Univerzitet u Sarajevu

Elektrotehnički fakultet

Odsjek za računarstvo i informatiku



**Sigurnosni aspekti web aplikacije razvijene pomoću React JS**

Završni rad

I ciklus studija

Mentor: Kandidat:

Prof. dr. Dženana Đonko Amila Borančić

Sarajevo, 2019. godina

**Postavka rada**

Cilj:

* Analiza sigurnosnih propusta komercijalnih aplikacija
* Upoznavanje sa metodama zaštite od najrasprostranjenijih napada
* Upoznavanje sa glavnim karakteristikama biblioteke React JS za razvoj web aplikacija
* Razvoj ogledne web aplikacije na kojoj će biti demonstrirane neke od metoda za zaštitu

Opis

Ovaj rad će obuhvatiti analizu sigurnosti web aplikacija, kao i upoznavanje sa metodama zaštite od istih. Potom će se obaviti kratki uvod u React JS biblioteku za razvoj *frontenda*, a nakon toga će se vršiti implementacija online trgovine pod nazivom #tech koja će ilustrovati koncepte razvoja u React JS-u.

Rad će se sastojati iz sljedećih poglavlja:

• Poglavlje 1 je „Uvod“ koje sadrži uvod u temu rada, glavne ciljeve i strukturu rada.

• Poglavlje 2 je „Upoznavanje sa najčešćim vrstama napada i metodama zaštite“

• Poglavlje 3 je „Analiza biblioteke React JS“ i predstavlja analizu načina rada i osnovnih koncepata React-a.

• Poglavlje 4 je „Specifikacija web aplikacije #tech“ u kojem je definisana namjena i funkcionalnosti pokazne web aplikacije.

• Poglavlje 5 je „Sigurnost aplikacije #tech“ u kojem je detaljno izučena implementacija zaštite od određenih napada.

• Poglavlje 6 je „Zaključak“ u kojem je sumirano sve što je predstavljeno u radu i rezultati dobijeni analizom sigurnosti razvijene ogledne aplikacije.

Očekivani rezultati:

* Analiza najčešćih napada na web aplikacije, kao i metoda zaštite od istih
* Analiza biblioteke React JS za razvoj *frontenda* aplikacije
* Razvijena ogledna aplikacija uz implementiranu sigurnosnu zaštitu

Polazna literatura:

* Robin Wieruch, The road to learn React, 2017
* Justin Clarke-Salt, SQL Injection Attacks and Defense, 2nd Edition

Mentor:

Prof. dr. Dženana Đonko

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Univerzitet u Sarajevu

Naziv fakulteta/akademije: Elektrotehnički fakultet

Naziv odsjeka i/ili katedre: Računarstvo i informatika

Predmet: Verifikacija i validacija softvera

**Izjava o autentičnosti radova**

Seminarski rad, završni (diplomski odnosno magistarski) rad za I i II ciklus studija i integrirani studijski program I i II ciklusa studija, magistarski znanstveni rad i doktorska disertacija[[1]](#footnote-1)

Ime i prezime: Amila Borančić

Naslov rada: “Sigurnosni aspekti web aplikacije razvijene pomoću React JS“

Vrsta rada: Završni rad I ciklusa studija

Broj stranica: ...

Potvrđujem:

• da sam pročitao/la dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;

• da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;

• da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;

• da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;

• da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;

• da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;

• da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentorice i akademskih tutora.

Mjesto, datum: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Potpis: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sadržaj

Sažetak

Abstract

Uvod

MyFitnessPal securty breach

Yahoo

Target lanac trgovina

Upoznavanje sa najčešćim vrstama napada i metodama zaštite

OWASP Top 10

Tipovi napada

Injection

Kako napad funkcioniše

Poslkedice i metode zaštite

XSS

Podvrste XSS napada

Kako napad funkcioniše

Posljedice i metode zaštite

Broken Authentication

Posljedice i metode zaštite

Analiza biblioteke React JS

Uvod u React JS

Komponente, props i state

Virtuelni DOM i JSX

Specifikacija web aplikacije #tech

Namjena

Pregled sadržaja

Funkcionalni zahtjevi

Akteri

Nefunkcionalni zahtjevi

Tehnologija za razvoj

Use Case dijagram

Sigurnost web aplikacije #tech

Zaključak

Prilog: Popis najčešćih XSS vektora

Prilog: Prototipi

Reference

Popis slika

Abstrakt

# *Uvod*

Sigurnost web aplikacija je veoma osjetljiva tematika koja je od presudnog značaja za svaku web aplikaciju. U nastavku obrađujemo zašto je toliko važna, koje su posljedice nedovoljne zaštite te koje mjere konkretno treba preduzeti kako bismo se zaštitili.

Kao uvod u tematiku, posmatrajmo primjer nekog stambenog objekta - recimo neka se radi o nečijem domu. Svako u svojoj kući ili stanu čuva određene dragocjenosti - od namještaja, stvari koje imaju sentimentalnu vrijednost ukućanima, do novca, nakita i drugih skupocjenih predmeta. Svjesni smo da je naš dom, kao i svačiji drugi, uvijek, u svako doba dana, podložan provalama. Taj rizik uvijek postoji (s obzirom na to da na području na kojem živimo uvijek ima kradljivaca), samo što ljudi često na njega zaborave.

Ima više načina da provalnik uđe u nečiji dom i ukrade sve što mu se prohtije - ali statistički najčešći je obijanje brave ulaznih vrata (34% svih provala u domove sačinjavaju provale pomoću obijanja brave (Hyld, 2019)).

Sada kada to znamo, prirodno se nameće veoma banalan način zaštite doma od provala; nabaviti jaka vrata, sa blindiranom bravom i svaki put kada napuštamo dom, dva puta zaključati vrata.

To, naravno, nije i *jedini* način da se zaštitimo, ali jeste jedan od efektivnijih.

Nekima može doći kao iznenađenje činjenica da je moguće (i to veoma jednostavno) povući analogiju između zaštite nečijeg doma od krađe i sigurnosti web aplikacija.

Kao i nečiji dom, baza podataka (s kojom svaka iole ozbiljnija web aplikacija mora komunicirati), čuva dragocjenosti, samo što se ne radi o fizičkim predmetima, već o nečemu mnogo delikatnijem - *informacijama.* Baza podataka čuva sve od ličnih podataka o korisnicima neke aplikacije, do brojeva kartica, pasoša, te raznih povjerljivih informacija za vladine organizacije, kao i mnoge druge važne podatke, ovisno od konkretne primjene.

*Kradljivac* bi, u kontekstu diskusije o web aplikacijama, bio *haker*, odnosno *korisnik* aplikacije zlih (i veoma ilegalnih) namjera koji pokušava probiti zaštitu aplikacije i ukrasti informacije koje je naša aplikacija dužna da čuva.

Ako se čitaoca ovim izlaganjem nije uvjerilo o tome zašto je toliko važna sigurnost aplikacija, pomozimo se nekim primjerima iz stvarnog života, koji veoma dobro ilustruju teške posljedice sigurnosnih propusta.

## *MyFitnessPal security breach*

25. marta 2018. godine je otkriven upad u bazu podataka aplikacije „MyFitnessPal“ čiji je vlasnik američka kompanija „Ander Armor“. Procjene stručnjaka kažu da su se hakeri probili do baze još u februaru, međutim tek nakon mjesec dana (možda i više) je upad otkriven.

Procjenjuje se da su podaci 150 miliona korisnika komprovizovani, te da su ukradena korisnička imena, šifre, kao i ostali lični podaci. Nekolicina tih podataka je završila na dark web-u, gdje su ih napadači prodavali za cca. 20 000$ BitCoina (što iznosi otprilike 138 miliona KM) .

Srećom po vlasnike aplikacije, napad je otkriven dovoljno rano i dovoljno brzo se je odreagovalo - svi korisnici su dobili email sa hitnom porukom o odabiru nove šifre, međutim problem je u tome što je iako veliki broj korisnika uspjelo spasiti svoje podatke i izbjeći katastrofu, veliki broj nije bio tolike sreće, te su njihovi podaci završili na dark web-u, te su tako postali žrtve moguće krađe identiteta.

## *Yahoo*

Ovo je vjerovatno najkritičniji sigurnosni propust 21. vijeka. Upad u bazu podataka društvene mreže *Yahoo* se je prvi put desio 2013. godine, o čemu je CEO kompanije šutio do septembra 2016. godine. Do danas nije poznato ko su bili napadači, međutim sumnja se na neku stranu vladu, najvjerovatnije Kinu.

Imena, prezimena, adrese, brojevi telefona i datumi rođenja 500 miliona korisnika su ukradeni i ti korisnici su svi pali pod rizik krađe identiteta. Par mjeseci kasnije, napad se je desio *ponovo* te su, pored ličnih podataka, ukradene šifre, usernames, email adrese korištene za resetovanje zaboravljenih šifri, te odgovori na neka sigurnosna pitanja još čitavog *biliona* korisnika.

Do 2016. godine se je vjerovalo da je 1.5 biliona korisničkih profila pokradeno, međutim procjena koju su radili stručnjaci 2017. godine je otkrila da je vjerovatnije da se radi o čitava *3 biliona* korisničkih profila.

*Yahoo*, koji je trebao biti prodat kompaniji *Verizon,* je izgubio 350$ dolara prodajne cijene radi ovog kobnog sigurnosnog propusta.

## *Target lanac trgovina*

U decembru 2013. godine napadači su se probili do baze podataka ovog velikana u trgovini pomoću *third-party HVAC vendor* (eng. *Heating, Ventilation and Air Conditioning vendor*).

Ovaj upad nam govori na koje korake su hakeri spremni kako bi se probili do nečije baze podataka. U slučaju lanca trgovina *Target,* prema inervjuu koji je predstavnik *Trageta* obavio sa reporterima *Wall Street Journal* i *Reuters*, ispostavlja se da su eksperti provbitni upad pratili unazad dok nisu došli do kompanije koja se je bavila HVAC instalacijama nazvane *Fazio Mechanical Services.* Dalja istraga je pokazala da su hakeri ukrali *netwrok access credentials* ove kompanije, te da su određeni sigurnosni propusti od strane *Targeta* omogućili da se ovi podaci iskoriste da hakeri upadnu na njihovu privatnu mrežu i tako se probiju do baze.

Što se tiče učinjene štete, prema posljednjoj procjeni, informacije o kreditnim karticama, kao i lični podaci oko 110 miliona korisnika je ukradeno. Procijenjena šteta s ekonomskog pogleda je iznosila oko 162 miliona američkih dolara.

Nakon provedene diskusije možemo zaključiti da se radi o veoma delikatnoj i osjetljivoj tematici, koju itekako treba detaljno proučiti i posvetiti joj posebnu pažnju. Takođe daje uvid u to zašto velike korporacije ulažu silni novac (radi se o ciframa od nekoliko miliona do nekoliko *biliona* američkih dolara, te varira od korporacije do korporacije) kako bi platili profesionalnce da im razvijaju sisteme za zaštitu. U nastavku detaljnije obrađujemo tipove napada koji postoje, njihove podvrste, te kako se od njih odbraniti.

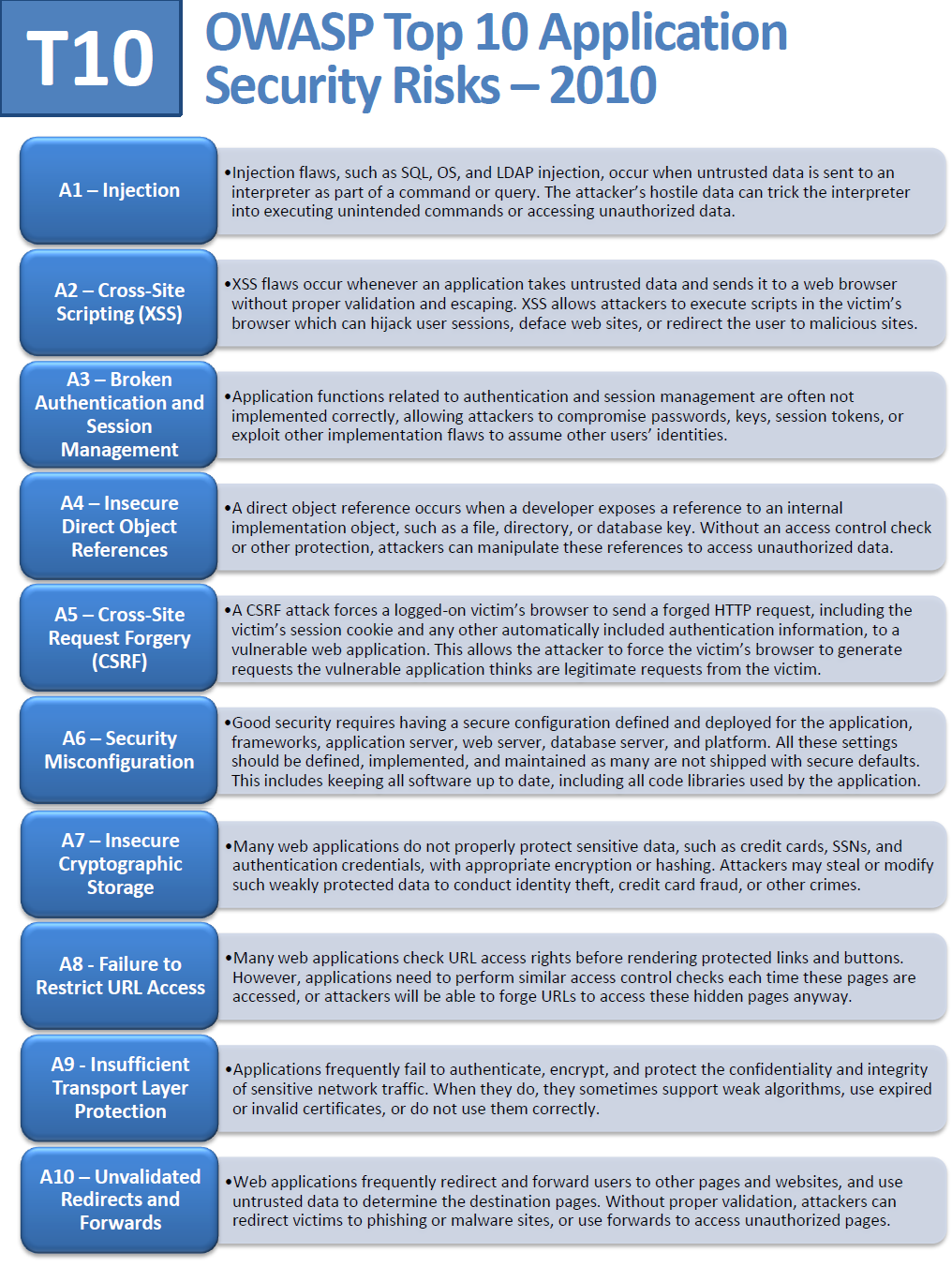
# *Upoznavanje sa najčešćim vrstama napada i metodama zaštite*

Sada kada smo uvidjeli zašto je zaštita od malicioznih napada toliko važna, slijedi da se upoznamo sa različitim tipovima napada koji postoje, kao i o sigurnosnim propustima koji su te napade omogućili. Pozivajući se ponovo na analogiju sa provalama u stambene objekte, kada smo rekli da je obijanje brave na ulaznim vratima *najčešći tip provale*, kao i da ih ima *više*, sada ćemo obraditi različite oblike malicioznih napada na web aplikacije, kao i statistički najčešći - Sql Injection.

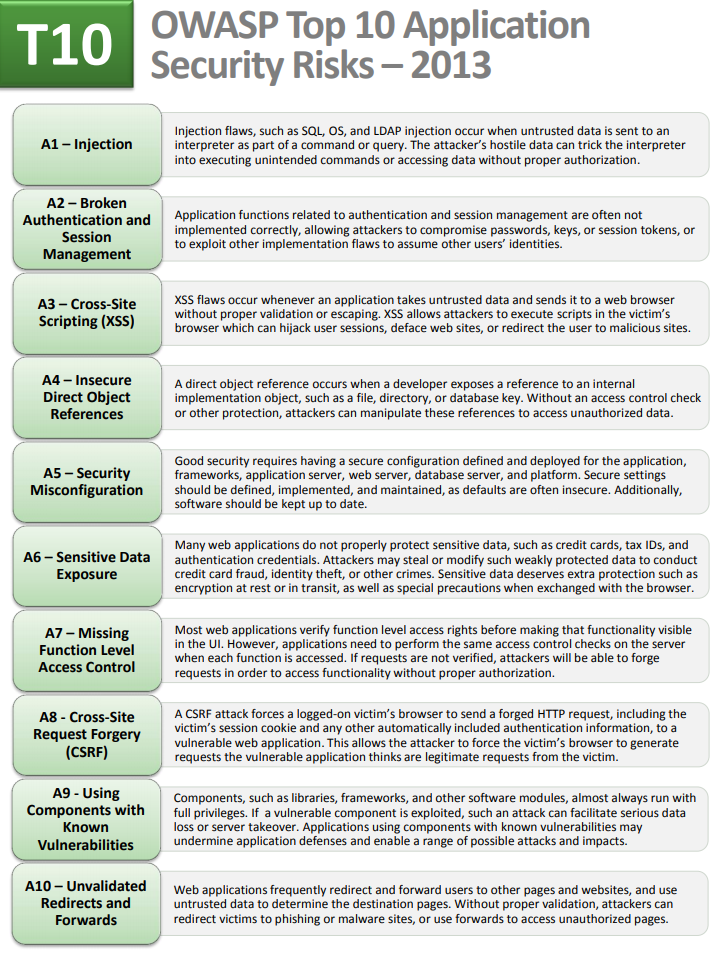
## *OWASP Top 10*

*„The Open Web Application Security Project“* (*skr*. “*OWASP“*) je neprofitna humanitarna organizacija koja se specifično bavi izučavanjem i poboljšavanjem sigurnosti softvera. Ova organizacija ima čitav asortiman korisnih alata za testiranje aplikacija, kao i dokumenata koji služe testerima kao velika pomoć prilikom testiranja. Razlog iz kojeg baš na ovom mjestu spominjemo „OWASP“ je činjenica da ova organizacija svake tri do četiri godine sastavlja listu top 10 najkritičnijih sigurnosnih propusta web aplikacija, na kojoj se nalaze razni tipovi napada, i to rangirani.

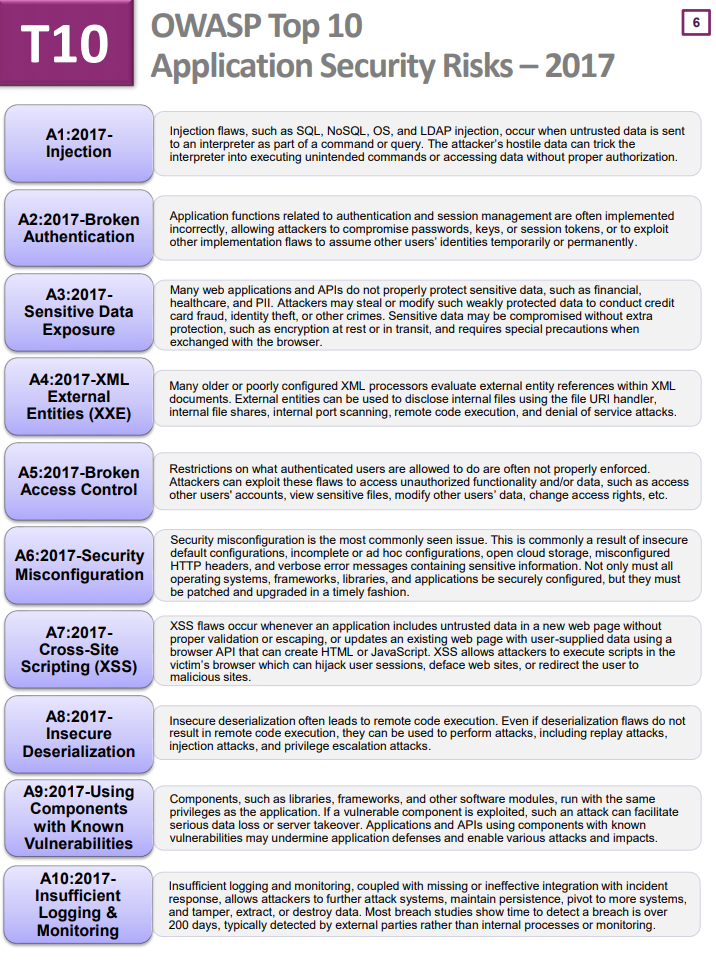
Najnovija OWASP Top 10 lista je iz 2017. godine. Uporedit ćemo listu iz 2017. sa dvije koje su joj prethodile - 2010. i 2013.



Slika 1 OWASP Top 10 2010.



Slika 2 OWASP Top 10 2013.



Slika 3 OWASP Top 10 2017.

Odmah se može primijetiti nešto veoma interesantno. Naime, u 7 godina koliko je trajalo prikupljanje podataka o sigurnosnim propustima, kao najveći problem i propust se uvijek javlja *Injection*. Injection-u će se u nastavku posvetiti posebna pažnja, s obzirom na njegovo alarmantno visoko rangiranje na tri uzastopne liste. *XSS* tipu napada će se također posvetiti pažnja, iako je na listi 2017. godine rangiran relativno nisko u odnosu na dosad.

## *Tipovi napada*

U nastavku obrađujemo detaljnije tipove napada koji su navedeni na listi iz 2017. godine, sa posebnim osvrtom na *Injection* tip napada. Treba napomenuti da pored svih navedenih napada postoji još mnogo vrsta (pri tome su neki opasniji od drugih) ali ipak je ovih 10 najozbiljnije i nanose najviše štete ukoliko se od njih ne odbrani.

## *Injection*

*Injection* tipovi napada nastaju kada napadač, putem HTML forme na web stranici, unosi maliciozni kod u tekstualna polja. Nevolja leži u tome što baza nezaštićene web stranice može biti u potpunosti *uništena.*

Svaki iole iskusniji haker je sposoban da pomoću Injection-a ukrade čitave tabele iz baze, da vrši upite iako nema nikakva prava pristupa, da briše sadržaj kao i razne druge manipulacije nad bazom.

Postoji više podvrsta Injection napada - SQL Injection, NoSQL Injection, LDAP Injection i drugi, od kojih je SQL Injection najrasprostranjeniji. Svaka podvrsta Injection napada direktno ovisi o tome kakvu bazu koristimo (relacionu, nerelacionu, objektnu itd).

U nastavku ćemo obrađivati samo SQL Injection napade, iz dva razloga. SQL Injection je najčešći tip Injection napada, pa je od itekakve važnosti da se dobro s njim upoznamo. Drugi razlog je činjenica da se svi tipovi Injection napada svode *na isto*, odnosno na isti način svaki od ovih napada nastaje i na isti način se od njih branimo, pa *nema potrebe* da obrađujemo svaki podtip ponaosob. Treba napomenuti da i SQLi ima svoje podvrste, odnosno da je to naziv za cijelu *familiju* napada.

Neke stvari koje napadač može da postigne pomoću SQLi su:

* ukrasti lične podatke o korisnicima iz baze (eng. *credentials*) te tako impersonirati te korisnike;
* dobiti pristup *svim* podacima na serveru na kojem se nalazi baza - ovakva krađa vlasnika aplikacije može oštetiti za *milione* u novčanim iznosima;
* manipulisati nad podacima u bazi - ubacivati nove podatke, ažurirati ili brisati postojeće (eng. *insert, update, delete*);
* brisati tabele pomoću DROP TABLE komande - čak ukoliko admin baze ima negdje *backup* podataka iz obrisane tabele, aplikacija neko vrijeme neće biti dostupna, što će imati negativan utjecaj na njen promet, i moguće je da admin nema *backup* za najnovije podatke koji su se nalazili u tabeli;

### Kako napad funkcioniše

Sada trebamo objasniti *kako tačno* dolazi do ovih napada, odnosno, koji *konkretno* sigurnosni propust omogućava napadaču da bez dozvole pristupa bazi. Ukoliko web stranica *direktno* koristi user input da vrši upite nad bazom, odnosno, ukoliko se podaci sa formi *ne provjeravaju ni na kakav način*, već se direktno ubacuju u SQL upite, *naša aplikacija je podložna SQL Injection napadu.*

U biti, web stranica je podložna SQL Injection-u na svakom mjestu na kojem tražimo neki vid unosa od korisnika, kao naprimjer username ili slična tekstualna informacija koja se treba provjeriti radi validnosti u bazi.

Analizirajmo par primjera:

[primjer 1]

[primjer 2]

[primjer 3]

U nastavku će biti objašnjeni koncepti i metode zaštite od ovog napada.

### Posljedice i metode zaštite

Napadači konstantno pretražuju internet kako bi pronašli SQLi propuste. Došlo je do toga da koriste alate koji automatiziraju potragu za manama u SQLi zaštiti i pokušavaju to iskoristiti za vlastitu finansijsku dobit.

S obzirom da je većina modernih aplikacija danas vođeno podacima (*eng. data driven*), i dostupno na internetu, SQLi propusti su veoma česti i rasprostranjeni.

Kada se jednom iskoristi SQLi propust, napadač može:

* ukrasti, promijeniti i uništiti osjetljive podatke kao što su nečiji podaci korišteni za identifikaciju (kao JMBG, šifre i usernames)
* koristiti privilegije na nivou aplikacije, baze ili čak operativnog sistema u nekim ekstremnijim slučajevima
* koristiti kompromizovani server na kojem se nalazi baza da napada druge sisteme koji se nalaze na istoj mreži

Postoji nekoliko ključnih načina da se zaštitimo.

**Paramtetrizovani upiti**

Programski jezici komuniciraju sa SQL bazama podataka pomoću database drivera. Driver omogućava aplikaciji da kreira i pokreće SQL upite na bazi, i da pritom dobija i manipuliše podacima, shodno potrebama. *Parametrizovani* upiti osiguravaju da se parametri odnosno ulazi u SQL upit tretiraju sigurno. Ovo je ubjedljivo najbolji i najjači način zaštite, koji će biti demonstriran u nastavku rada.

**ORM (Object Relational Mapping)**

ORM frameworks mapiraju SQL upite u objekte, tako da developer koji manipuliše nad podacima u bazi efektivno radi samo nad kreiranim objektom, a ne nad samim upitom. Ispod haube, ovi objekti koriste parametrizovane upite nad bazom. Dakle, ORM je suštinski alternativa parametrizovanim upitima, jer radi istu stvar, samo što programer nema obavezu da sam piše parametrizovane upite.

**Escaping Inputs**

Injection napadi se oslanjaju na to da će napadač prerano zatvoriti string koji se pojavljuje kao argument u SQL upitu (iz tog razloga često vidimo ‘ ili “ karaktere u SQL napadima), pa je ideja da se uradi *replace* karaktera ‘ ili “ sa ‘’ i “” respektivno. Bitno je napomenuti da je ovaj vid zaštite samo *dodatna* zaštita, te ni pod kojim slučajem ne može zamijeniti parametrizovane upite ili ORM framework. Ova metoda bi se uvijek trebala koristiti samo kao dodatno osiguranje da do probijanja neće doći, ali *nikada* kao primarni vid zaštite.

**Sanitizing Inputs**

Ovo je veoma dobra praksa za sve tipove aplikacija. Ideja je u tome da proispitujemo polja za šifru ili username na formi. Naime, kada napadač pokušava probiti zaštitu, kao što smo vidjeli u primjeru iznad [dodaj neki primjer bypassa zaštite sa onim fazonom or 1=1--], on koristi stringove poprilično sumnjivog izgleda. Ideja je da programer koji implementira validaciju input polja zabrani sve ulaze koji izgledaju sumnjivo, recimo pomoću regularnih izraza, osiguravanja da alphanumerička polja ne sadrže simbole i brisanje whitespacea. Kao i za escaping inputs, ovaj vid zaštite bi se trebao koristiti samo kao dodatni vid zaštite, nipošto kao primarni ili jedini. Razlog je nešto drugačiji - za sanitizing inputs je veoma teško procijeniti šta *tačno* znači “sumnjivog izgleda”. Teško je postaviti kriterij koji će odlučiti da li je neki ulaz sumnjiv a da pri tome ne bude ni preliberalan ni prerestriktivan.

**Dodatna zaštita**

Postoje i neki sekundarni vidovi zaštite koji nisu esencijalni kao dosad navedeni, ali se smatraju dobrom praksom i trebalo bi se uložiti vrijeme i trud da se i oni ispoštuju.

* održavati sve softverske komponente (biblioteke, plug-ins, frameworks i slične) ažurnim sa najnovijim prepravkama sigurnosnih detalja koje nam proizvođači nude
* koristiti minimalni broj privilegija - ukoliko je našem web siteu za neometan rad dovoljan samo SELECT izraz da dobija podatke iz baze, ne bismo joj trebali davati privilegije koje će omogućiti INSERT, UPDATE ili DELETE operacije. U mnogo slučajeva, ovo je moguće postići korištenjem rola, pri čemu se mora voditi računa da aplikaciji *nikad* ne dozvolimo da se poveže na bazu podataka kao administrator.
* ne bismo trebali koristiti dijeljenu bazu podataka za više web aplikacija ili stranica.
* osmisliti dobar mehanizam za prijavljivanja server-side grešaka. Ove greške se *nikada* ne bi trebale prikazivati korisnicima, iz razloga što su one pune tehničkih detalja koje bi napadač (koji je također korisnik) mogao iskoristiti da podesi svoje upite i probije zaštitu.

## *XSS*

*Cross-site Scripting* (*skr.* *XSS*) je napad izvršen sa klijentske strane. Napadač pokušava pokrenuti maliciozne skripte u web browseru žrtve tako što maliciozni kod ubaci u pouzdanu i legitimnu web stranicu ili aplikaciju. Napadnuta web stranica ili aplikacija postaje prenosno sredstvo za skriptu, te je dovodi do browsera.

Česte situacije gdje napadači pokušavaju izvesti XSS su forumi, message boards i razne web stranice koje dozvoljavaju javno ostavljanje komentara.

XSS napadi su mogući u VBScript, ActiveX, Flash, te čak i u CSS-u, međutim, daleko *najčešći* su u vidu JavaScripta. Razlog za ovo leži u činjenici da je JavaScript fundamentalan većini današnjih browsera.

### Podvrste XSS napada

Postoje sveukupno tri podvrste XSS napada:

* stored/persistent XSS
* reflected/non-persistent XSS
* DOM-based XSS

**Stored XSS**

Ovo je tip XSS napada koji je u stanju nanijeti najveću štetu klijentu koji je meta napdaču. Napadač će iskoristiti ovaj tip napada da ubaci maliciozni kod (najčešće pisan u JavaScriptu) u ranjivu web aplikaciju. Ako nismo osigurali validaciju podataka koje korisnici unose u naše HTML forme, ovaj kod se *direktno* unosi u neki vid trajnog čuvanja podataka, kao na primjer u bazu.

Česte situacije u kojima se pojavljuje su blog postovi i forumi.

**Reflected XSS**

Drugi najčešći tip XSS napada. U ovom slučaju, napadačev *payload* mora biti dio zahtjeva koji se šalje web serveru. Nakon ovoga se vraća nazad (end. *is reflected*) na način da HTTP response sadrži payload koji se je nalazio u requestu. Napadači često podmuklo koriste metode poput *phishing*, maliciozne linkove i sumnjive tehnike da namame klijenta da neznajući pošalje zahtjev serveru.

Ovi tipovi napada najčesšće imaju jednu konkretnu žrtvu koju žele da sabotiraju, a često se javljaju na društvenim mrežama.

**DOM-based XSS**

Ovo je najnapredniji i najkompleksniji tip XSS napada. Ono što ga omogućava jeste situacija u kojoj skripte web aplikacije koje rade na client-side zapisuju podatke (koje im korisnik daje u DOM (Document Object Model).

U ovom slučaju, payload se ne šalje na server uopšte, te ga zato WAFs (skraćeno od *Web Application Firewalls*) teško mogu detektovati. DOM objekti koji su najranjiviji su document.URL, location.hash i document.referrer.

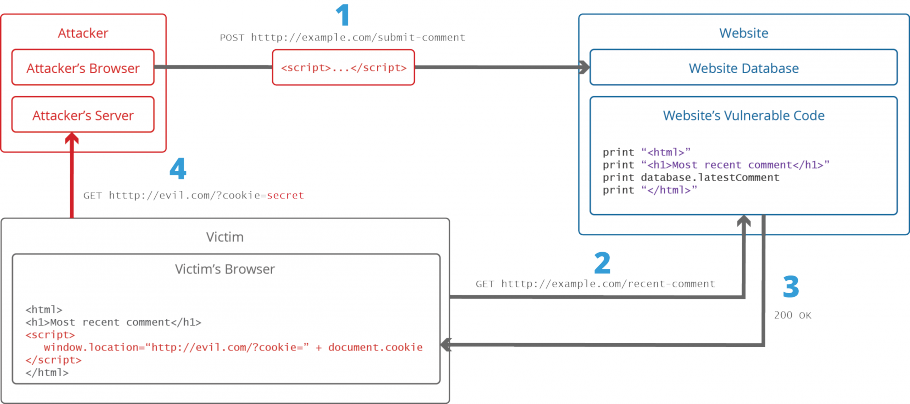
### Kako napad funkcioniše

Svaki tip XSS napada se sastoji iz dva osnovna koraka:

1. Pokrenuti maliciozni JS kod u korisnikovom browseru. Kako bi napadač ovo ostvario, on mora prvo pronaći način da ubaci neki malicioznu skriptu na neku web stranicu koju taj korisnik posjećuje.
2. Nakon što se to uradi, čeka se da korisnik pristupi stranici na kojoj se sada nalazi maliciozna skripta. Ako napadač *tačno* zna ko mu je meta, može koristiti *phishing[[2]](#footnote-2)* da pošalje maliciozni URL korisniku. Zbog prirode *phishing*-a, taj URL će korisnikovom browseru izgledati kao da je došao iz provjerenog izvora, iako nije.

Kako bi se prvi korak mogao ostvariti, potrebno je da stranica koju napadač napada direktno koristi input od korisnika u svom radu. Napadač onda unosi maliciozni string koji će biti korišten u web stranici, te koji će browser ugroženog korisnika tretirati kao *source code* te stranice.

Kada se stranica učita u klijentovom browseru, napadačeva skripta se izvršava i klijent je ne može zaustaviti.



Slika 4 Krađa cookies

1. Napadač ubaci *payload[[3]](#footnote-3)* u bazu tako što u ranjivu input formu ubaci maliciozni JS kod.
2. Klijent (ne znajući) pošalje zahtjev web serveru da mu dobavi tu stranicu.
3. Web server opsluži klijentov browser tako što mu pošalje unesenu malicioznu skriptu kao dio HTML body.
4. Klijentov browser izvršava malicioznu skriptu koja se nalazi u tijelu HTML dokumenta. U ovom slučaju browser šalje *cookie* napadačevom serveru.
5. Sve što napadač sada mora da uradi jeste da izvuče (eng. *extract*) klijentov *cookie* kada mu HTTP request stigne na server.
6. Napadač može koristiti ukradeni cookie za impersonaciju.

U prilogu “*Najčešći XSS vektori* [[4]](#footnote-4)” se nalazi popis najčešćih HTML tagova koji su korišteni za ostvarivanje XSS napada.

### Posljedice i metode zaštite

Šta tačno napadač može postići pomoću XSS napada?

* promijeniti sadržaj napadnute web stranice
* preusmjeriti korisnika na drugu stranicu (često takođe malicioznu)
* maliciozni JavaScript može pristupiti svim objektima kojima pristupa i web stranica, *uključujući user cookies*. Cookies sadrže informacije o sesiji svakog korisnika, te ukoliko se napadač uspije dokopati cookie-ja, može impersonirati korisnika čiji je cookie ukrao, te izvršavati sve akcije za koje taj korisnik ima privilegije.
* JavaScript u modernim browserima može pomoću HTML5 API pristupiti korisnikovoj geografskoj lokaciji, webcam-u, mikrofonu i čak nekim fajlovima iz korisnikovog file sistema.
* Moguće je kompromizovanom korisniku ubaciti trojanca u sistem.
* Moguće su krade identiteta.

U suštini, svi XSS napadi nastoje da napadnu neki web site pomoću inputa koje daje korisnik. Kod koji će napasti stranicu može varirati u svojoj kompleksnosti od jednostavne input forme, do komplikovanih JSON skripti, ovisno od tog šta napadač tačno želi da postigne. Glavna ideja zaštite je dakle, da je developer u svakom momentu svjesan da mu podaci dolaze iz vanjskih izvora kojima ne treba slijepo vjerovati.

Shodno tome, nameće se nekoliko metoda zaštite od XSS napada.

**Filtriranje**

Najjednostavnija ideja jeste da se sav input koji se uzima od korisnika provuče kroz neki vid filtera koji će analizirati i ukloniti sve potencijalno opasne ključne riječi, poput <script> taga, JS komandi, CSS stilova (koji takođe mogu biti maliciozni) i generalno svih HTML markupa koji sadrže event handlers.

Mnogi web developeri sami razvijaju vlastite filtere na server-side. Najčešće se razvijaju u PHP-u, mada se mogu koristiti i drugi jezici. Filter traži ključne riječi koje mu zada developer i mijenja ih praznim stringovima.

Moglo bi se pomisliti da su regularni izrazi dobra ideja za podlogu filtera, međutim pokazalo se da u praksi to nije tako. Razlog leži u činjenici da su napadači u većini slučajeva puno iskusniji od developera, te da ovako jednostavne filtere (kao te koji su bazirani na regularnim izrazima) mogu bez većih poteškoća zaobići. Iz tog razloga se preporučuje da se koristi neka temeljito istestirana biblioteka.

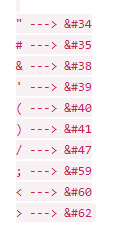
Biblioteka koje se koriste u ovu svrhu ima mnogo, a ono što određuje izbor biblioteke jeste *backend* tehnologija koja je korištena za razvoj aplikacije. Uvijek nam je u interesu da odaberemo onu biblioteku koja je ažurna i koja stalno updateuje svoje sigurnosne metode i tehnike.

Jedina mana ovog pristupa je što će prestrog filter zabraniti unos nekih legitimnih riječi, pa se o tome treba povesti računa.

**Escaping**

*Escaping* je tehnika pomoću koje klijent kaže browseru da sve što mu se pošalje kao odgovor treba da se tretira kao podatak, i ni na kakav drugi način. Ono što se ovom metodom postiže je to da browser *ne* izvršava skriptu koju je napadač ubacio u stranicu koju smo zatražili.

U HTML-u je escaping relativno jednostavan za izvesti. Sve što treba da uradimo jeste da opasne simbole zamijenimo njihovim escape code-om i naš posao je gotov.

  
Escape kodovi za “opasne” simbole

Ovo će browseru efektivno reći da se ovi simboli *ne trebaju* izvršavati, već samo parsirati kao najobičniji tekst, što napadačevu skriptu čini uzaludnom.

Nešto teže je uraditi escaping za JS, CSS i XML, i nažalost, to je upravo ono što je neophodno da se uradi kako bi naša zaštita bila potpuna. Veoma je teško ručno uraditi escaping za sve tri navedene tehnologije, i upravo se iz tog razloga uvijek preporučuje da se koriste isprobane escaping biblioteke kao ESAPI (omogućio OWASP) i AntiXSS (omogućio MiscroSoft).

Još jedan izazov postoji s ovom metodom, a to je kako znati koliko escaping-a je previše escaping-a? Naime, problem se javlja u tome što nikada ne možemo uraditi escaping *za sve*, jer u tom slučaju naša stranica više neće biti funkcionalna - sav JS kod bi bio escapiran, kao i HTML i CSS.

Postoje određene preporuke kako bi se izbjegao ovaj problem:

* koristiti HTML escaping kada se sumnjiv kod ubaci između početnih i završnih HTML tagova
* koristiti JavaScript escaping kada se sumnjivi podaci ubace u jednu od skripti ili na neko drugo mjesto gdje se JS kod može neometano izvršiti. Ovo uključuje neke atrivute kao što je style i sve event handlers.
* koristiti CSS escaping kada se sumnjivi podaci ubace u jedan od CSS stilova. Kao što smo vidjeli ranije, mnogi CSS atributi se mogu zloupotrijebiti u ovu svrhu.

## *Broken Authentication*

Prilikom ovog napada napadač nastoji da zaobiđe metode autentikacije koje web site koristi, pri čemu mu je cilj da ukrade pristupne podatke (*eng. credentials*) osobe koja ima određene privilegije. Obično je ta osoba administrator baze, direktor kompanije ili slično.

*Broken Authentication* se ispoljava u sljedećim sigurnosnim propustima:

* web site dozvoljava upotrebu defaultnih ili slabih šifri (poput “Password1” ili “admin”) koje napadač može veoma lagano pogoditi i tako pokupiti pristupne podatke korisnika
* web site koristi slab ili neefektivan *recovery* zaboravljenih podataka, kao naprimjer *knowledge-based answers* (postavljanje ličnih pitanja koja će “dokazati” da ste upravo Vi prava osoba)
* ne posjeduje ili ima neefektivnu multi-faktor autentikaciju (autentikacija u više koraka)
* u URL-ove se upisuje id sesije
* ne koristi se rotacija id-jeva sesije nakon logina
* dozvoljava brute-force automatizovane napade
* ne koristi dovoljno dobar algoritam za heširanje šifri
* id-jevi sesije se ne poništavaju pravilno nakon log out-a

### Posljedice i metode zaštite

Posljedice, kao i u većini drugih napada, se ogledaju u krađi podataka korisnika koji u mnogim slučajevima ni ne zna da su mu podaci ukradeni.

Slijedi nekoliko primjera kako izgledaju Broken Authentication napadi, kao i moguća rješenja.  
  
**Credential stuffing**

Pretpostavimo da je napadač nekad ranije uspio probiti zaštitu na bazi neke stranice/servisa te je, nažalost, uspio pokupiti ogromnu količinu pristupnih podataka. Obično se radi o nekoliko stotina hiljada ili čak miliona redova iz tabela u bazi.

Pretpostavimo sada da napadaču nije dovoljno što je probio zaštitu *jedne* web stranice, kako to obično i jeste slučaj, već želi *više*.

Kada napadač (obično uz pomoć nekog programa) brute force metodom testira sve pristupne podatke koje je ukrao iz neke baze da pristupi *drugim* stranicama koje imaju ili nemaju veze sa onom iz koje je napadač izvorno pokupio podatke naziva se *credential stuffing.* Obično se radi o *ogromnoj* količini stranica na koje se velikim brzinama pokušava pristupiti svim podacima koje napadač posjeduje.

Što se tiče zaštite od credential stuffing, ona se radi iz dvije perspektive - iz perspektive korisnika i perspektive firme koja želi da svoje klijente zaštiti od ovakvih napada.

Jedino što kao korisnik trebamo poduzeti da se zaštitimo je da se pobrinemo da na svakoj stranici na kojoj imamo profil koristimo različitu šifru. Ovo će onemogućiti credential stuffing na klijentskoj strani. Također, kao mjeru predostrožnosti je uvijek dobro omogućiti two-factor autentikaciju (autentikaciju iz dva koraka).

Kompanija koja želi klijente zaštiti od ovog napada ima nešto teži zadatak, s obzirom da je credential stuffing rezultat probijanja zaštite *druge* kompanije. Jedan način zaštite jeste platiti dobar Bot Management System čiji je zadatak da osluškuje da li maliciozni botovi unose podatke na stranicu.

**Url rewriting**

Recimo da želimo da rezervišemo smještaj za ljetovanje na Havajima. Neka kompanija kojom smo dosada rezervisali odmore nudi popust za putovanje koje nas interesuje, i recimo da želimo da obavijestimo prijatelje o tome.

Ukoliko web site koji smo posjetili i čiji popust želimo iskoristiti ubaci id sesije u url, nastaje problem. Takav url bi mogao izgledati otprilike ovako:

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

Ukoliko ovakav url pošaljemo prijateljima, te oni kliknu na njega, oni će dobiti id naše sesije, što se *ni slučajno* ne smije desiti, jer sesija čuva strogo povjerljive informacije o korisniku koji je koristi. Kada neko ima id tuđe sesije u svojim rukama, on iz nje može izvući ogromnu količinu osjetljivih podataka - u ovom konkretnom slučaju, u sesiju bi, pored ličnih podataka, mogao biti i upisan broj kreditne kartice (pošto se radi o online rezervaciji i online plaćanju). Propuste ove vrste ne smijemo dozvoliti.

**Šifre nisu dobro heširane**

Pretpostavimo da je napadač uspio da se dokopa baze podataka. Ukoliko nismo koristili pravilan algoritam za heširanje, napadač je u stanju da vidi apsolutno sve i jednu šifru koja je evidentirana u bazi u njenoj *izvornoj* formi.

Šifre bi *uvijek* trebale biti heširane, i od ovog pravila se *nikada* ne bi trebalo odstupati.



Slika 5 Primjer heširane šifre pomoću MD5 hash algoritma

Kao što se vidi na slici iznad, napadač iz pravilno heširane šifre ne može dobiti *ma kakvu* korisniku informaciju, jer su algoritmi za heširanje i kriptografiju po svojoj prirodi takvi da se ne može naći *inverzni* algoritam, koji bi iz heširane informacije dobio izvornu.

**Lahko predvidljivi login podaci**

Neophodno je forsirati korisnike da koriste šifre koje je teško pogoditi, bez obzira na korištenje algoritama za heširanje. Čak ukoliko koristimo algoritam za heširanje šifri, to neće spriječiti napadača da prosto isproba nekoliko “tipičnih” šifri i usernames kao npr. “admin”, “šifra”, “123” i slično.

Potrebno je pojačati validaciju polja za šifre što je više moguće na sljedeće načine:

* praksa je pokazala da bi minimalna dužina šifre trebala biti osam (8)
* šifre bi trebale biti kombinacija alfanumeričkih karaktera (dobro bi bilo zahtijevati barem jedno veliko slovo, kao i barem dva broja)
* onemogućiti login nakon određenog broja propalih pokušaja pogađanja šifre

**Dodatna zaštita**

Postoji još mnogo vidova zaštite i mjera predostrožnosti koji bi se mogli (i trebali) implementirati. Neki od njih su:

* implementirati multi-faktor autentikaciju gdje god je to moguće
* prilikom deployanja aplikacije *ni slučajno* deployati sa bilo kakvim *credentials* - pobrinuti se da u bazi ne postoji niti jedan jedini login credential neposredno prije deployanja
* koristiti pouzdan algoritam koji randomizira id-jeve sesija kako bismo bili što više sigurni da se oni ne mogu pogoditi
* nikada ne upisivati id sesije u url-ove
* postaviti rok isticanja (*eng. timeout*) na sesije
* testirati šifru koju korisnik unese na osnovu listi tipa “1000 najlakših šifri za pogoditi” koje se mogu lagano naći na Internetu

U nastavku poglavlja 2 ukratko opisujemo još nekoliko napada sa OWASP Top 10 listi.

## Preostali napadi

Radi nedostatka prostora neće se ulaziti u detalje za svaki od napada koji će biti opisan u ovoj sekciji. Navodiće se ime i kratki opis napada.

**Sensitive Data Exposure**

Mnoge web aplikacije i API-ji ne štite dovoljno podatke koji bi trebali biti strogo povjerljivi, kao finansijske informacije, informacije o zdravstvu, osiguranju i slično. Napadači koriste ovu činjenicu da ukradu ovakve informacije bez prevelikog truda. Potrebno je dobro paziti na to kako ćemo prenositi ovako osjetljive informacije, ko ih smije vidjeti, te kako su pohranjene u bazi.

**XML External Entities**

Nastaje u aplikacijama koje parsiraju XML inpute. Ukoliko neki XML input sadrži referencu na vanjski entitet[[5]](#footnote-5), a pri tom koristi slab parser XML-a, ili ako aplikacija prihvata XML direktno i takvog ga unosi u XML dokument, aplikacija je pod rizikom.

**Broken Access Control**

Uskon vezano sa autorizacijom neke web aplikacije. Mnoge stranice ne implementiraju dovoljno dobro uloge/role, te koje privilegije ima koja uloga. Napadači mogu ovo iskoristiti da vide podatke koje inače ne bi smjeli, da mijenjaju tuđe podatke, promijene prava pristupa stranici i dr.

**Insecure Deserialization**

Serijalizacija je proces konvertovanja podataka u format u kojem se oni mogu zapisati u disk (npr. XML i JSON). Deserijalizacija je obratno, konvertovanje podataka iz takvog formata u objekat. Nevolja nastaje kada se pokušava uraditi deserijalizacija podataka koji nisu došli iz pouzdanog izvora.

**Insufficient Logging And Monitoring**

Kada se ne prave sistemski logovi dovoljno često, napadač može da to iskoristi u svoju korist na način da pravi probleme aplikaciji za koje zna da će proći *nedetektovani*. Nije dovoljno samo implementirati često logiranje dešavanja unutar aplikacije, već je također potrebno osposobiti mehanizam koji će, uz pomoć developera koji su zaduženi za održavanje i nadgledanje, sastavljati izvještaje o ponašanju aplikacije, te biti u stanju da detektuje i obavijesti o uočenim sumnjivim dešavanjima.

Bitno je naglasiti da postoji još mnogo tipova napada, ali radi opširnosti same tematike ovaj rad ih neće moći sve pokriti. Iz tog razloga ovdje se prekida izlaganje o napadima i mehanizmima odbrane i kreće se sa poglavljem u kojem se pravi kratki uvod u tehnologiju koja će biti korištena za razvoj ogledne aplikacije.

# *Analiza biblioteke React JS*

Kao nastavak izlaganja pravimo kratki uvod u tehnologiju koja je odabrana za razvoj ogledne aplikacije, na kojoj će biti demonstrirana zaštita od SQLi i Broken Authentication.

## *Uvod u React JS*

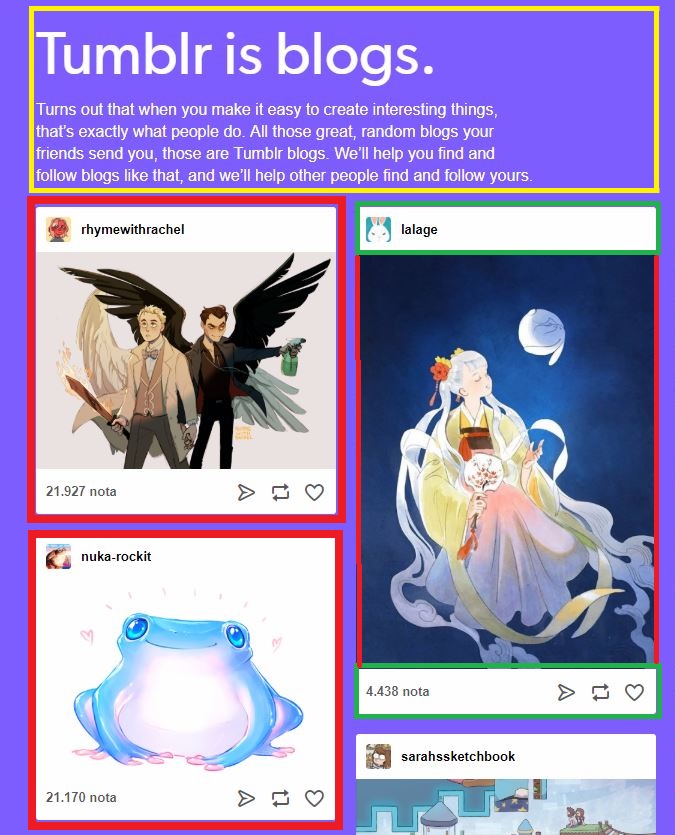
React JS je deklarativna, efikasna i fleksibilna JavaScript biblioteka za pravljenje *frontenda* web aplikacija. Koristi deklarativni pogled koji olakšava kodiranje, na način da kod čin predvidljivim i jednostavnim, te olakšava debuggiranje.

Ono što je specifično za React kao tehnologiju jeste činjenica da on omogućava razvoj aplikacija koje su sklabilne i veoma brze jer su bazirane na konceptu koji se zove *single page applications.* Ovaj koncept podrazumijeva da aplikacija koja se razvija ima samo jednu stranicu, pri čemu se sadržaj aplikacije mijenja dinamički, u ovisnosti od zahtjeva korisnika.

Pri razvoju React-a korišten je drugačiji pristup od onog koji je korišten u kreiranju Angulara i ostalih MVC okruženja. U tim okruženjima moguće je izgraditi cijelu aplikaciju, bez potrebe za dodatnim bibliotekama, jer pružaju HTML templates, kontrolere i servise. S druge strane, React je fokusiran samo na *view* sloj aplikacije, pa se za izgradnju funkcionalne aplikacije moraju koristiti i druge biblioteke.

## *Komponente, props i state*

Razvoj aplikacija u Reactu se svodi na dijeljenje jedne velike cjeline na manje dijelove koji se zovu *komponente.* Svaka komponenta se zasebno kreira i importuje u aplikaciju po potrebi, što olakšava rad i podstiče ponovnu iskoristivost (*eng*. *reuse*) koda, što je uvijek dobra stvar.



Slika 6 Primjer razlaganja dijela stranice na komponente

Na slici iznad je prikazan primjer podjele stranice na komponente. Sve što je označeno na slici se može smatrati komponentom koja je zadužena za rad sa malim dijelom HTML-a, CSS-a, kao i logike koja manipuliše ponašanjem komponente.

Implementacijski gledano, komponente se mogu kodirati na dva načina:

* kao funkcije
* kao klase

**Funkcijska implementacija**

function Item = () => { return (); }

Kada se komponenta implementira kao funkcija, za tu komponentu se u React žargonu kaže da je “glupa”. Razlog za to ćemo vidjeti tek nakon što se upoznamo sa *state* objektom.

**Klasna implementacija**

class Item extends React.Component {

constructor(){

super();

this.state = {}

}

render() {

return ();

}

}

Za razliku od funkcijskih komponenti, svaka klasna komponenta mora imati render metodu koja se automatski poziva nakon svake izmjene komponente, bilo da se izmjena dešava tokom razvoja, bilo da se dešava dinamički. Pored ove metode, developer može u klasu dodavati metode koje želi, ovisno od potrebe.

Konstruktor, kao i kod običnih klasa, služi za inicijalizaciju komponente.

Komponente se mogu ugniježdavati, kao na slici //broj//. Pri tome, komponenta koja sadrži druge komponente se naziva *parent* komponenta, a komponente koje se nalaze u nekoj parent komponenti *child* komponente.

*Props* su parametri koje svaka parent komponenta može proslijediti svojim child komponentama. Ovi parametri mogu biti varijable ili *pure[[6]](#footnote-6)* JavaScript funkcije. Props su *immutable*, odnosno ne mogu se mijenjati nakon što se proslijede child komponenti.

*State* je objekat koji sadrže *isključivo* klasne implementacije komponenti. State objekat je ekvivalent atributima klase u objektno-orijentisanom programiranju.

Upravo iz ovog razloga se funkcijska implementacija React komponenti naziva “glupom” - ona uvijek radi jednu te istu stvar, sa vrijednostima koje ne mogu dinamično mijenjati svoju vrijednost. Klasne komponente se dinamično mijenjaju, u ovisnosti na isprogramirani odgovor na događaje, upravo zahvaljujući činjenici da one predstavljaju klase, čiji atributi mogu mijenjati vrijednost.

## *Virtuelni DOM i JSX*

Za razliku od većine JavaScript okruženja za razvoj *frontend*-a, React ne koristi browserow DOM, nego gradi virtuelnu reprezentaciju DOM-a, u obliku stabla JavaScript objekata. Pisanjem koda u React-u se ne manipuliše DOM-om direktno, nego kreiranim virtualnim DOM-om, što React čini *neuporedivo* brzom tehnologijom. Za prikaz promjena u stvarnom browserovom DOM-u je takođe zadužen React.

Pri manipulaciji DOM-a potrebno je prvo locirati element, a zatim direktno promijeniti taj element koristeći atribut *innerHTML*. Ovakav način razvoja može biti problematičan zbog otežanog praćenja trenutnog stanja svakog od elemenata i zbog toga što je manipulacija DOM-a skupa operacija koja može usporiti aplikaciju.

Ovdje uskače upotreba virtualnog DOM-a.

Reactova implementacija virtualnog DOM-a dolazi sa bitnim optimizacijama performansi. Inžinjeri Facebook-a koji su radili na razvoju React-a su uspjeli da smanje kompleksnost algoritma koji se koristi za ažuriranje virtualnog DOM-a sa O(n3) na O(n)[[7]](#footnote-7). Pored toga što korištenje virtualnog DOM-a povećava performanse modifikacije DOM-a, ažuriranje virtuelnog DOM-a je olakšano, jer se događa samo onda kada se state objekat promijeni, kao što je već pomenuto.

JSX (eng. *JavaScript Syntax Extension*) predstavlja sintaksu koja omogućava pisanje React komponenti pomoću HTML-a. Upotreba JSX-a nije obavezna pri kreiranju React komponenti, ali pruža određene prednosti u odnosu na korištenje čistog JavaScript-a. Jedna od glavnih prednosti jeste optimizacija pokretanja koda prilikom prevođenja JSX-a u JavaScript koristeći pretprocesor. Pored toga, omogućeno je pisanje JavaScript koda unutar JSX-a.

# *Specifikacija web aplikacije #tech*

U ovom poglavlju će biti izložena specifikacija ogledne aplikacije na kojoj ćemo implementirati zaštitu od SQLi i Broken Authentication.

## *Namjena*

Ova aplikacija ima nekoliko namjena. Prva je da ilustrira korištenje React Native okruženja za kreiranje izgleda web aplikacija, te da se objasni koje su beneficije korištenja ovog okruženja. Druga namjena je da se pokaže kako napadač može iskoristiti sigurnosne propuste web aplikacije da ukrade informacije iz baze. I konačno, treća namjena je da se na primjeru ilustruje kako se web aplikacija pravilno štiti od SQLi napada.

## *Pregled sadržaja*

Aplikacija je zamišljena da se sastoji iz poprilično simplističnog interfejsa i veoma malo funkcionalnosti.

* Home page - stranica na kojoj se korisnik može ulogovati na svoj profil.
* Register - stranica koja služi za registraciju novih korisnika.
* Profile - služi da korisnik može vidjeti svoje podatke i/ili mijenjati ih.
* Cart - služi da korisnik aplikacije može da vidi items u svojoj korpici koje može ukloniti iz iste ili nastaviti dalje s kupovinom.
* Shop - stranica na kojoj se nalazi search bar i koja služi za pretragu dostupnih proizvoda.

## *Funkcionalni zahtjevi*

* Login - Da bi se korisnik prijavio potrebno je da bude registrovan. Prijava se vrši ukucavanjem email-a i šifre. Pri tome, i email i šifra se moraju validirati.
* Registracija - Ime, prezime, email, adresa, broj telefona, broj kreditne kartice, password i confirm password. Sva polja su obavezna, s tim da šifra mora imati barem jedan broj i biti duga barem 8 karaktera, te mora biti identična šifri ukucanoj u confirm password polju.
* Odjava - Korisnik ima mogućnost da se odjavi sa aplikacije klikom na log out dugme.
* Pregled proizvoda - Korisnik je u mogućnosti da pretražuje sve dostupne proizvode pomoću search bara pri čemu se svi pronađeni proizvodi pojavljuju na istoj stranici ispod search bara.
* Add to cart - Kada korisnik pronađe proizvod koji ga interesuje, može ga dodati u svoju korpicu klikom na add to cart dugme.
* Pregled korpice - Mogućnost da korisnik vidi sve proizvode u svojoj korpici. Ukoliko se korisnik log out-uje sa stranice, svi predmeti u korpici ostaju tu do sljedeće prijave.
* Pregled profila - Korisniku se omogućava da u svakom trenutku pristupi svojim ličnim podacima i da ih mijenja ukoliko to želi.

## *Akteri*

* Korisnik - Osoba koja se registruje na stranicu i ima mogućnost korištenja svih funkcionalnosti stranice.
* Admin - Specijalni korisnik koji ima mogućnost dodavanja, mijenjanja i brisanja proizvoda i korisnika iz aplikacije.
* Gost - Potencijalni budući korisnik aplikacije. Kada se gost registruje na sistem, postaje korisnik.

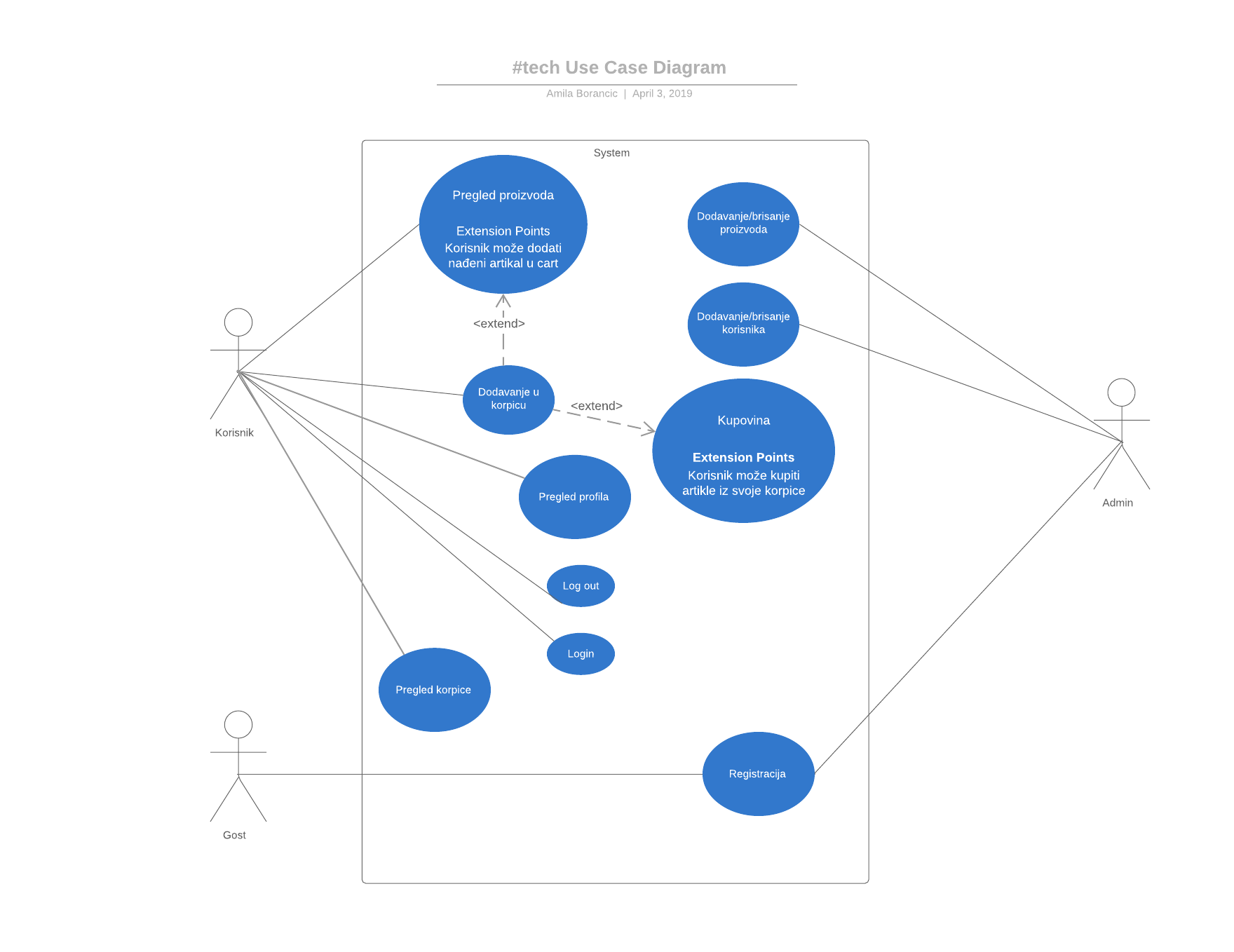
## *Nefunkcionalni zahtjevi*

* Zahtjevi performansi
  + aplikacija treba da bude brza te da je veliki broj prijavljenih korisnika ne usporava.
* Sigurnosni zahtjevi
  + Prilikom registracije i logina potrebno je validirati sve unesene podatke.
  + Šifru je potrebno heširati.
  + Šifra treba da ima minimalno 8 karaktera, od kojih je barem jedan karakter broj.
  + Potrebno je zaštititi aplikaciju od SQLi.
* Kvaliteta softvera
  + Upotrebljivost - Potrebno je da je UI jednostavan i jasan za korištenje, kako bi se novi korisnici što lakše snašli.
  + Pouzdanost - Sistem je potrebno osigurati od padova (eng. *failures*). Mora biti pouzdan i tolerantan na korisničke greške.
  + Testabilnost - Sistem treba biti implementiran na način da olakša testiranje i ispravljanje grešaka.
  + Održavanje - Potrebno je da sistem omogući jednostavno proširivanje aplikacije dodavanjem novih funkcionalnosti i/ili modifikovanjem postojećih.

## *Tehnologija za razvoj*

Aplikacija će biti razvijena pomoću React JS tehnologije za *frontend* i Node JS i MySql baza za *backend*.

## *Use Case dijagram*



Slika 7 Use Case dijagram za oglednu aplikaciju

# *Sigurnost aplikacije #tech*

Kako je ranije naglašeno, #tech, pored navedenih funkcionalnosti, treba da implementira zaštitu od SQLi napada, kao i validaciju šifre (heširanje, minimalno osam slova).

Kako bismo se zaštitili od SQLi, biće korišteni ranije objašnjene parametrizovani upiti.

# *Zaključak*

# *Prilog: Najčešći XSS vektori*

<script> tag

Ubjedljivo najčešći i najdirektniji XSS payload. Pomoću <script> taga referenciramo neki eksterni JS kod ili ga sami pišemo.

JS events

onload i onerror events se mogu koristiti unutar raznih tagova, što ih čini itekako popularnim XSS payloadom.

<body> tag

U <body> tagu napdač može pomoću JS events ili atributa body taga da ubaci maliciozni kod. Može koristiti atribute koji sami po sebi nemaju nikakve veze sa samim JavaScriptom, kao npr background.

<img> i <input> tag

Neki browseri su u stanju izvršiti JS kod koji se nalazi u src atributu <img> i <input> tagova koji za tip imaju type=”image”.

<iframe> tag

<iframe> tag omogućava da se jedna HTML stranica “ugradi” (eng. *embed*) u stranicu koja se trenutno prikazuje klijentu. Oni se konkretno koriste za *phishing*, odnosno za namamljivanje korisnika na neki maliciozni URL.

<link> tag

Ovaj tag se inače koristi za referenciranje eksternih skripti, te se kao takav može koristiti da se

<table> i <div> tag

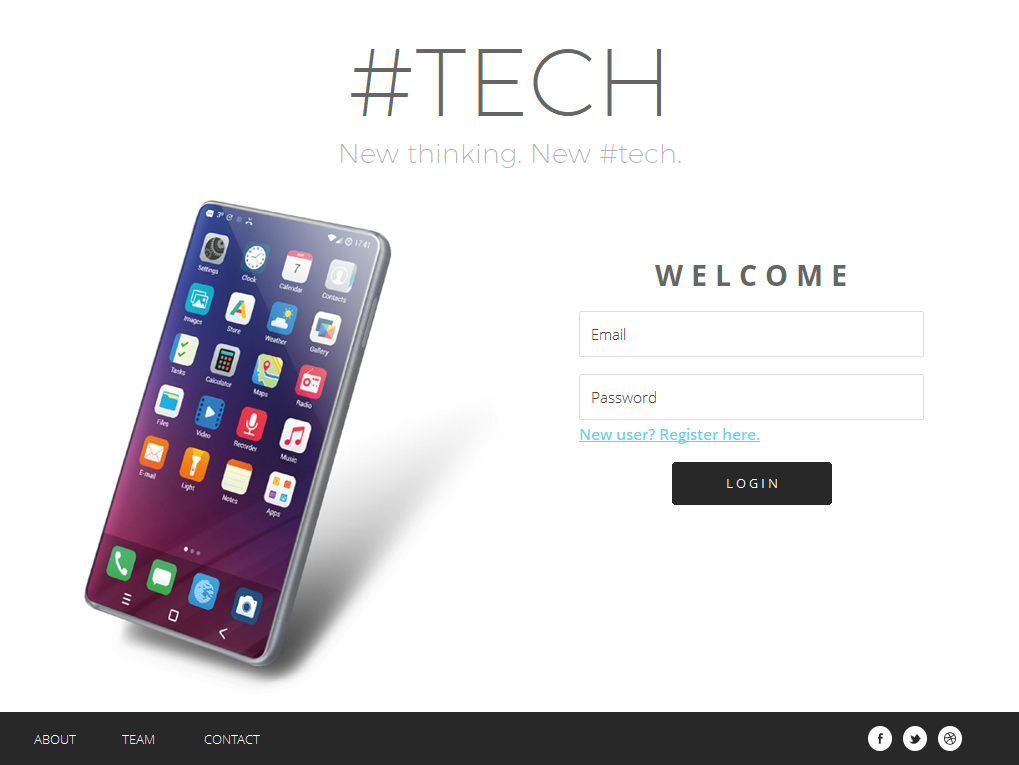
background atribut <table> i <div> tagova se može iskoristiti da se referencira skripta umjesto neke slike.

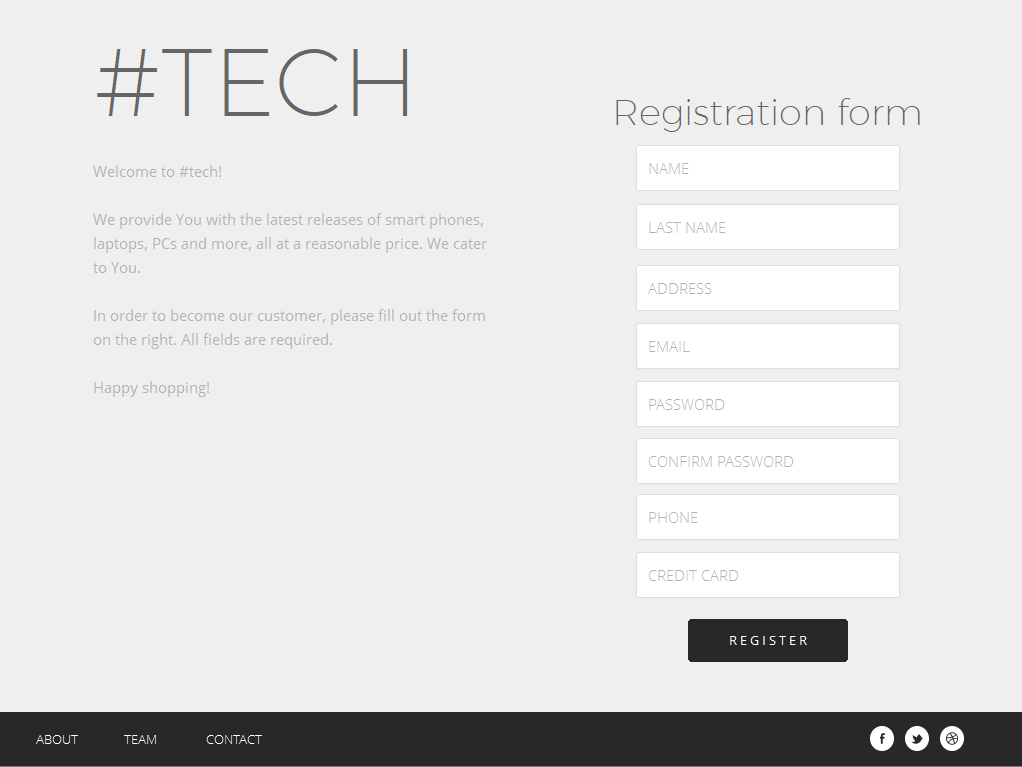
<object> tag

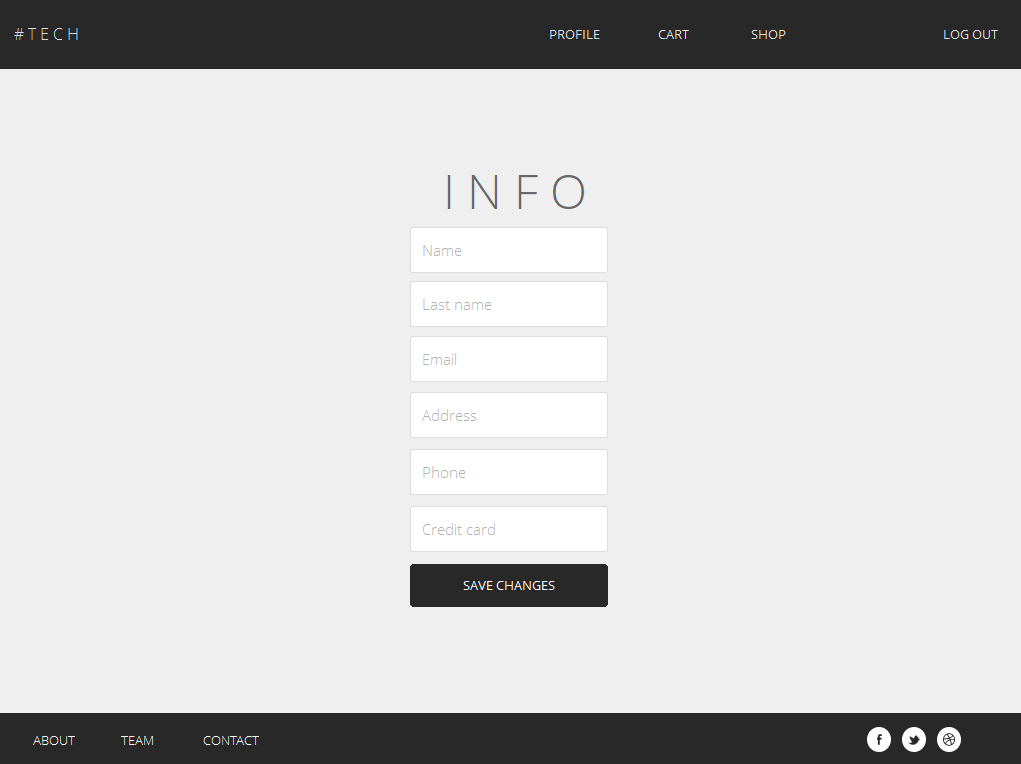
Ovaj tag se može iskoristiti za referenciranje eksterne skripte postavljajući data atribut na URL pomenute skripte.

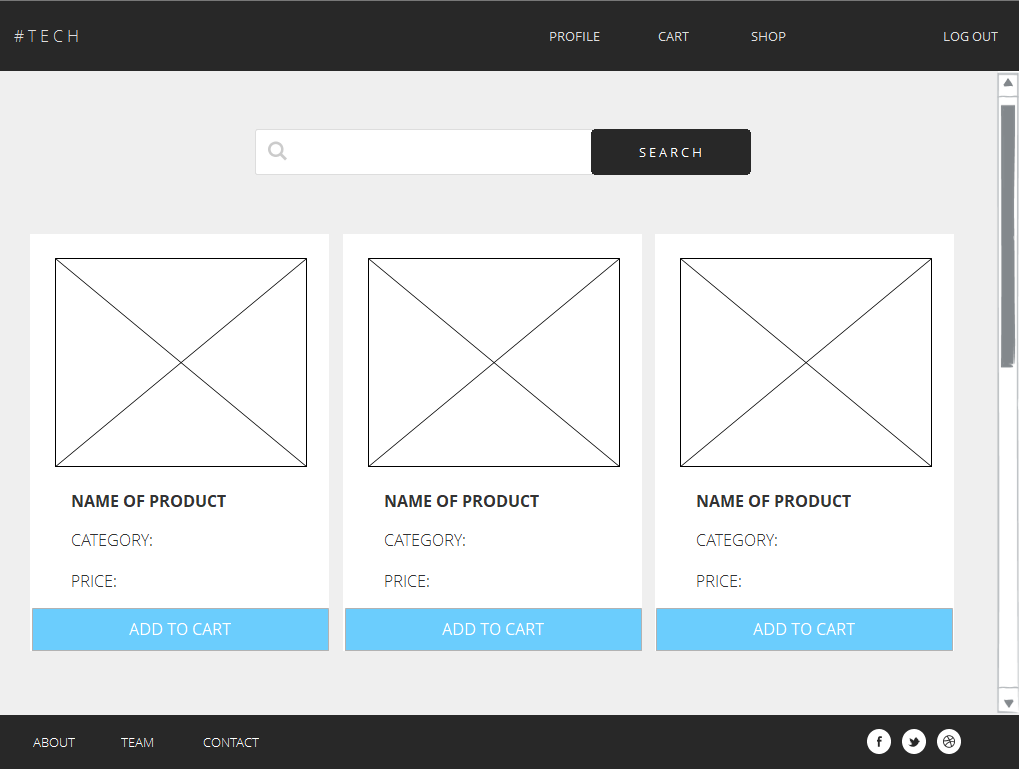
# *Prilog: Prototipi*

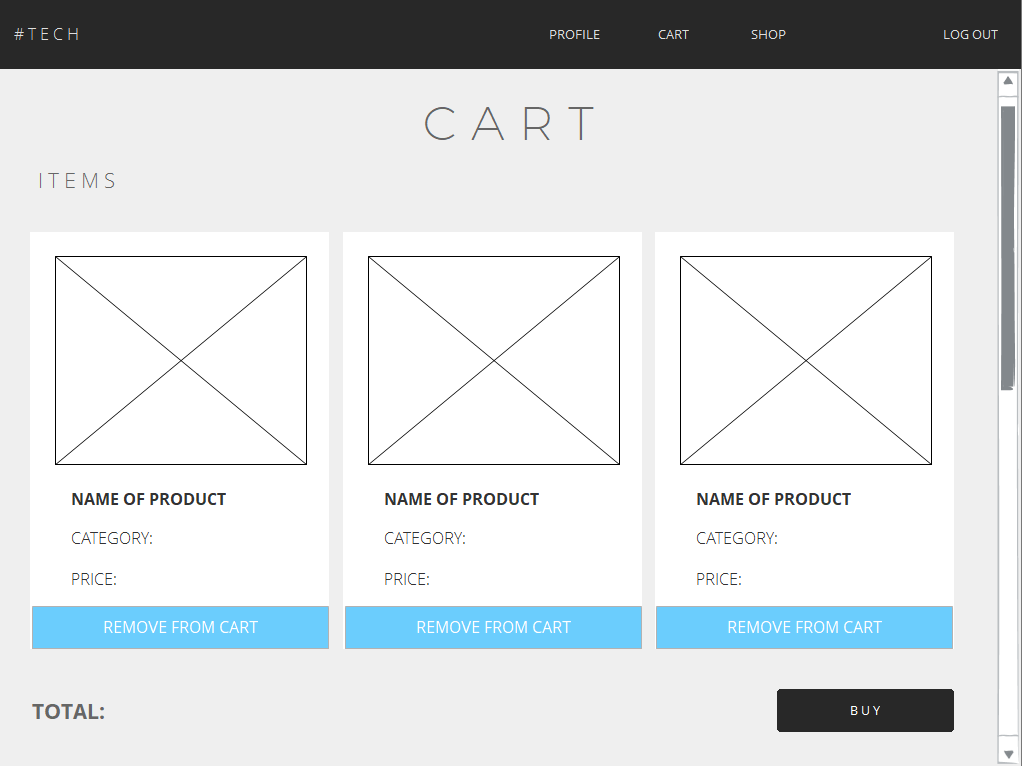
U ovoj sekciji će biti prikazan dizajn interaktivnog prototipa web aplikacije #tech. Prototipi su rađeni u alatu JustInMind.











# *Reference*

[1] Generalno oko sigurnosti: <https://www.goodfirms.co/glossary/web-security/>

[2] OWASP: <https://www.owasp.org/index.php/Main_Page>

[3] OWASP Top 10: <https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf>

[4] Statistika o provalama: <https://reolink.com/home-burglary-crime-statistics/>

[5] SQL Injection: <https://www.w3schools.com/sql/sql_injection.asp>

[6] SQL Injection: <https://en.wikipedia.org/wiki/SQL_injection>

[7] SQL Injection: <https://www.acunetix.com/websitesecurity/sql-injection/>

[8] MyFitnessPal security breach: <http://www.rts.rs/page/magazine/sr/story/1882/tehnologija/3089836/kradja-licnih-podataka-miliona-korisnika-popularne-aplikacije.html>

[9] MyFitnessPal: <http://fortune.com/2019/02/14/hacked-myfitnesspal-data-sale-dark-web-one-year-breach/>

[10] Anthem security breach: <http://balkans.aljazeera.net/vijesti/anthem-jedna-od-najvecih-krada-podataka>

[11] In-depth research of Anthem security breach: <https://www.bankinfosecurity.com/new-in-depth-analysis-anthem-breach-a-9627>

[12] Biggest Data Breaches of the 21st century <https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html>

[13] Yahoo security breach: <https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html>

[14] Target: <https://krebsonsecurity.com/2014/02/target-hackers-broke-in-via-hvac-company/>

[15] Target: <https://www.csoonline.com/article/2134248/target-customers--39--card-data-said-to-be-at-risk-after-store-thefts.html>

[16] Broken Authentication: <https://hdivsecurity.com/owasp-broken-authentication-and-session-management>

[17] Credential Stuffing: <https://www.cloudflare.com/learning/bots/what-is-credential-stuffing/>

[18] Insufficient Logging: <https://www.immuniweb.com/blog/OWASP-insufficient-logging-and-monitoring.html>

# *Popis slika*

1. U radu su korišteni slijedeći dokumenti: Izjava autora koju koristi Elektrotehnički fakultet u Sarajevu; Izjava o autentičnosti završnog rada Centra za interdisciplinarne studije – master studij „Evropske studije”, Izjava o plagijarizmu koju koristi Fakultet političkih nauka u Sarajevu. [↑](#footnote-ref-1)
2. *Phishing* je napad krađe prilikom kojeg se napadač pretvara da je pouzdan izvor (kao recimo, sigurna i validirana web stranica), te nastoji ukrasti usernames, šifre i ostale osjetljive informacije o korisniku kako bi ih kasnije koristio u vlastite svrhe. [↑](#footnote-ref-2)
3. Sinonim za maliciozni kod. [↑](#footnote-ref-3)
4. Način ubacivanja malicioznog koda. [↑](#footnote-ref-4)
5. Entitet predstavlja vrstu *storage-a* koji je definisan XML standardom. [↑](#footnote-ref-5)
6. Funkcija koja prima parametar na ulazu i vraća vrijednost a da pri tome ne modifikuje ni jednu varijablu koja se nalazi izvan njenog scope-a. [↑](#footnote-ref-6)
7. Za definisanje gornje kompleksnosti algoritama koristi se Big-O notacija. [↑](#footnote-ref-7)