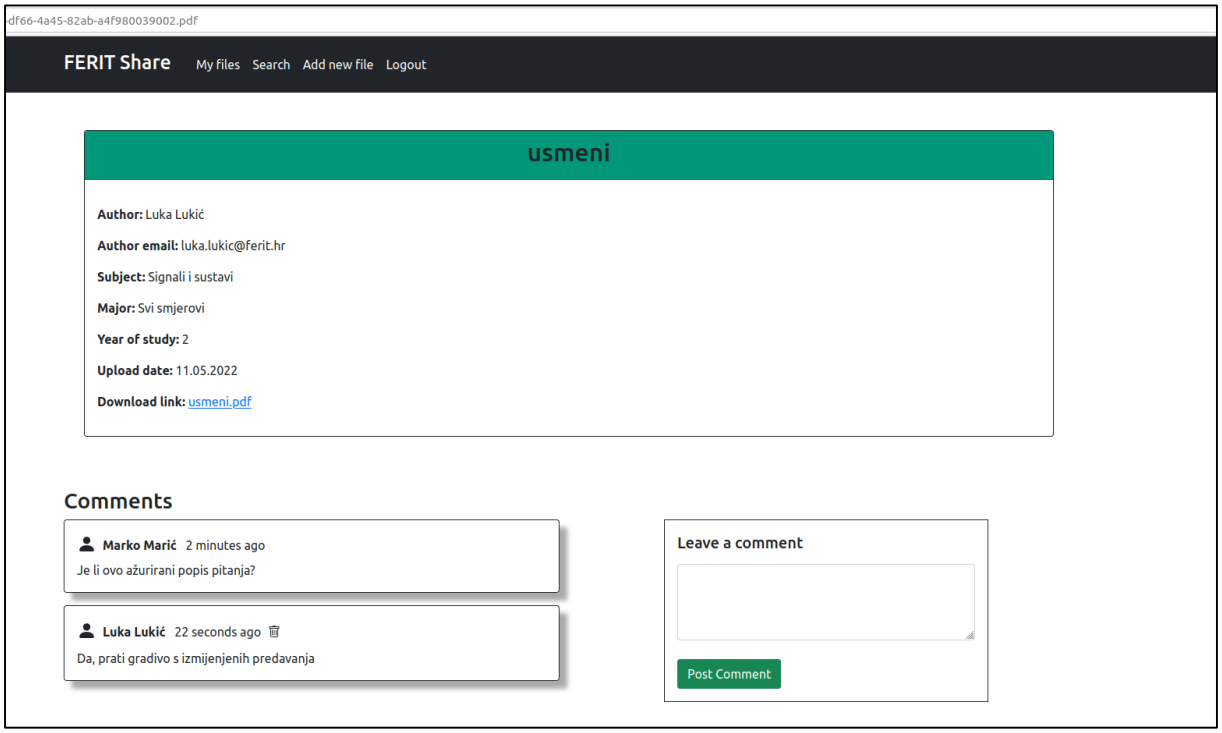
Slika 5.17. *Rezultati pretrage*



Slika 5.18. *Poruka u slučaju nepostojanja datoteka*

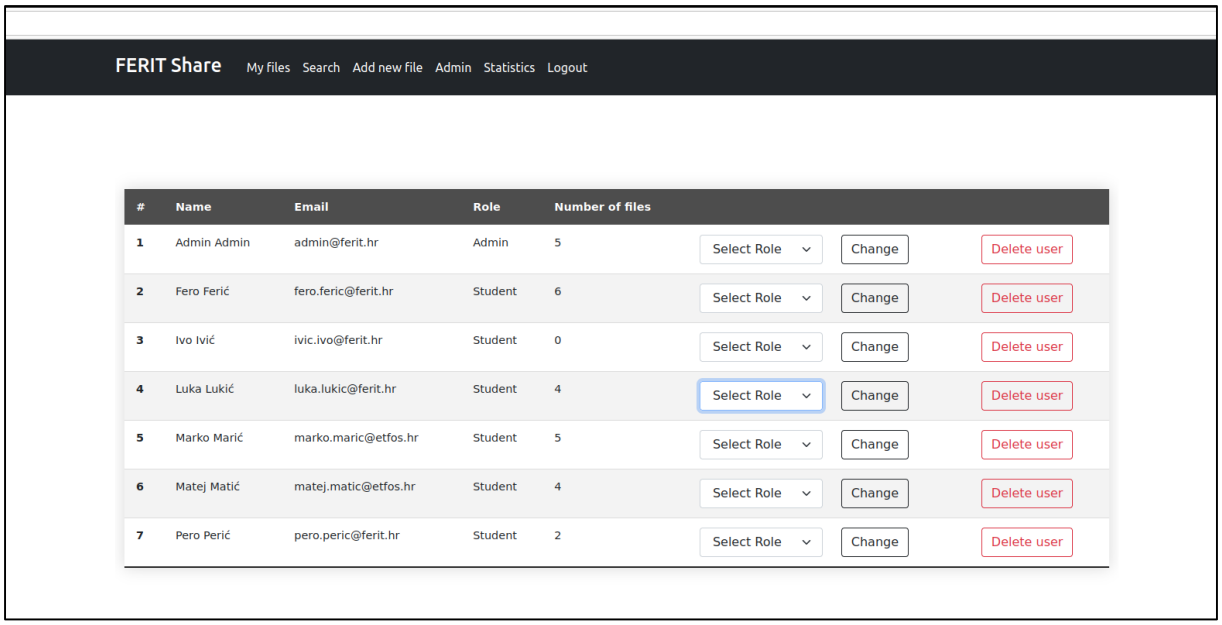
Kako je ranije navedeno, s početnog zaslona je moguće odabrati opciju za više detalja o datoteci. U tom slučaju, prikazuje se ime autora i njegova email adresa, predmet kojem je datoteka namijenjena te naziv studentskog smjera i godina kojoj predmet pripada, datum objavljivanja i poveznica za preuzimanje datoteke. Ispod navedenih informacija nalaze se komentari korisnika. Običan korisnik može obrisati svoj komentar dok administrator može

obrisati sve. Komentar može ostaviti bilo koji prijavljeni korisnik. U slučaju postavljanja dostupnosti datoteke na privatno, komentari se ne brišu iz baze te će biti ponovno vidljivi u slučaju nove promjene privatnosti. Komentari se uklanjaju iz baze jedino u slučaju brisanja datoteke iz aplikacije. Prikaz detalja datoteke zajedno s komentarima vezanim uz nju nalazi se na slici 5.19.

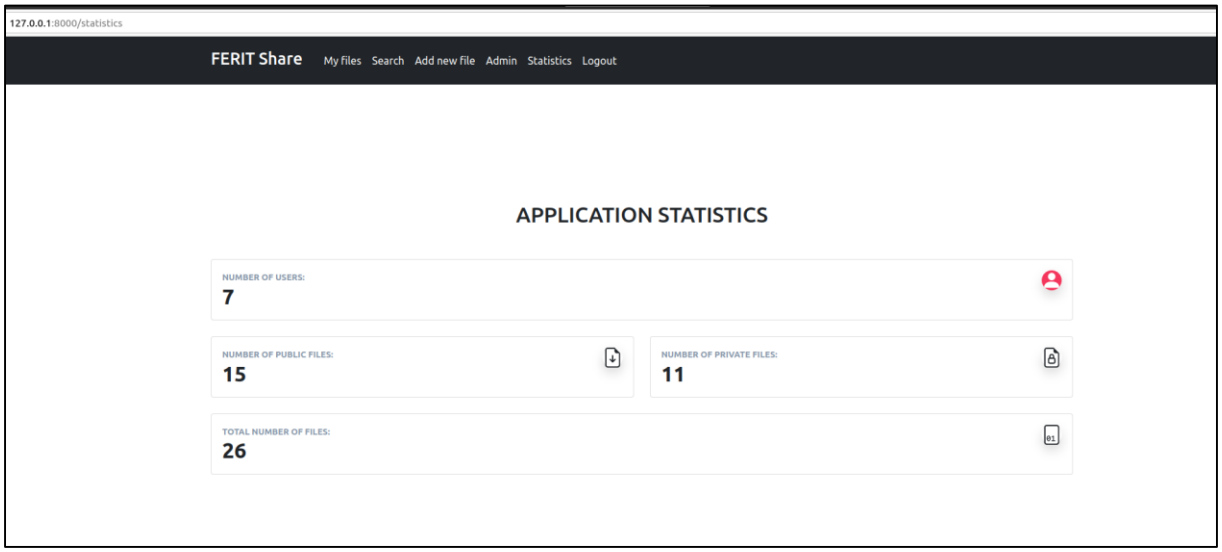


Slika 5.19. *Zaslon s detaljima o datoteci i komentarima*

Za kraj će se prikazati funkcionalnosti dostupne jedino administratoru. Prva je zaslon s popisom svih korisnika, na kojoj vidi njihovo ime, email adresu, trenutnu ulogu te broj pohranjenih datoteka. Administrator može mijenjati ulogu svim korisnicima aplikacije, odnosno dati im ili oduzeti ovlasti administratora. Također, ima opciju brisanja korisnika, pri čemu se uklanjanju sve njegove datoteke iz baze podataka i iz pohrane. Drugi zaslon dostupan administratoru je onaj na kojem su prikazani statistički podaci o aplikaciji. Zaslon sadrži podatke o ukupnom broju korisnika i datoteka, kao i broj privatnih i javnih datoteka u aplikaciji. Izgled navedenih zaslona prikazan je na slikama 5.20. i 5.21. Navedene stranice dostupne su samo korisniku s ovlastima administratora te se njihove poveznice ne prikazuju u izborniku običnog korisnika. Osim toga, same putanje su zaštićene provjerom ovlasti trenutno prijavljenog korisnika, čime se sprječava neovlašteni pristup.



Slika 5.20. *Zaslon s informacijama o korisnicima*



Slika 5.21. *Zaslon sa statističkim podacima*

6 ZAKLJUČAK

U ovom radu, prikazana je važnost primjene sigurnosnih postupaka prilikom izrade aplikacije. Porast broja aplikacija i korisnika, kao i sve veća upotreba tehnologije u svakodnevnom životu, povećavaju važnost sigurnosnog segmenta aplikacije. Svakodnevno raste broj korisnika, sve je veća količina informacija o njima te se aplikacije primjenjuju prilikom svakodnevnih aktivnosti. Ukoliko dođe do sigurnosnih propusta, šteta prouzročena uspješnim web napadima u današnje vrijeme je značajno veća nego u prošlosti. Zbog svega navedenog, sigurnost aplikacije zahtjeva fokus razvojnog tima prilikom njezine izrade.

Aplikacija kreirana u ovom radu na jednostavan način prikazuje različite mogućnosti napada te postupke koji se mogu poduzeti u njihovom sprječavanju. Osim navedenih napada, postoje još mnogi koji nisu obrađeni. Primarni fokus ove aplikacije bio je na napadima koji su najčešće i najuspješnije izvođeni. Aplikacija je realizirana na način da su prvo definirani zahtjevi i funkcionalnosti za svaku ulogu korisnika, a nakon toga implementirano programsko rješenje. Nakon realizacije svih funkcionalnosti, provedene su dodatne sigurnosne mjere za osiguravanje ispravne upotrebe aplikacije.

Sama aplikacija može se dodatno proširiti. Osim mogućnosti proširenja navedenih u poglavlju 5.1., mogu se dodati nove uloge korisnika, poboljšati interakcija među korisnicima te razne druge funkcionalnosti koje bi aplikaciju proširile u smjeru društvene mreže. Također, moguće je staviti dodatan naglasak na skalabilnost aplikacije.

LITERATURA

[1] M. Moilanen, Developing a Web Service - Databases, Security and Access Control, Jyväskylä, 2019.

[2] K.D. Mitnick, R. Vamosi, The Art of Invisibility, New York, 2017.

[3] A. Shostack, Threat Modeling - Designing for Security, Indiana, 2014.

[4] R.B. Castrillo, Research and Implementation of Cybersecurity Techniques in the Backend of a Web Application - poglavlje 6, Turku, 2019.

[5] M. Howard, D. LeBlanc, Writing Secure Code – poglavlje 4, Washington, 2015.

[6] J. Grossman, Cross Site Scripting Attacks: XSS Exploits and Defense, Vermont, 2007

[7] M. Shema, Seven Deadliest Web Application Attacks – poglavlje 2, New York, 2010.

[8] M. Chapple, Access Control and Identity Management, 3rd Edition, Burlington, 2020.

[9] D. Cherry, Securing SQL Server, 2nd Edition, Boston, 2012.

[10] L. Kohnfelder, Introduction to Software Security Chapter 3.3: Directory Traversal Attacks, 2022.

[11] J.C. Davis, F. Servant, The Impact of Regular Expression Denial of Service (ReDoS) in

Practice: An Empirical Study at the Ecosystem Scale, Virginia Tech, 2018.

[12] B. Jabiyev, E. Kirda, Preventing Server-Side Request Forgery Attacks, Northeastern University, 2021.

[13] P. Prasad, Mastering Modern Web Penetration Testing – poglavlje 12, Birmingham, 2016.

[14] Laravel dokumentacija, https://laravel.com/docs/8.x/blade, pristupljeno: 28.5.2022.

[15] StackHawk blog, https://www.stackhawk.com/blog, pristupljeno: 28.5.2022.

[16] OWASP – Laravel Security Cheat Sheet, https://cheatsheetseries.owasp.org/cheatsheets/Laravel_Cheat_Sheet.html,

pristupljeno: 28.5.2022.

SAŽETAK

U ovom radu kreirana je web aplikacija za pohranjivanje i dijeljenje datoteka među studentima Fakulteta elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Aplikacija kombinira karakteristike aplikacije za pohranu datoteka i društvenih mreža. U radu je stavljen naglasak na sigurnosni segment aplikacije, odnosno na različite vrste potencijalnih napada na aplikaciju. Navedeni su i objašnjeni najčešće izvedeni napadi te su dani koraci za prevenciju njihovog izvođenja. U praktičnom dijelu, realizirana je navedena aplikacija i prikazani su sigurnosni postupci za svaki od opisanih napada. Aplikacija je realizirana pomoću razvojnog okvira Laravel, uz ispunjavanje različitih funkcionalnosti za običnog korisnika i administratora.

Ključne riječi: Laravel, pohrana datoteka, prevencija web napada, sigurnosne prijetnje, web aplikacija

ABSTRACT

Web application for sharing files with different access permissions

In this master thesis, a web application for storing and sharing files for students of Faculty of Electrical Engineering, Computer Science and Information Technology in Osijek was developed. This application combines characteristics of application for storing files and social network. The emphasis of this thesis is security of web application, namely on different types of web attacks aimed at web application. Most common attacks were named and explained, together with a guide on how to prevent their execution. In practical part of this thesis, web application described earlier was implemented and security procedures for preventing each of the attacks named in thesis were shown. Application was implemented with the usage of Laravel framework, together with different types of functionalities for regular user and administrator.

Keywords: data storage, Laravel, security threats, web application, web attack prevention