

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2280

**SIGURNOST WEB APLIKACIJA REALIZIRANIH
PROGRAMSKIM RAZVOJNIM OKVIROM ANGULAR**

Mario Kovačević

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2280

**SIGURNOST WEB APLIKACIJA REALIZIRANIH
PROGRAMSKIM RAZVOJNIM OKVIROM ANGULAR**

Mario Kovačević

Zagreb, lipanj 2020.

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2280

Pristupnik: **Mario Kovačević (0036494769)**
Studij: Računarstvo
Profil: Programsko inženjerstvo i informacijski sustavi
Mentor: prof. dr. sc. Mirta Baranović

Zadatak: **Sigurnost web aplikacija realiziranih programskim razvojnim okvirom Angular**

Opis zadatka:

Analizirati i klasificirati sigurnosne prijetnje kojima su izloženi suvremeni softverski proizvodi. Analizirati razvojni okvir Angular te mogućnosti on koje pruža pri razvoju sigurnih aplikacija u web okruženju. Razviti web aplikaciju fokusiranu na sigurnost, što uključuje programiranje, sigurnosnu analizu te provođenje testova. Prikazati eksplicitne i implicitne metode za poboljšanje sigurnosti aplikacije. Analizirati uobičajene ranjivosti i napade, u skladu s provedenom analizom i klasifikacijom te prikazati načine kako ih spriječiti koristeći razvojni okvir Angular. Analizirati rezultate različitih testova koji će ukazati na poboljšanje sigurnosti izgrađene web aplikacije. Prikazati rezultate istraživanja s pomoću prikladnih alata za vizualizaciju. Definirati preporuke i pravila za izgradnju sigurnih aplikacija pri korištenju radnog okvira Angular.

Rok za predaju rada: 30. lipnja 2020.

Sadržaj

Uvod	1
1. Razvojni okvir Angular	2
1.1. Osnovne značajke	2
1.2. Izgrađena aplikacija	5
2. Ranjivosti i napadi	7
2.1. Umetanja.....	8
2.1.1. Umetanje naredbi operacijskog sustava	8
Primjer umetanja naredbi operacijskog sustava	9
2.1.2. Umetanje SQL koda	10
Primjer umetanja SQL koda	10
2.1.3. Zaštita od umetanja.....	12
2.2. Pokvarena autentikacija	15
2.2.1. Primjer pokvarene autentikacije	15
2.2.2. Zaštita od pokvarene autentikacije	16
2.3. Loša kontrola pristupa	17
2.3.1. Primjer loše kontrole pristupa.....	17
2.3.2. Zaštita od loše kontrole pristupa.....	19
2.4. Loše sigurnosne postavke	21
2.4.1. Primjer loših sigurnosnih postavki	21
2.4.2. Zaštita od loših sigurnosnih postavki	22
2.5. Međustranično skriptiranje (XSS)	24
2.5.1. Primjer međustraničnog skriptiranja	25
2.5.2. Zaštita od međustraničnog skriptiranja.....	26
Sanitizacija u Angularu	26
Vjerovanje sigurnim vrijednostima	27

2.6.	Međustranično lažiranje zahtjeva (CSRF).....	28
2.6.1.	Primjer međustraničnog lažiranja zahtjeva	28
2.6.2.	Zaštita od međustraničnog lažiranja zahtjeva.....	29
3.	Najbolja praksa	31
3.1.	Ahead-of-Time (AOT) prevoditelj	31
3.2.	Izbjegavanje Angularovog rizičnog aplikacijskog programskog sučelja	32
3.3.	Aplikacijski protokol HTTPS.....	33
3.4.	Sigurnosna zaglavlja.....	34
3.4.1.	Politika sigurnosti sadržaja.....	35
3.5.	Najbolja praksa programiranja u Angularu	36
3.6.	Ostale metode	38
4.	Testiranje sigurnosti	40
4.1.	Testiranje OWASP ZAP alatom.....	40
4.2.	Testiranje Checkbot: SEO, Web Speed & Security Checker skenerom.....	42
4.3.	Testiranje Mozilla Observatory skenerom	43
4.4.	Testiranje Security Headers skenerom	45
	Zaključak	46
	Literatura	47
	Sažetak.....	50
	Summary.....	51

Uvod

Kako web nastavlja rasti, tako rastu i zahtjevi na koje je potrebno odgovoriti za održavanje visoke razine povjerenja i sigurnosti web aplikacija. Web aplikacije su računalni programi koji se izvodi u internetskom pregledniku klijenta [1]. Često su zasnovani na distribuiranoj klijent/poslužitelj arhitekturi te se danas koriste pri brojnim internetskim uslugama, od bankovnih mrežnih sjedišta do društvenih mreža. Uzimajući u obzir njihovu poslovnu važnost, kao i potencijalne koristi za napadače i hakere, napadi na njih iznimno su česti. Najčešće su posljedica nefokusiranosti razvojnika na sigurnost što rezultira sigurnosnim manama u dizajnu i razvoju aplikacije.

Potreba za izgradnjom inteligentnijih i sigurnijih web aplikacija postaje sve očitija te njihova sigurnost postaje jednom od ključnih tema računalne sigurnosti. Također, kako svaka nova tehnologija postaje zastarjela za samo par mjeseci, kontinuirani fokus na sigurnosti od goleme je važnosti.

U današnje vrijeme klijentski dio web aplikacije često je izgrađen kao jednostranična aplikacija nasuprot tradicionalnim višestraničnim aplikacijama. Kod jednostraničnih web aplikacija dolazi do dinamičkog prepisivanja trenutne web stranice s novim podacima od poslužitelja, umjesto tradicionalne metode u kojoj preglednik učitava cijelu novu stranicu [2]. Osim toga, od ostalih prednosti ističu se bolje performance (iako inicijalno učitavanje traje duže), modularnost i jednostavnost. Jedan od razvojnih okvira za stvaranje klijentskog dijela aplikacije je *Angular*, okvir koji se prema istraživanju *Stack Overflowa* za 2020. godinu nalazi na 3. mjestu najkorištenijih web razvojnih okvira [3]. Upravo je istraživanje njegove sigurnosti pri razvoju web aplikacija glavna tema ovoga rada.

Cilj rada prikazati je sigurnosne prednosti i mane ovoga okvira služeći se brojnim poznatim ranjivostima i napadima. Upravo je zbog njih izgrađena primjerena ranjiva web aplikacija koja ne sadrži nikakvu dodanu zaštitu. Osim prednosti i mana, u radu bit će prikazane metode zaštite i najbolje prakse za sprječavanje navedenih ranjivosti. Nakon dodavanja tih metoda unutar ranjive aplikacije, rad se zaključuje paralelnim sigurnosnim testiranjem ranjive i sigurne verzije aplikacije koje će ocijeniti poboljšanje sigurnosti.

1. Razvojni okvir Angular

Angular (također poznat i kao *Angular 2+*) je programski razvojni okvir i platforma za kreiranje efikasnih i sofisticiranih jednostraničnih aplikacija. Vođen je od strane *Googlea* te *Angularove* zajednice. Originalno je objavljen kao *AngularJS* 2010. godine, ali je zbog pojave novih razvojnih okvira poput *Reacta* i *Vuea* koji su nudili veći raspon funkcionalnosti, 2016. došlo do potpune preinake *AngularJS*-a te je nastao *Angular*. Trenutna stabilna verzija je 9.1.9 objavljena 20. svibnja 2020. [4]

1.1. Osnovne značajke

Angular se pokreće na čistome *TypeScriptu* i *HTML*-u te ne zahtijeva nikakve dodatne pakete ovisnosti da bi funkcionirao. *TypeScript* je programski jezik otvorenog kôda koji predstavlja *JavaScript* nadskup što znači da su i postojeći *JavaScript* kôdovi ispravni *TypeScript* kôdovi. *TypeScript* razvija i održava *Microsoft* [5].

No, kako preglednici ne mogu izravno izvršavati *TypeScript* kôd, nužno je njegovo prevođenje u *JavaScript* pomoću *tsc* prevoditelja čija se konfiguracija nalazi unutar *Angularove tsconfig.json* datoteke [6].

Za razliku od *JavaScripta*, *TypeScript* je objektno-orijentirani programski jezik, dok je *JavaScript* skriptni jezik. Druga značajna prednost je mogućnost strogog tipiziranja, odnosno deklariranja tipova podataka, svojstva koje *JavaScript* ne podržava.

Angular aplikacije koriste komponentnu arhitekturu, što znači da se sastoje od brojnih komponenti koje se grupiraju u modul. Modul je razred koji ujedinjuje više funkcionalnosti u jednu cjelinu. Aplikacija minimalno mora sadržavati korijenski modul, po konvenciji nazvan *AppModule*, koji omogućuje početno pokretanje (eng. *bootstrapping*) aplikacije te sadržava korijensku komponentu *AppComponent*. Ostale komponente ugnježđuju se unutar korijenske [7].

Neke od osnovnih funkcionalnosti koje *Angular* pruža su:

- Komponente – Svaka komponenta sadrži *TypeScript* razred koji sadrži aplikacijske podatke i logiku te je povezan s predloškom (sadrži *HTML* kôd i specifične *Angularove* oznake) koji definira prikaz kakav će biti vidljiv korisniku. Osim toga, taj razred sadrži i dekorater *@Component()* koji identificira da se radi o komponenti [7].

Određivanje komponente razredom ostvaruje se moguća ponovna uporaba komponenti u raznim dijelovima aplikacije.

- Direktive – Direktive predstavljaju *Angularov* koncept za ostvarenje promjene izgleda, strukture ili ponašanja dokumentno objektnog modela (*DOM*)¹. Očigledno je da su onda i komponente direktive, no razlikuju se po tome što komponente sadrže predložak koji određuje kakav će prikaz biti vidljiv. Uz to, direktive sadrže *@Directive()* dekorater, ali imaju i selektor (kao i komponente). Za njihov rad, potrebno ih je dodati uz željenu *HTML* oznaku kojom se želi upravljati. Osim već pojašnjenih komponentnih direktiva, *Angular* definira još dvije vrste:

1. Strukturalne – mijenjaju strukturu *DOM*-a, dodajući ili uklanjajući elemente. Ugrađene strukturalne direktive su: *NgFor*, *NgSwitch* i *NgIf* [8].
2. Atributne – upravljaju atributima *HTML* elemenata, čime se mijenja izgled i ponašanje *DOM*-a. Postoje tri predefinirane direktive: *NgStyle*, *NgClass* te *NgNonBindable* [9].

- Povezivanje podataka – koncept koji omogućuje komunikaciju komponente i predloška. Postoje četiri vrste povezivanja podataka koje se razlikuju u smjeru prijenosa podataka:
 1. Interpolacija – unutar predloška pridružuje vrijednost svojstva iz komponente koristeći sintaksu: *{{value}}*, gdje *value* predstavlja vrijednost koja će se pridružiti.

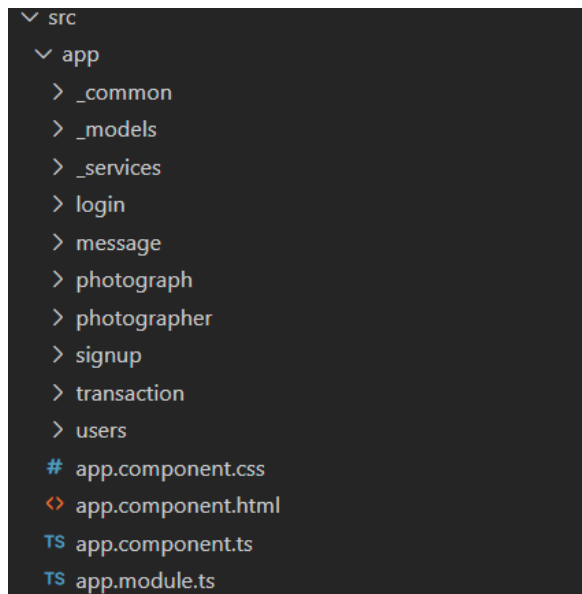
¹ Programsko sučelje za *HTML* i *XML* dokumente.

2. Povezivanje svojstva – pridružuje vrijednost iz komponente do specifičnog svojstva u predlošku (koje je često *HTML* atribut). Koristi sljedeću sintaksu: `[property]="value"`, pri čemu je *value* vrijednost koja se pridružuje svojstvu *property*.
 3. Povezivanje događaja – povezuje događaj te odgovarajuću funkciju koja će se izvesti pri njegovom ostvarenju. Podrazumijevana sintaksa je: `(event)="function"`, gdje *event* predstavlja događaj, a *function* funkciju koja će se pozvati prilikom ostvarenja događaja.
 4. Dvosmjerno povezivanje podataka – omogućuje prijenos podataka na oba načina, od komponente do predloška te od predloška do komponente. Zato se koristi direktiva *ngModel* uz pretpostavljenu sintaksu: `[(ngModel)]="value"`, gdje *value* predstavlja vrijednost koja se prenosi [10].
- Usluge – Usluge su *Angularov* pojam koji se koristi za bilo koji podatak ili aplikacijsku logiku koja nije vezana za specifični prikaz te se želi dijeliti među komponentama. Bazirane su na principu ubrizgavanja ovisnosti, programskog obrasca u kojem objekt prima druge objekte te koristi njihove funkcionalnosti umjesto da ih sam stvara. Uslugu većinom čini razred s dobro definiranom funkcionalnošću te dekoraterom `@Injectable()` [11].
 - Usmeravanje – *Angular* za usmjeravanje koristi *Router* koji omogućuje definiranje navigacijske staze za različita stanja aplikacije. Koristi standardnu konvenciju navigiranja unutar preglednika s osnovnom razlikom da *Router* povezuje uneseni *URL* s odgovarajućim prikazima umjesto sa stranicama [12]. Tako se u datoteci *routes.config.ts* svaka moguća ruta unutar *Angular* aplikacije povezuje s odgovarajućim prikazom koji će biti izložen kada korisnik pristupi toj ruti.
 - Obrasci – Svrha obrazaca je prikupljanje podataka od korisnika te *Angular* pri radu s njima koristi koncept povezivanja podataka, tako da se podaci uneseni u obrazac (koji se nalazi unutar predloška) mogu povezati sa svojstvima komponente odakle se mogu dalje obrađivati. Unutar *Angulara* postoje dvije vrste obrazaca:

1. Obrasci temeljeni na predlošku (eng. *template-driven forms*) – Obrasci koji se grade unutar predloška odgovarajuće komponente. Lako se stvaraju, ali nisu skalabilni te se zbog toga koriste za kreiranje jednostavnih obrazaca. Za njihovo korištenje potrebno je uvesti *FormsModule* modul unutar glavnog modula aplikacije.
 2. Reaktivni obrasci (eng. *reactive forms*) – Obrasci koji se grade unutar razreda komponente. Skalabilni su te se preporučuju za svakodnevno korištenje u aplikaciji. Za njihovo korištenje potrebno je uvesti *ReactiveFormsModule* u glavni modul aplikacije [13].
- Komunikacija s poslužiteljem – Za komunikaciju preko *HTTP* protokola s poslužiteljem, na kojemu je implementirano *REST* aplikacijsko programsko sučelje, *Angular* koristi biblioteku *HttpClient* koja sadrži sve potrebne *HTTP* metode (*get*, *post*, *put*, *delete*, *patch*) [7].
- Osim toga, pruža podršku za presretanje zahtjeva i odgovora te za upravljanje greškama.

1.2. Izgrađena aplikacija

Kao praktični dio ovoga rada, izgrađena je *PhotographiusScope* web aplikacija koja se sastoji od klijentskog i poslužiteljskog dijela na kojemu se nalazi baza podataka. Klijentski je dio kreiran pomoću *Angulara* (verzija 9.1.0) i to koristeći *Angularovo* naredbeno linijsko sučelje, dok je za poslužiteljski dio iskorišten *Express* (*Node.js* razvojni okvir za web aplikacije). Baza podataka kreirana je pomoću *SQLite* biblioteke. Struktura klijentskog dijela aplikacije vidljiva je na slici (Sl. 1.1) gdje se primjećuje implementirana komponentna arhitektura.



Sl. 1.1 Struktura klijentskog dijela *PhotographiusScope* aplikacije

PhotographiusScope je web aplikacija kojoj je osnovni cilj omogućiti svojim korisnicima kupovanje te prodaju kupljenih fotografija. Za obavljanje tih akcija potrebna je prethodna prijava ili registracija pri kojoj korisnici unose osnovne informacije. Uz to, korisnici mogu obavljati transakcije te slati poruke. Aplikacija sadrži jednoga administratora kojemu je primarni cilj upravljati podacima o fotografijama te o fotografima koji su ih fotografirali. To uključuje njihovo dodavanje, izmjenu te moguće brisanje. Osim toga, administrator može pregledati sve obavljene transakcije kao i informacije o svim korisnicima. Pregledi pojedinih spomenutih značajki bit će predstavljeni unutar primjera napada i ranjivosti u okviru idućeg poglavlja.

Aplikacija ne sadrži nikakvu zaštitu (osim već ugrađenih) te je izgrađena na što nesigurniji način s ciljem prikaza i iskorištenja njenih sigurnosnih mana da bi se ona na kraju s brojnim metodama zaštite i primjene najbolje prakse pretvorila u sigurnu i funkcionalnu. Tako će na kraju postojati dvije verzije iste aplikacije: ranjiva i sigurna.

2. Ranjivosti i napadi

Općeniti problem sa sigurnošću web aplikacija veže se uz ranjivosti koje predstavljaju moguće slabosti unutar softvera koje napadač može iskoristiti. Ranjivosti mogu biti posljedice raznih problema, od grešaka izvornog kôda web aplikacije do kriptografskih i mrežnih mana.

Tako je u nastavku rada prikazan skup ranjivosti i napada koji iskorištavaju te ranjivosti. Sve ranjivosti implementirane su u izgrađenoj *PhotographiusScope* aplikaciji, kao i izvedeni napadi. Za svaku spomenutu ranjivost dane su i metode zaštite koje su zatim i primijenjene.

Za odabir ranjivosti, koristila se *OWASP*-ova lista top 10 sigurnosnih rizika za web aplikacije iz 2017. godine što je trenutno najnovija verzija, kao i metode zaštite koje *OWASP* preporučuje za njihovo preveniranje [14]. *OWASP* je osnovan u prosincu 2001. kao međunarodna neprofitna organizacija s ciljem poboljšanja web sigurnosti. Gotovo kroz cijelo njihovo postojanje, pratili su možda svaki moguć tip napada na web sigurnost te nudili rješenja i upute za njihovo preveniranje. Pri odabiru ranjivosti s *OWASP*-ove liste, u obzir su se uzete ranjivosti protiv kojih *Angular* nudi ili sadrži ugrađene zaštite kao i ranjivosti koje nisu usko vezane za razvojni okvir, ali su česte u web aplikacijama poput problema s autentikacijom ili autorizacijom.

2.1. Umetanja

Napadi umetanjem vrsta su napada u kojima napadač opskrbljuje aplikaciju unosom kojemu se ne može vjerovati. Umetanja spadaju među najstarije i najopasnije napade kojima su podložne web aplikacije te se zbog toga nalaze na prvome mjestu *OWASP*-ovih sigurnosnih prijetnji za web aplikacije [15].

Mogu dovesti do krađe ili gubitka podataka, odbijanja usluge kao i kompromitacije cijeloga sustava.

Primarni razlog koji omogućuje postojanje ranjivosti koja dovodi do ovih napada je nedovoljna ili nepostojeća validacija korisničkog unosa.

Neke od poznatih vrsta umetanja uključuju: umetanja kôda, *CRLF* umetanja, umetanja naredbi operacijskog sustava, umetanja zaglavlja maila te umetanja *SQL* kôda. U nastavku ovoga rada naglasak je postavljen na dvije vrste ovih napada: umetanja naredbi operacijskog sustava te umetanja *SQL* kôda.

2.1.1. Umetanje naredbi operacijskog sustava

Web aplikacije ponekad moraju izvršavati naredbe operacijskog sustava radi ispunjenja određene funkcionalnosti koju nude ili zbog komunikacije s datotečnim sustavom na operacijskom sustavu poslužitelja. Iako ova mogućnost može biti iznimno korisna, također može biti i veoma opasna kada se ne koristi propisno. Upravo je to osnova za napad umetanjem naredbi operacijskog sustava.

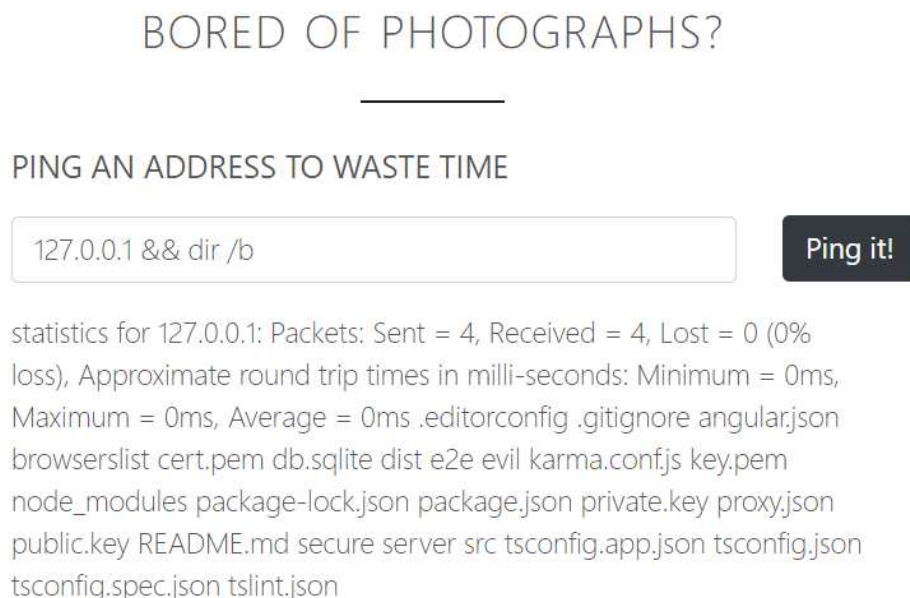
Cilj tog napada je izvršavanje proizvoljnih naredbi operacijskog sustava na operacijskom sustavu na kojemu se pokreće ranjiva aplikacija. Njihovo izvršavanje moguće je kada aplikacija prosljeđuje nesiguran korisnički unos u ljusku operacijskog sustava koji izvršava odgovarajuće naredbe s privilegijama poslužitelja web aplikacije [16]. Napadi umetanjem naredbi mogu rezultirati gubitkom ili modifikacijom podataka te onemogućavanjem usluge.

No prije izvođenja konkretnih napada napadač mora saznati o kojemu se operacijskom sustavu radi kako bi znao prilagoditi opasan unos za odgovarajući operacijski sustav. Pri

tome mu mogu pomoći naredbe *uname* (*Linux* i *Mac OS*) te *ver* (*Windows*) koje daju informaciju o kojemu se operacijskom sustavu radi.

Primjer umetanja naredbi operacijskog sustava

Kao primjer napada razmotrit će se funkcionalnost unutar *PhotographiusScope* aplikacije koja omogućuje prozivanje željene *IPv4* adrese. Ranjiva aplikacija ne provjerava korisnički unos, tako da se proizvoljan korisnički unos prosljeđuje do poslužitelja web aplikacije koji unutar ljuske svog operacijskog sustava izvršava naredbu *ping ipAddress*, pri čemu parametar *ipAddress* predstavlja korisnički unos željene *IP* adrese. Aplikacija očekuje ispravan unos *IPv4* adrese, no kako to ne provjerava, napadač može unijeti i zlonamjerni unos poput: „*127.0.0.1 && dir /b*”. Osim što će se prozvati željena adresa i dati ispis dobivenog rezultata, također će se ispisati i sadržaj svih datoteka unutar direktorija u kojemu se poslužitelj web aplikacije nalazi, što može značajno pomoći napadaču u otkrivanju novih ranjivosti i pripremi za iduće napade. Prikaz opisanog napada nalazi se na slici (Sl 2.1).



Sl 2.1 Primjer uspješnog napada umetanjem naredbi operacijskog sustava

Pri ovome napadu koristi se „&&” koji predstavlja metaznak ljeske što omogućuje da se naredbe ulančavaju. Osim „&&” još se mogu koristiti: „&”, „|”, „||” (*Windows* i *Linux*) te „;”, „0x0” ili „\n” (samo za *Linux*).

Kako bi se preveniralo umetanje naredbi operacijskog sustava, preporučuje se prije svega izbjegavanje razvoja funkcionalnosti web aplikacije koje kao posljedicu imaju izvršavanje naredbi operacijskog sustava. No, ako su te funkcionalnosti nužne, mora se osigurati valjana validacija unosa koji se izvršava u ljusci operacijskog sustava.

2.1.2. Umetanje SQL koda

Napad umetanjem *SQL* kôda sastoji se od umetanja *SQL* upita preko korisničkih unosa. Unos se dalje prosljeđuje do poslužitelja koji izvršava *SQL* upite (nad bazom podataka) koji sadrže nesiguran korisnički unos. Uspješnim izvođenjem ovoga napada mogu se pročitati ili modificirati osjetljivi podaci iz baze podataka te izvršavati administratorske operacije nad bazom [17].

Napad se događa kada se korisnički unos ne provjerava te kada se *SQL* upit generira dinamički.

Prve javne rasprave o umetanju *SQL* kôda počele su još 1998. godine kada ih je Jeff Forristal otkrio i objavio u *Phrack 54* magazinu [18]. Iako se radi o dobro poznatoj ranjivosti, nedavni primjer iz 2018. godine kada je kompanija *Valve* platila 25 tisuća dolara korisniku koji je otkrio i prijavio moguće umetanje *SQL* kôda na njihovoj *Steam* platformi, pokazuje da su ti napadi i dalje mogući [19].

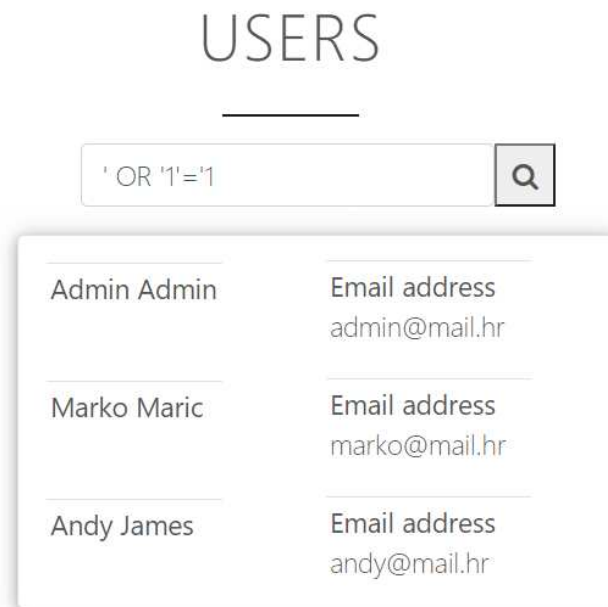
Primjer umetanja SQL koda

Kao primjer uspješnog izvođenja ovoga napada, u *PhotographiusScope* aplikaciji postavljena je tražilica koja omogućuje pretraživanje korisnika po imenu. Kako se korisnički unos ne provjerava, on se prosljeđuje do poslužitelja koji dinamički stvara *SQL* upit: „var sql = \"SELECT * FROM user WHERE firstName = ' \" + input + \" ' \" ;\". Ovakvo stvaranje *SQL* upita stvara ranjivost koju napadač lako može iskoristiti. Unosom „' OR 'I'='I'” stvorit će se *SQL* upit: „SELECT * FROM user WHERE firstName = ' ' OR 'I'='I' ”. Pri izvođenju tog upita, baza podataka više ne uspoređuje podatke u

tablici s imenom kojeg je korisnik unio, nego provjerava istinitost tvrdnje `'I'='I'`. Kako je ta tvrdnja uvijek istinita, cijeli *WHERE* dio *SQL* upita bit će istinit, što će rezultirati vraćanjem svih redova u tablici *user*.

Općeniti cilj umetanja *SQL* kôda stvoriti je *SQL* upit koji uvijek vraća istinu u *WHERE* dijelu upita, ali pritom treba paziti kakvu sintaksu koristi *SQL* upit. Tako napadač može pogoditi sintaksu metodom pokušaja i pogrešaka ili ju saznati preko prikupljenih informacija o strukturi i vrsti baze podataka [20].

Prikaz opisanog izvedenog napada nalazi se na slici (Sl. 2.2). Izvođenjem napada napadač je došao do elektroničke adrese administratora, koja prije toga nije bila nigdje vidljiva.



Sl. 2.2 Primjer uspješnog napada umetanjem *SQL* kôda

Osim nužne validacije korisničkog unosa, da bi se izbjegli napadi umetanjem *SQL* kôda, nikako se ne smije koristiti dinamičko kreiranje *SQL* upita. Siguran način uključuje korištenje parametriziranih izjava (eng. *parameterized statments*) koje imaju oblik predloška unutar kojega se vrijednosti zamjenjuju tijekom svakog izvođenja. Unutar *PhotographiusScope* aplikacije sva dinamička generiranja upita zamijenjena su sigurnim parametriziranim načinom. Tako da sada *SQL* upit za dohvat traženog korisnika izgleda ovako: „*var sql = 'SELECT * FROM user WHERE firstName = (?)'*“. Tada će se na korisnički unos „*' OR 'I'='I'*“ stvoriti *SQL* upit oblika: „*SELECT * FROM user WHERE firstName = ' OR 'I'='I'*“, što znači da će se tražiti korisnik kojemu je ime jednako „*' OR*

'1'='1". Očigledno je da ovakvo generiranje *SQL* upita neće uzrokovati ranjivosti kao što je prouzrokovalo dinamičko generiranje.

2.1.3. Zaštita od umetanja

Validacija korisničkog unosa primarna je metoda zaštite protiv svih oblika umetanja. *Angular* omogućuje validaciju korisničkog unosa unutar obje vrste obrazaca: reaktivnih i vođenih predloškom.

Za dodavanje validacije u obrasce vođene predloškom potrebno je dodati validacijske attribute koji se inače koriste za validaciju *HTML* obrasca, poput: *maxlength*, *minlength*, *min*, *max*, *pattern* i *required*. Svaki puta kada se vrijednost kontrole obrasca² (eng. *form control*) promijeni, *Angular* izvršava validaciju i generira listu validacijskih grešaka koje mogu rezultirati nevažećim statusom ili s *null* koji odgovara važećem statusu [21].

Unutar *PhotographiusScope* aplikacije dodana je validacija korisničkog unosa *IP* adrese za prozivanje i to unutar obrasca vođenog predloškom. U *HTML* kôd za unos *IP* adrese dodaju se već spomenuti *HTML* validacijski atributi: *required*, *minlength*="7", *maxlength*="15" te *pattern*="^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\$". Oni provjeravaju odgovara li unos ispravnom obliku *IPv4* adrese te ako ne odgovara, onemogućuju pozivanje usluge za prozivanje dane *IP* adrese. Slika (Sl. 2.3) prikazuje izgled opisane validacije na djelu.



Sl. 2.3 Primjer ostvarene validacije u *PhotographiusScope* aplikaciji

Što se tiče validacije unutar reaktivnih obrazaca, razlika je što se validacije ne postiže pomoću validacijskih atributa u *HTML* kôdu, već pomoću validacijskih funkcija koje se

² *Angular*ov razred koji omogućuje pristup stanju elementa obrasca.

dodaju direktno u model kontrole obrasca u klasi komponente. *Angular* tada poziva te funkcije kada se vrijednost kontrole obrasca promijeni. Validacijske funkcije mogu biti sinkrone ili asinkrone. Zbog performance, *Angular* pokreće samo asinkrone validacijske funkcije jedino ako sve sinkrone funkcije daju važeći status. Kao validacijske funkcije mogu se koristiti *Angularove* ugrađene (koje odgovaraju *HTML* validacijskim atributima unutar obrazaca vođenih predloškom) ili se mogu izraditi vlastite validacijske funkcije [21].

Za validaciju korisničkog unosa pri spomenutom traženju korisnika, implementirala se validacija unutar reaktivnih obrazaca. No kako *Angularove* ugrađene validacijske funkcije nisu bile dovoljne za provjeravanje valjanosti korisničkog unosa, implementirala se i dodatna validacijska funkcija (Kôd 2.1). Ta funkcija implementira *Angularovo ValidatorFn* sučelje te vraća *null* ako je kontrolna vrijednost važeća ili validacijske greške ako nije. Prije toga provjerava se sadrži li upit znamenke, znakove „;”, „--”, „=” ili riječi poput „insert” i „drop table”. Takvi se unosi odbijaju jer nisu smisleni za traženje korisnika po imenu te jer je velika mogućnost da se s tim unosom pokušava izvesti *SQL* umetanje.

```
export function validSearch():ValidatorFn {
  return (control:AbstractControl):{[key:string]:any}|null => {
    if (/* unos sadrži nedozvoljene znakove */)
      return {validSearch:true};
    else
      return null;
  }
}
```

Kôd 2.1 – Primjer kôda proizvoljno izrađene validacijske funkcije

Vlastito izgrađena validacijska funkcija mora se dodati izravno u kontrolu obrasca kojemu pripada zajedno s ostalim validacijskim funkcijama kao što je prikazano u kôdu (Kôd 2.2), gdje se za provjeru korisničkog unosa, osim već spomenute vlastito izrađene validacije funkcije, nalaze i ugrađene *Angularove* validacijske funkcije *required* i *maxlength(25)*.

```

this.form = this.fb.group({
  search: new FormControl('', Validators.compose([
    Validators.required,
    validSearch(),
    Validators.maxLength(25)
  ])),
},);

```

Kôd 2.2 – Primjer kôda za dodavanje vlastito izrađene validacijske funkcije u kontrolu obrasca

Osim implementiranja validacije na klijentskoj strani, validaciju korisničkog unosa također je potrebno implementirati i na poslužiteljskoj strani koristeći iste provjere koje su se koristile i na klijentskoj.

2.2. Pokvarena autentikacija

Pokvarena autentikacija ranjivost je koja je uzrokovana slabo implementiranim mehanizmom upravljanja sjednicom i slabom autentikacijom. Autentikacija je pokvarena kada napadači uspiju kompromitirati lozinke, identifikatore sjednice ili informacije o korisničkim računima. Najčešći cilj napadača je oteti jedan ili više računa te tako preuzeti korisničke privilegije žrtve. Zbog velike opasnosti i raširenosti, ova se ranjivost nalazi na drugome mjestu unutar *OWASP*-ovih top 10 sigurnosnih prijetnji [15].

2.2.1. Primjer pokvarene autentikacije

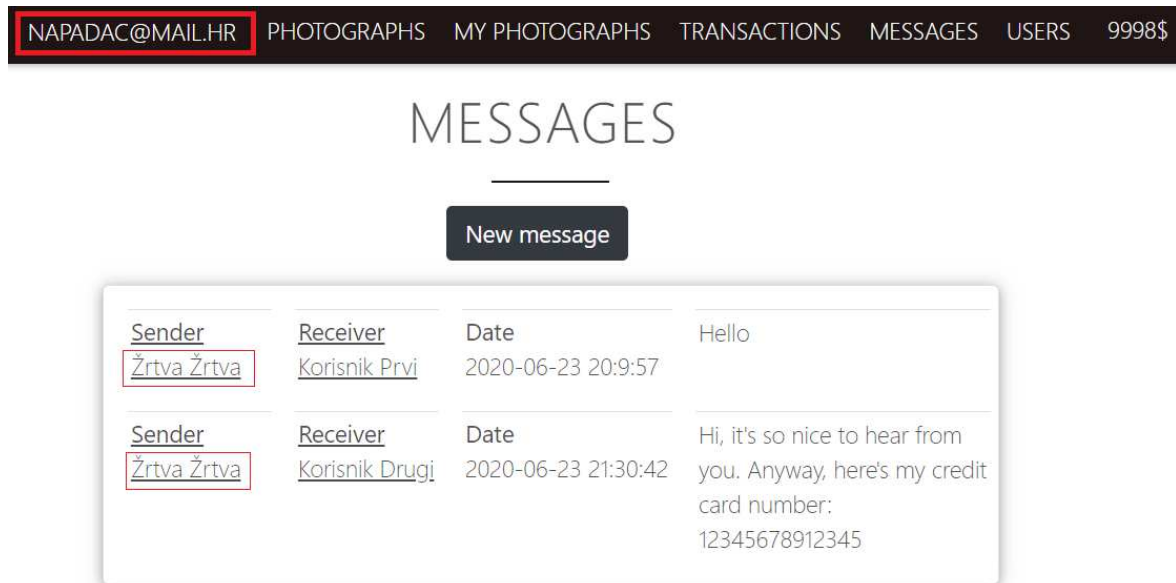
Trenutna implementacija *PhotographiusScope* aplikacije sadrži pokvarenu autentikaciju. Aplikacija ne sadrži nikakvu validaciju prilikom kreiranja računa tako da je moguće korištenje veoma slabih i nesigurnih lozinki (na primjer: „123”, „password”, „qwerty”...), lozinke su pohranjene u bazu podataka kao čisti tekst, identifikator sjednice nalazi se unutar *URL*-a pojedinih ruta, sjednica ne ističe te ne postaje nevažeća prilikom odjave korisnika. Kada aplikacija sadrži ovako loše implementiranu autentikaciju, samo je pitanje vremena kada će se njene ranjivosti iskoristiti. Najpoznatija tri napada koji iskorištavaju ove ranjivosti su *credential stuffing*, napad sirovom snagom (eng. *brute force attack*) te krađa sjednice.

Credential stuffing je napad pri kojemu se koriste alati koji omogućuju automatsko testiranje važećih korisničkih imena i lozinki koje su ukradene od jedne tvrtke, a primjenjuju se na web aplikaciji druge. Kako korisnici često koriste iste lozinke na višestrukim računima, napadači s ovom metodom mogu postići značajan uspjeh [22].

Napad sirovom snagom uključuje proces pokušavanja svake moguće lozinke sve dok se ne pronađe ispravna. U praksi se to često ne koristi tako, nego se pokušava s listom najčešće korištenih lozinki. Kako korisnici često imaju potrebu birati jednostavne lozinke te ih koristiti na više računa, ovaj napad postaje vrlo jednostavan, a efikasan [23].

Krađa sjednice napad je pri kojemu jedan korisnik koristi sjednicu drugoga. Primjer uspješnog izvođenja krađe sjednice moguć je u *PhotographiusScope* aplikaciji. Kako je identifikator sjednice dostupan u *URL*-u, korisnik bi eventualnim dijeljenjem njemu bezazlene poveznice ugrozio svoj račun i informacije jer bi tada napadač jednostavno

mogao uzeti identifikator sjednice iz te poveznice i iskoristiti za nešto malicioznije, poput pregledavanja poruka tog korisnika što je i prikazano na slici (Sl. 2.4).



Sl. 2.4 Primjer uspješne krađe sjednice

2.2.2. Zaštita od pokvarene autentikacije

Nakon dodanih mjera zaštite, *PhotographiusScope* aplikacija postaje primjer aplikacije s dobro definiranom autentikacijom. Lozinka se više ne pohranjuje kao čisti tekst, već se koristi *Argon2* algoritam za stvaranje sažetka lozinke te se dobiveni sažetak sprema u bazu. Onemogućeno je korištenje slabih i dobro poznatih lozinki, odnosno definirana je politika lozinki koja zahtijeva da lozinka sadrži minimalno 8 znakova, od kojih je minimalno jedno veliko i malo slovo te znamenka. Uz to, provjera se odgovara li lozinka nekoj lozinki iz *OWASP*-ovih top 1000 loših lozinki. Opisana politika lozinki implementirana je na klijentskoj strani uz pomoć već opisane *Angularove* validacije, dok je ista provjera dodana i na poslužiteljsku stranu. Opisanim mehanizmima zaštite značajno se smanjuju uspjesi za napad sirovom snagom.

Kao protumjera za automatizirane napade kao što su *credential stuffing* i napad sirovom snagom, dodana je višestruka autentikacija pri prijavljivanju u obliku *Googleovog reCAPTCHA* sustava koji sprječava automatizirano prijavljivanje u sustav. Što se tiče sjednice, identifikator sjednice više se ne pohranjuje u *URL-u*, već se nalazi u kolačiću svakog *HTTP* zahtjeva i odgovora koji se s atributom *maxAge* limitira na željeno

vrijeme trajanja. Osim toga, sjednica se uspješno briše nakon odjave korisnika. Navedene zaštite efikasno smanjuju vjerojatnost uspješnosti krađe sjednice.

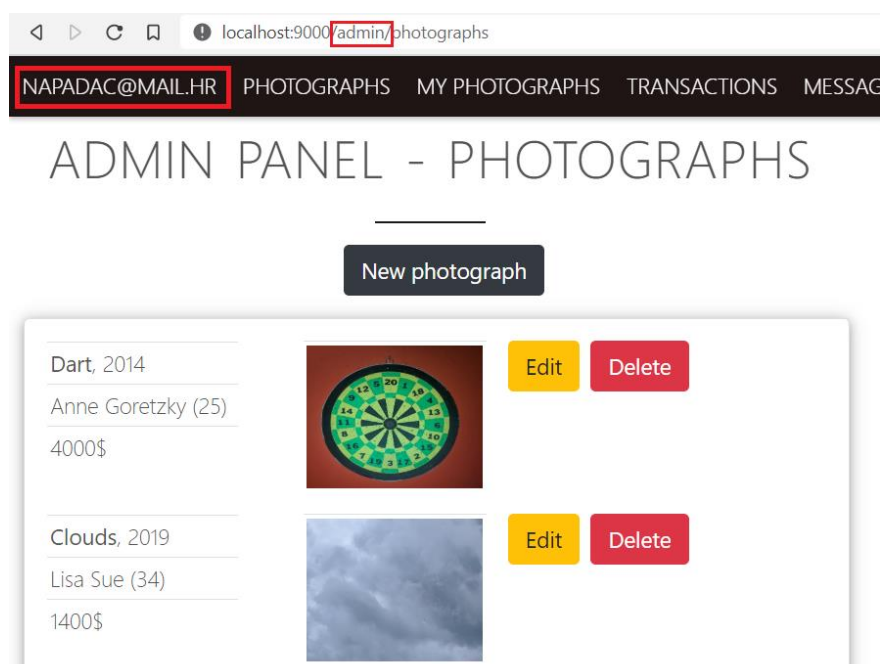
2.3. Loša kontrola pristupa

Loša kontrola pristupa uključuje ranjivosti koje se događaju zbog loše implementirane autorizacije, mehanizma pomoću kojega web aplikacija dopušta pristup sadržaju i funkcijama za određene korisnike. Ove provjere izvode se nakon autentikacije i reguliraju dozvole autoriziranim korisnicima. Iako problem kontrole pristupa zvuči jednostavno, radi se o kompleksnom problem koji je usko vezan za sadržaj i funkcije koje web aplikacija pruža. Stoga, ako se autorizacija loše implementira, napadač bi osim uvida u neautoriziran sadržaj mogao mijenjati ili brisati ga, izvoditi neautorizirane akcije ili čak i preuzeti administraciju web aplikacije koja je često primarna meta za napade. Ova ranjivost nalazi se na 5. mjestu spomenute *OWASP*-ove liste [24].

Jedan od načina na koje napadač može otkriti lošu kontrolu pristupa je takav da prvo posjeti sve moguće rute aplikacije dok je prijavljen, a zatim pokuša isto dok je odjavljen. Osim toga, moguće je i isprobavanje svih mogućih kombinacija puteva unutar aplikacije. Primjerice, `/admin` ili `/settings` primjeri su puteva kojima bi jedino administrator trebao moći pristupiti. Ako bilo koji drugi korisnik tome pristupi, to se može smatrati ranjivošću. Iz perspektive korisnika, kontrola pristupa može se podijeliti na vertikalnu, horizontalnu i kontekstno zavisnu. U okvirima ovoga rada razmotrit će se prva dva tipa.

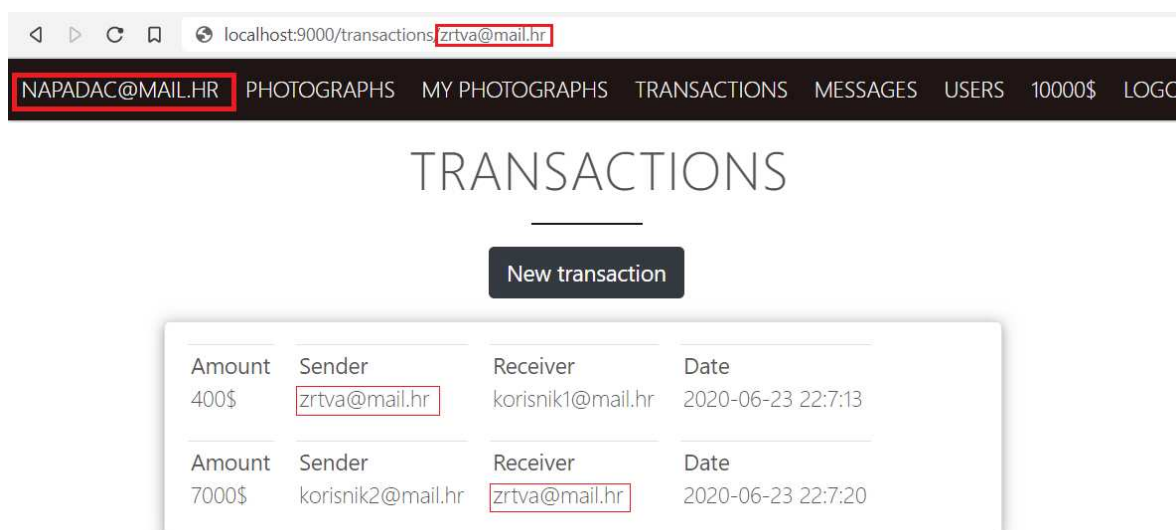
2.3.1. Primjer loše kontrole pristupa

Vertikalna kontrola pristupa mehanizam je koji sprječava pristup funkcionalnosti za neautorizirane korisnike. Primjer loše implementirane vertikalne kontrole pristupa vidljiv je u *PhotographiusScope* aplikaciji. *URL* za fotografije kojima pristupaju korisnici je `users/photographs`, što napadač može promijeniti u `admin/photographs` te pristupiti administratorskom sučelju za fotografije na kojima može izvršavati funkcije poput dodavanja, brisanja ili uređivanja fotografija iako nema pravo na to. Slika (Sl. 2.5) prikazuje izgled aplikacije s opisanom ranjivosti.



Sl. 2.5 Primjer iskorištavanja vertikalne kontrole pristupa

U horizontalnoj kontroli pristupa sprječava se pristup sadržaju za neautorizirane korisnike. Primjer iskorištavanja dostupan je u spomenutoj aplikaciji gdje je *URL* za pregledavanje vlastitih transakcija: *mytransactions/moj@mail.hr*. Napadač može modificirati *URL* u *mytransactions/zrtvin@mail.hr* te će pristupiti transakcijama žrtve. Prikaz iskorištavanja ove ranjivosti dan je na slici (Sl. 2.6).



Sl. 2.6 Primjer iskorištavanja horizontalne kontrole pristupa

2.3.2. Zaštita od loše kontrole pristupa

Za ostvarenje snažne kontrole pristupa i zaštite protiv navedenih ranjivosti, u *PhotographiusScope* aplikaciji razvila se kontrola pristupa bazirana na ulogama (eng. *Role-based access control*). Tako autorizirani korisnici mogu imati dvije vrste uloga: „ADMIN“ i „USER“. Informacije o ulogama prijavljenog korisnika nalaze se unutar *JSON Web Token (JWT)*³ tokena (ujedno predstavlja i identifikator sjednice) koji se postavlja u kolačić prilikom korisničke prijave ili registracije. Prije izvođenja svake akcije, provjerava ima li korisnik odgovarajuću ulogu za izvođenje te akcije. Ako korisnik nema propisanu ulogu, zahtjev se odbija te poslužitelj vraća status *403 Forbidden*. Navedenim mehanizmom osigurala se autorizacija poslužiteljske strane.

Što se tiče klijentske strane, za skrivanje sadržaja i funkcija od neautoriziranih korisnika može se implementirati vlastita *Angularova* strukturalna direktiva koja implementira *OnDestroy* sučelje. Direktiva će prikazati ili sakriti određeni *HTML* element ovisno o tome sadrži li korisnik odgovarajuću ulogu. Korištenje opisane direktive dano je u kôdu (Kôd 2.3) gdje će dani *HTML* element biti vidljiv jedino korisniku koji ima ulogu administratora.

```
<li *rbacAllow="[ 'ADMIN' ]">
    <a routerLink="/photographers">Photographers</a>
</li>
```

Kôd 2.3 – Primjer kôda za korištenje vlastito izgrađene direktive za provjeru korisničke uloge

Osim navedene direktive, *Angular* nudi još jedan mehanizam koji omogućuje zaštitu ruta unutar aplikacije. Radi se o zaštitarima ruta (eng. *route guards*) koji mogu omogućiti ili zabraniti pristup određenim dijelovima navigacije. Postoje 4 vrste zaštitara ruta:

- *CanActivate*: kontrolira hoće li se ruta aktivirati.
- *CanActivateChild*: kontrolira hoće li se podruta aktivirati.
- *CanLoad*: kontrolira hoće li se ruta uopće učitati.
- *CanDeactivate*: kontrolira može li korisnik napustiti rutu [25].

³ Internetski standard za kreiranje podataka s opcionalnim potpisom i enkripcijom.

Unutar *PhotographiusScope* aplikacije dodan je zaštitar rute koji implementira *CanActivate* sučelje te sadrži *canActivate* funkciju koja provjerava ima li korisnik odgovarajuću ulogu da bi pristupio određenoj ruti. Ako nema, korisnik se preusmjerava na pretpostavljenu rutu. *CanActivate* funkcija ima i pristup parametrima *ActivatedRouteSnapshot* te *RouterStateSnapshot* u slučaju da su potrebne informacije o trenutnoj ruti. Kôd opisanog zaštitara rute nalazi se u kôdu (Kôd 2.4). Implementirani zaštitar rute mora se postaviti unutar željene rute u datoteci *routes.config.ts*.

```
export class AuthorizationGuard implements CanActivate {
  constructor (private allowedRole:string, private
    authService:AuthService, private router:Router) {}

  canActivate (route:ActivatedRouteSnapshot, state
    RouterStateSnapshot): Observable<boolean> {
    return this.authService.user$.pipe(
      map(user => this.allowedRole == user.role),
      first(), (allowed => {
        if (!allowed)
          this.router.navigateByUrl('/');
      }),
    );
  }
}
```

Kôd 2.4 – Programski kôd implementiranog zaštitara rute

2.4. Loše sigurnosne postavke

Loše sigurnosne postavke uključuju ranjivosti koje može izazvati vlastito napisan kôd aplikacije, kôd unaprijed izrađenih funkcionalnosti ili aplikacijsko programsko sučelje. Predstavljaju iznimno široku kategoriju ranjivosti te su zbog toga veoma česte. Mogu se pojaviti u samoj aplikaciji, na poslužitelju, bazi podataka ili u resursima koji se koriste tijekom razvojnog procesa. Kako aplikacija raste, postaje sve teže držati sigurnosne konfiguracije sigurnim i efikasnim, pogotovo ako aplikacija sadrži nepotrebne ili nekorištene funkcionalnosti. Potencijalni utjecaj i iskoristivost ovih ranjivosti ovisi o vrsti pogrešne konfiguracije te u najgorem slučaju može dovesti do potpunog preuzimanja sustava [15].

Zbog navedenih razloga, ova je ranjivost na 6. mjestu *OWASP*-ove liste top 10 sigurnosnih prijetnji u web aplikacijama.

U prosincu 2019. godine *Microsoft* je postao žrtva loših sigurnosnih postavki kada je došlo do izlaganja 250 milijuna slučajeva korisničke podrške iz *Azure* baze podataka. Podacima se moglo pristupiti bez bilo kakve autentikacije [26].

2.4.1. Primjer loših sigurnosnih postavki

PhotographiusScope sadrži neke od primjera loših sigurnosnih postavki. Pri ispisu greške ispisuje se cijeli trag stoga (eng. *stack trace*) što može napadaču dati uvid u moguće postojeće mane te mu tako olakšati pripremu za idući napad. Primjer ispisa greške pri prijavi korisnika dan je na slici (Sl. 2.7). Osim toga, na poslužitelju nije onemogućeno izlistavanje direktorija. Napadač tako može listati direktorije i pronaći bilo koju datoteku sa sigurnosnim manama te ih kasnije i iskoristiti.

Kako je poslužitelj aplikacije izgrađen pomoću *Express* razvojnog okvira, *The X-Powered-By* zaglavlje s vrijednosti *Express* pretpostavljeno je dodano na svaki *HTTP* odgovor od poslužitelja. To može pomoći napadaču da se usmjeri samo na ranjivosti poslužitelja izgrađenog preko *Expressa*.

Error There was a trouble with your login: TypeError: Cannot read property 'password' of undefined at loginAndBuildResponse (C:\Users\mario\Desktop\PhotographiusScope\server\user\login.route.ts:33:30) at Statement.<anonymous> (C:\Users\mario\Desktop\PhotographiusScope\server\user\login.route.ts:23:13) at Statement.replacement (C:\Users\mario\Desktop\PhotographiusScope\node_modules\sqlite3\lib\trace.js:19:31) at Statement.replacement (C:\Users\mario\Desktop\PhotographiusScope\node_modules\sqlite3\lib\trace.js:19:31) ✕

LOGIN

EMAIL:

zrtva@mail.hr

PASSWORD:

.....

Login

Sl. 2.7 Ispis greške pri neuspješnoj prijavi

2.4.2. Zaštita od loših sigurnosnih postavki

Za zaštitu od spomenutih loših sigurnosnih postavki u *PhotographiusScope* aplikaciji potrebno je implementirati sljedeće mjere. Prije svega onemogućiti izlistavanje direktorija na poslužitelju (pretpostavljena opcija u *Expressu*) te *The X-Powered-By* zaglavlje. Uklonjeno je ispisivanje grešaka koje daju cijeli trag stoga te se umjesto njih šalju generične poruke koje ne odaju previše informacija, poput „Adresa elektroničke pošte ili lozinka nisu točni“ koja se prikazuje kod neuspješne prijave. Tako primjerice potencijalni napadač prilikom *credential stuffinga* ili napada sirovom snagom ne zna je li krivo unesena lozinka ili adresa elektroničke pošte žrtve.

Za efikasno upravljanje greškama na klijentskoj strani mogu se iskoristiti *Angularovi* mehanizmi: *ErrorHandler* te *HttpInterceptor*. Postoje dvije kategorije grešaka, one koje dolaze s klijentske te one koje dolaze s poslužiteljske strane. Provjeravanjem je li greška instanca *ErrorEvent* može se zaključiti kojoj kategoriji greška pripada.

ErrorHandler je klasa koja sadrži metodu *handleError(error:any)* te pretpostavljena implementacija ove klase ispisuje grešku u konzolu. Za drukčije upravljanje i ispisivanje greške dovoljno je napraviti vlastitu klasu koja implementira *ErrorHandler* te sadrži

handleError metodu [27]. Implementirana klasa mora se uključiti u *catch* blok funkcija koje komuniciraju s poslužiteljem unutar zadane usluge.

Ovakvo upravljanje greškama dobro je rješenje za samo jednu uslugu, no kako prave *Angular* aplikacije sadrže velik broj usluga, uključivanje ove klase u *catch* blokove funkcija svih usluga primjer je antiobrasca u *Angular* razvoju.

Za upravljanje greškama globalno, može se koristiti *Angularov HttpInterceptor*. *HttpInterceptor* omogućuje presretanje *HTTP* zahtjeva i odgovora radi njihove transformacije ili upravljanja prije prosljeđivanja. Ovo je općenito koristan alat koji omogućuje i modifikaciju zaglavlja, kao i formata podataka. Za korištenje *HttpInterceptora* pri upravljanju s greškama, potrebno je kreirati klasu koja će implementirati *HttpInterceptor* sučelje i njegovu funkciju *intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>* u kojoj se definira kako će se greška obraditi [28]. S ovim načinom nije potrebno implementiranu klasu dodavati u *catch* blokove, već samo unutar polja *providers* u korijenskom modulu.

2.5. Međustranično skriptiranje (XSS)

Napadi međustraničnim skriptiranjem (eng. *Cross-site scripting*) su vrsta umetanja u kojima se maliciozne skripte umeću u inače dobroćudnu web stranicu. Napad se izvršava kada napadač koristi web aplikaciju za slanje malicioznog kôda u obliku skripti do drugog krajnjeg korisnika čiji ju preglednik izvršava jer ne zna da se toj skripti ne može vjerovati. Kako misli da skripta dolazi od pouzdanog izvora, maliciozna skripta može pristupati kolačićima, identifikatorima sjednice i drugim osjetljivim informacijama koje preglednik zadržava i koristi na toj stranici [29].

Mane koje dopuštaju da ovi napadi uspiju su široko rasprostranjene i događaju se na svakom mjestu u kojemu web aplikacija uzima unos od korisnika bez prethodne validacije i sanitizacije, procesa koji omogućuje eliminaciju nesigurnih znakova iz nesigurne vrijednosti metodama uklanjanja, zamjene ili izbjegavanja znakova. Zbog toga se ovi napadi nalaze na 7. mjestu *OWASP*-ove liste sigurnosnih prijetnji za web aplikacije te čine čak 40 % svih *cyber* napada u 2019. godini prema *PreciseSecurity.com* istraživanju [30].

U siječnju 2020. godine otkrivene su XSS ranjivosti na poznatoj aplikaciji *TikTok*, odnosno njihovoj poddomeni <https://ads.tiktok.com/>. Napad je bio moguć kroz obrazac za pretragu, gdje je napadač mogao unositi maliciozne skripte [31].

XSS napadi mogu se kategorizirati u tri grupe: pohranjeni, reflektirani te bazirani na *DOM*-u.

Pohranjeni XSS napadi su oni napadi u kojima se umetnuta skripta trajno pohranjuje na ciljanom poslužitelju (bazi podataka). Tada žrtva pri zahtijevanju određene informacije od poslužitelja, povlači i malicioznu skriptu.

Kod reflektiranog napada dolazi kada aplikacija prima podatke (malicioznu skriptu) u *HTTP* zahtjevu i uključuje te podatke u odgovor na nesiguran način.

Napadi bazirani na *DOM*-u napadi su u kojima se maliciozne skripte izvode kao rezultat modificiranja *DOM* okoline u pregledniku žrtve. Sama stranica se ne mijenja, ali sam kôd na klijentskoj strani ponaša se drugačije zbog malicioznih promjena koje su se dogodile u *DOM* okolini [29].

Kako su pohranjeni napadi najopasniji zbog njihove moguće trajne pohrane na poslužitelju, fokus ovoga rada bit će na tom tipu XSS napada.

2.5.1. Primjer međustraničnog skriptiranja

U *PhotographiusScope* aplikaciji vidljiv je primjer uspješno izvedenog pohranjenog XSS napada. Ona dopušta korisniku da prilikom registracije unese poveznicu na svoj profil sa *Secursy* portala (izmišljena stranica za pohranu profila korisnika). Kako se unos ne sanitizira ni validira, napadač tu može upisati malicioznu skriptu poput: „`javascript:document.location="http://localhost:9000/api/addmessage/receiver=napadac@mail.hr&content="+document.cookie;document.body.innerHTML = " ";`”. Tada će žrtva klikom na poveznicu *Secursy* profila napadača zapravo pokrenuti malicioznu skriptu koja šalje poruku napadaču sadržavajući kolačiće žrtve koje napadač kasnije može iskoristiti. Slikoviti prikaz opisanog napada je vidljiv na slici (Sl. 2.8).



Sl. 2.8 Primjer uspješnog XSS napada

2.5.2. Zaštita od međustraničnog skriptiranja

Što je tiče zaštite protiv XSS napada na poslužiteljskoj strani, nužna je sanitizacija i validacija svih nesigurnih vrijednosti koje stižu do poslužitelja. Za se mogu koristiti paketi treće strane ili vlastito napravljene funkcije. U *PhotographiusScope* aplikaciji dodan je paket treće strane *express-validator* koji radi propisnu sanitizaciju ulaznih podataka na poslužitelju.

Osim toga, na kolačić s identifikatorom sjednice dodan je atribut *HttpOnly* koji onemogućuje da se kolačiću pristupi kroz klijentsku skriptu [32].

Na klijentskoj strani *Angular* pruža dovoljan nivo zaštite jer pretpostavljeno tretira sve vrijednosti kao nesigurne da bi se izbjeglo da maliciozni kôd pristupi *DOM*-u što je početak XSS napada. Kada se vrijednost unese u *DOM* putem predloška, svojstva, atributa, stila, povezivanja klasa ili interpolacije, *Angular* sanitizira nepouzidane vrijednosti [33].

Sanitizacija u Angularu

U mnogim slučajevima, sanitizacija uopće ne mijenja vrijednost jer ovisi o kontekstu, na primjer ono što je bezazleno za *CSS* može biti potencijalno opasno za *URL*. *Angular* sanitizira nesigurne vrijednosti za *HTML*, *CSS* i *URL*-ove. U razvojnom načinu rada *Angular* ispisuje upozorenja u konzoli kada se neka vrijednost promijenila zbog sanitizacije [33].

Primjer sanitizacije dan je na slici (Sl. 2.9). Vrijednost sadržaja *htmlSnippet* koja se provjerava je: „`ANGULAR <script>alert(\"XSS\")</script>`“. Taj sadržaj se prvo interpolira unutar dvostrukih vitičastih zagrada, a zatim se povezuje s *innerHTML* svojstvom. Interpolirani sadržaj uvijek se izbjegava, to jest u ovome slučaju maliciozni kôd neće se izvršiti kao skripta nego se sadržaj samo ispisuje kao tekst. Što se tiče malicioznoga kôda unutar *innerHTML* svojstva, *Angular* prepoznaje `<script>` oznake, tretira vrijednost kao nesigurnu te ju automatski sanitizira, što rezultira uklanjanjem spomenutih `<script>` oznaka, ali istovremeno zadržava ostatak sigurnog sadržaja koji se izvršava kao ispravan *HTML* kôd.


```
<p>{{htmlSnippet}}</p>  
<p [innerHTML]="htmlSnippet"></p>
```

```
<b>ANGULAR</b> <script>alert("XSS")</script>  
ANGULAR
```

Sl. 2.9 Primjer sanitizacije u *Angularu*

Vjerovanje sigurnim vrijednostima

Ponekad aplikacije trebaju uključiti izvršivi kôd koji se može koristiti za izgradnju potencijalno opasnog *URL*-a. Da bi se prevenirala automatska sanitizacija u tim situacijama, može se koristiti *Angularova* klasa *DomSanitizer* (Kôd 2.5) koja sadrži šest metoda pomoću kojih se *Angularu* dojavljuje da toj specifičnoj vrijednosti može vjerovati te da će ona uvijek biti sigurna, a to su: *sanitize*, *bypassSecurityTrustHtml*, *bypassSecurityTrustScript*, *bypassSecurityTrustStyle*, *bypassSecurityTrustUrl* te *BypassSecurityTrustResourceUrl* [33].

```
export enum SecurityContext {NONE, HTML, STYLE, SCRIPT, URL,  
    RESOURCE_URL}  
export abstract class DomSanitizer implements Sanitizer {  
    abstract sanitize(context: SecurityContext,value:  
        SafeValue|string|null): string|null;  
    abstract bypassSecurityTrustHtml(value: string): SafeHtml;  
    abstract bypassSecurityTrustStyle(value:string): SafeStyle;  
    abstract bypassSecurityTrustScript(value:string): SafeScript;  
    abstract bypassSecurityTrustUrl(value: string): SafeUrl;  
    abstract bypassSecurityTrustResourceUrl(value: string):  
        SafeResourceUrl;  
}
```

Kôd 2.5 – Programski kôd razreda *DomSanitizer*

PhotographiusScope aplikacija vjeruje korisničkom unosu poveznice za *Secursy* profil, odnosno koristi *bypassSecurityTrustUrl* metodu za sprječavanje sanitizacije spomenutog unosa kojega nije ni validirala. Zbog toga je i bilo moguće izvoditi *XSS* napade. Da bi se to spriječilo, preporučuje se ne koristiti nijednu od spomenutih metoda, ali ako je to nužno (kao u ovom slučaju), vrijednost se prije toga mora validirati kao što je i obavljeno u sigurnoj verziji aplikacije. Za unesenu korisničku vrijednost, provjerilo se je li ona stvarno *URL* koji pripada domeni *Secursy* web aplikacije te jedino ako je, vrijednost se proslijedila *bypassSecurityTrustUrl* metodi.

2.6. Međustranično lažiranje zahtjeva (CSRF)

Međustranično lažiranje zahtjeva (eng. *Cross-site request forgery*) napad je koji prisiljava krajnjeg korisnika na izvršavanje neželjenih akcija na web aplikaciji u kojoj je trenutno prijavljen. Posebno je orijentiran na zahtjeve koji uzrokuju promjenu stanja poput transfera novca ili promjene lozinke, a ne na krađu podataka jer napadač ne može vidjeti odgovor na lažirani zahtjev (žrtva ga vidi) [34].

CSRF ranjivosti poznate su te u nekim slučajevima i iskorištene još od 2001. godine [35]. 2018. godine usmjerivači tvrtke *Draytek* našli su se na udaru *CSRF* napada koji su iskorišteni za promjenu *DNS* konfiguracije usmjerivača. Detalji nisu poznati, zbog očitih sigurnosnih razloga [36].

Napad se bazira na tome što preglednici automatski uključuju autentikacijske podatke (poput kolačića s identifikatorom sjednice) korištene na nekoj domeni u svaki zahtjev poslan na tu domenu. Ako je korisnik trenutno prijavljen u web aplikaciji, ona neće moći razlikovati lažirani zahtjev od onoga legitimnog poslanog od žrtve [34].

CSRF se ne nalazi u *OWASP*-ovih top 10 ranjivosti 2017. godine, iako se u verziji iz 2013. nalazio na 8. mjestu. Razlozi njegovog pada su složenost izvedbe te sve veći broj dostupnih zaštita.

2.6.1. Primjer međustraničnog lažiranja zahtjeva

CSRF napad moguće je izvesti u *PhotographiusScope* aplikaciji te je primjer uspješno izvedenog napada dan u nastavku.

Napadač odlučuje iskoristiti socijalni inženjering⁴ te u poruci šalje žrtvi poveznicu uz objašnjenje da se na toj poveznici krije ostatak poslanog novca (Sl. 2.10). Ako žrtva odluči kliknuti na tu poveznicu, preusmjerena je na drugu stranicu te žrtva nesvjesno šalje napadaču 10000 dolara (Sl. 2.11). Ono čega žrtva nije bila svjesna je da se otvaranjem poveznice pozvala operacija „*sendmoney/receiver=napadac@mail.hr&amount=10000*“

⁴ Niz tehnika pomoću kojih pojedinac, iskorištavanjem ljudskih pogrešaka i slabosti, utječe na drugog pojedinca kako bi ga naveo da učini nešto što nije u njegovom interesu [37].

koja je na ranjivoj aplikaciji obavila izvršavanje spomenute transakcije. Napad je uspio jer pregledniku izgleda kao da je zahtjev stigao od legitimnog korisnika jer je taj zahtjev sadržavao autentikacijske podatke korisnika (kolačić s identifikatorom sjednice korisnika). Ovo je bio primjer *CSRF GET* napada. Osim *GET* scenarija, postoje i *POST*, *PUT* i *DELETE* koji se samo razlikuju u načinu na koje ih žrtva pokreće.

Sender	Receiver	Date	
<u>Napadac</u> <u>Napadac</u>	<u>Žrtva</u> <u>Žrtva</u>	2020-06-23 22:34:22	Hey, the rest of money I owe you is on this link -> Link

Sl. 2.10 Primjer socijalnog inženjeringa u ostvarenju *CSRF* napada

TRANSACTIONS			
New transaction			
Amount	Sender	Receiver	Date
10000\$	zrtva@mail.hr	napadac@mail.hr	2020-06-23 22:42:26

Sl. 2.11 Transakcija obavljena preko *CSRF* napada

2.6.2. Zaštita od međustraničnog lažiranja zahtjeva

Kao osnovna mjera zaštite protiv *CSRF* napada, u *PhotographiusScope* aplikaciji implementirana je *double submit cookie* tehnika. Kod ove tehnike pri autenticiranju korisnika, poslužitelj kreira kriptografski snažnu nasumičnu vrijednost (token) koju postavlja u novi kolačić (pretpostavljenog imena *XSRF TOKEN*) kojeg preglednik također uključuje u svaki zahtjev. Na klijentskoj strani potrebno je uzeti spomenuti token i postaviti ga u odgovarajuće zaglavlje (pretpostavljenog imena *X-XSRF TOKEN*) tako da će svaki zahtjev koji dolazi s klijentske strane sadržavati zaglavlje koje sadrži nasumičnu vrijednost s poslužitelja [38].

Angularov HttpClientModule automatski implementira klijentski dio zaštite. Pri svakom zahtjevu, uzima token iz *XSRF TOKEN* kolačića te ga postavlja u *X-XSRF-TOKEN*

zaglavlje. Ako se žele koristiti druga imena za kolačić ili zaglavlje, u korijenski modul je potrebno dodati *HttpClientXsrfModul* modul. Modul u metodi *withOptions()* definira naziv kolačića iz kojeg će uzeti nasumičnu vrijednost te naziv zaglavlja u koji će umetnuti spomenutu vrijednost. Ako nazivi kolačića i zaglavlja nisu definirani, *Angular* će pretpostaviti da su *XSRF-TOKEN* i *X-XSRF-TOKEN*. Osim *withOptions()* metode, modul sadrži i *disable()* metodu čije pozivanje onemogućava pretpostavljenu zaštitu.

Dakle, nakon što se vrijednost iz *XSRF-TOKEN* kolačića postavi u zaglavlje, potrebno je implementirati provjeru na poslužitelju kojoj se prije izvršavanja svakog zahtjeva provjerava je li vrijednost unutar *XSRF-TOKEN* kolačića jednaka vrijednosti u *X-XSRF-TOKEN* zaglavlju (spomenuti kolačić i zaglavlje nalaze se unutar *HTTP* zahtjeva) [39].

Ova tehnika efikasna je jer svi preglednici implementiraju politiku istoga izvorišta⁵, što znači da samo kôd sa stranice na kojoj su kolačići postavljeni može čitati kolačiće s te stranice te postavljati proizvoljna zaglavlja na zahtjevima za tu stranicu. Odnosno, samo dotična web aplikacija smije čitati ovaj token iz kolačića te postavljati proizvoljno zaglavlje što osigurava da nijedan zahtjev izvan domene aplikacije ne može uspjeti.

Osim *double submit cookie* tehnike, za sprječavanje *CSRF*-a dodan je i *SameSite* atribut za kolačiće koji nose identifikator sjednice. Ovaj atribut pomaže pregledniku utvrditi treba li poslati kolačiće za zahtjeve izvan domene. Moguće vrijednosti atributa su *Lax*, *Strict* ili *None*. *Strict* vrijednost onemogućuje da se kolačići šalju pri svim zahtjevima izvan domene (čak i za legitimne zahtjeve), dok *Lax* omogućuje njihovo slanje izvan domene samo za zahtjeve koji uzrokuju navigaciju gornje razine, odnosno one koje mijenjaju *URL* u adresnoj traci. *None* vrijednost označava da se kolačići mogu slati pri svim zahtjevima izvan domene te se ona prvotno koristila u *PhotographiusScope* aplikaciji sve dok nije zamijenjena sa *Strict* vrijednošću [38].

Među ostalim mjerama prevencije koje su implementirane su zahtijevanje korisničke interakcije kod visoko osjetljivih operacija (potrebna je lozinka za potvrdu novčane transakcije), provjeravanje *Referer*⁶ zaglavlja te nekorištenje *GET* zahtjeva kod operacija koje mijenjaju stanja.

⁵ Pravilo pomoću kojeg web preglednici dopuštaju skriptama s jedne web stranice da pristupe sadržaju druge, ali jedino ako obje stranice imaju isto izvorište.

⁶ Zaglavlje koje sadrži vrijednost adrese prijašnje web stranice s koje je korisnik posjetio trenutnu.

3. Najbolja praksa

U nastavku su opisani neki od elemenata najbolje prakse pri radu s *Angularom*, čiji se srodni koncepti mogu primijeniti i na ostale slične razvojne okvire za stvaranje web aplikacija. Od njih se ističu eksplicitne metode koje su u potpunosti orijentirane na poboljšanje sigurnosti aplikacije, a koje već nisu istaknute kao metode zaštite kod prikazanih napada i ranjivosti. Osim njih tu su i implicitne metode za ostvarenje najbolje prakse programiranja u *Angularu* poput primjenjivanja raznih oblikovnih obrazaca i ostvarenja kôda visoke kvalitete, dizajniranja praktične arhitekture aplikacije te uporabe testova sa svrhom eliminacije grešaka. Cilj ovih metoda stvoriti je aplikaciju visoke kvalitete i funkcionalnosti, uz veliku stopu čitljivosti i mogućnosti ponovne uporabe što će i olakšati implementiranje i održavanje sigurnosti.

Sve navedene metode najbolje prakse kasnije su implementirane u *PhotographiusScope* aplikaciji.

3.1. Ahead-of-Time (AOT) prevoditelj

Angularov Ahead-of-Time prevoditelj (također poznat kao *offline* prevoditelj predložaka) pretvara *Angular HTML* i *TypeScript* kôd u efikasni *JavaScript* kôd tijekom faze izgradnje prije nego se taj kôd preuzme i pokrene od strane preglednika, za razliku od *Just-in-Time* prevoditelja koji prevodi aplikaciju u pregledniku prilikom njenog pokretanja. Ovakav način prevođenja aplikacije prilikom izgradnje omogućuje njeno brže učitavanje u pregledniku. Osim bržeg učitavanja, *Ahead-of-Time* prevoditelj osigurava i postojanje manje asinkronih zahtjeva te manju potrebnu količinu *Angularovih* datoteka za preuzimanje. Također ranije uočava greške unutar predloška te najbitnije, osigurava bolju sigurnost aplikacije.

Osigurava bolju sigurnost aplikacije tako što sprječava cijelu klasu ranjivosti umetanja predložaka. Ranjivosti nastaju pri dinamičkom generiranju predložaka jer *Angular* vjeruje kôdu u predlošku, a generiranjem predloška koji sadrži korisnički unos, zaobišla bi se

ugrađena *Angularova* zaštita. *Offline* prevoditelj eliminira potrebu za dinamičkim generiranjem predložaka [40].

Kada se pokrenu naredbe *ng build* (samo izgradnja) ili *ng serve* (izgradnja i lokalno pokretanje), vrsta prevođenja aplikacije ovisi o vrijednosti *aot* svojstva koje je specificirano u *angular.json* konfiguracijskoj datoteci. Od *Angular* 9 verzije to svojstvo ima vrijednost *true* jer je *Ahead-of-Time* pretpostavljeni prevoditelj. Pravila za korištenje *AOT* prevoditelja mogu se kontrolirati specificiranjem opcija prevođenja predloška unutar *tsconfig.json* konfiguracijske datoteke.

3.2. Izbjegavanje Angularovog rizičnog aplikacijskog programskog sučelja

Angular preporučuje izbjegavanje aplikacijskog programskog sučelja koje je unutar dokumentacije označeno kao „Sigurnosni rizik“. Među njima se nalazi već spomenuti *DomSanitizer* sa svojim *bypassSecurityTrust* metodama koje onemogućuju provođenje ugrađene *Angularove* sanitizacije nad proslijeđenom vrijednošću. Ipak, osim *DomSanitizer*-a tu je i *ElementRef* koji je puno češće u uporabi. Taj razred dozvoljava izravan pristup *DOM*-u što može aplikaciju napraviti ranjivom na *XSS* napade tako što će tako onemogućiti *Angularovu* automatsku sanitizaciju. Do istoga rizika će se doći i korištenjem drugih biblioteka (osim *Angularovih*) koje manipuliraju sadržajem *DOM*-a. Umjesto *ElementRef*-a, preporučuje se korištenje *Angularovih* predložaka te povezivanje podataka. Alternativno, moguće je koristiti i *Renderer2* koji osigurava aplikacijsko programsko sučelje koje se može koristiti na siguran način čak i kada nije omogućeno izravno pristupanje izvornom elementu [41].

3.3. Aplikacijski protokol HTTPS

HTTP je aplikacijski protokol koji se koristi za prijenos podataka preko mreže i to koristeći dvije vrste poruka: zahtjeve i odgovore. *HTTP* ne sadrži nikakvu enkripciju imajući za posljedicu da će zahtjevi i odgovori biti poslani u čistom obliku, što znači da ih svatko može čitati [42].

HTTPS je sigurna verzija *HTTP* protokola. Omogućuje enkripciju podataka koji se šalju koristeći *SSL/TLS* tako da napadači ne mogu krasti podatke. Ovo je posebno bitno pri slanju osjetljivih podataka, kao što su lozinke. Uz to, *SSL/TLS* potvrđuje identitet web poslužitelja. Navedene prednosti sprječavaju niz mogućih napada, od kojih je sigurno najpoznatiji *man-in-the-middle*⁷ [43].

Uzimajući u obzir navedene prednosti, jasno je da bi svaka web stranica trebala koristiti *HTTPS*. U modernim preglednicima poput *Google Chromea*, web stranice koje ne koriste *HTTPS* označene su kao nesigurne [44].

Angular pretpostavljeno pokreće aplikaciju preko *HTTP*-a, ali omogućuje i pokretanje preko *HTTPS*-a koristeći *SSL*. Za to je potrebno definirati tri parametra zajedno s *ng serve* naredbom unutar *package.json* datoteke:

- `--ssl <Boolean>`: pretpostavljeno postavljeno na *false* - omogućavanje/onemogućavanje *SSL*-a.
- `--ssl-cert <string>`: pretpostavljeno postavljeno na „*ssl/server.crt*” - put do certifikata.
- `--ssl-key <string>`: pretpostavljeno postavljeno na „*ssl/server.key*” - put do privatnog ključa [45].

⁷ Vrsta napada u kojoj napadač upada u komunikaciju klijenta i poslužitelja s ciljem prisluškivanja ili izmjene informacija.

3.4. Sigurnosna zaglavlja

Kada korisnik posjeti stranicu koristeći preglednik, poslužitelj odgovara sa zaglavlјima *HTTP* odgovora. Ova zaglavlja upućuju preglednik na to kako se ponašati tijekom komunikacije s posjećenom stranicom. Mogu se koristiti za definiranje komunikacije kao i poboljšanje sigurnosti web aplikacije. Sigurnosna zaglavlja potrebno je implementirati na strani poslužitelja te ako je on izgrađen koristeći *Express* razvojni okvir (kao u *PhotographiusScope* aplikaciji), može se koristiti paket *Helmet*.

Helmet je kolekcija 12 manjih funkcija koje postavljaju zaglavlja na *HTTP* odgovore. *Helmetove* funkcije koje su korištene u sigurnoj verziji *PhotographiusScope* aplikacije su:

- *dnsPrefetchControl* – postavlja *X-DNS-Prefetch-Control* zaglavlje na vrijednost *off* i time sprječava preglednik da izvrši *DNS* predučitavanje.
- *frameguard* – postavlja *X-Frame-Options* zaglavlje na vrijednost *DENY* čime sprječava postavljanje dotične web stranice unutar bilo koje *iframe* oznake (onemogućuje *clickjacking*⁸ napade).
- *hidePoweredBy* – uklanja automatski postavljeno *X-Powered-By* zaglavlje.
- *hsts* – postavlja *Strict-Transport-Security* zaglavlje i tako upućuje preglednik da ostane na *HTTPS*-u i nikad ne posjećuje nesigurnu *HTTP* verziju.
- *ieNoOpen* – postavlja *X-Download-Options* zaglavlje na vrijednost *noopen* čime se sprječava da starije verzije *Internet Explorera* dopuste izvršavanje preuzetog malicioznog *HTML* kôda u kontekstu dotične aplikacije.
- *noSniff* – postavlja *X-Content-Type-Options* zaglavlje na vrijednost *nosniff* čime prevenira preglednik od pogađanja *MIME* tipa, što može imati sigurnosne implikacije.
- *xssFilter* – postavlja *X-XSS-Protection* zaglavlje na vrijednost *1; mode=block* čime se preveniraju reflektirani *XSS* napadi.
- *featurePolicy* – omogućuje ograničavanje korištenja značajki preglednika.

⁸ Napad u kojem se žrtva prevari s ciljem klikanja na element web stranice koji je nevidljiv.

- *referrerPolicy* – postavlja *Referrer* zaglavlje na vrijednost *no-referrer* (onemogućavanje zaglavlja) ili *same-origin* (omogućavanje samo za stranice iz istoga izvora).
- *contentSecurityPolicy* – postavlja *Content-Security-Policy* zaglavlje koje definira politiku sigurnosti sadržaja [46].

Bitno je napomenuti da su prvih 7 nabrojanih funkcija bile pretpostavljeno omogućene, dok je za ostatak bila potrebna posebna konfiguracija. Najvažnije od ovih zaglavlja je *Content-Security-Policy* o čemu je riječ u nastavku.

3.4.1. Politika sigurnosti sadržaja

Politika sigurnosti sadržaja predstavlja dodatni sloj sigurnosti koji pomaže detektirati i ukloniti određene tipove napada, uključujući XSS i napade umetanjem. Prilikom rada koriste brojne smjernice od kojih svaka definira izvor određenog sadržaja koji preglednik smije učitati pri korištenju te web aplikacije. Sadržaj može uključivati *JavaScript* skripte, *CSS* stilove, *HTML* okvire, fontove, slike te brojne druge mogućnosti. Preglednici koji ne podržavaju politiku sigurnosti sadržaja i dalje surađuju s poslužiteljima koji ju implementiraju i obrnuto. Ako ju preglednici ne podržavaju, samo ju ignoriraju, funkcionirajući kao i prije, koristeći standardno pretpostavljenu politiku istoga izvorišta. Ista stvar je i kada preglednik podržava politiku sigurnosti sadržaja, ali ju web aplikacija ne nudi. No danas većina modernih preglednika podržava politiku sigurnosti sadržaja [47].

Helmetova contentSecurityPolicy funkcija podržava uporabu 23 različite smjernice od kojih su u *PhotographiusScope* aplikaciji korištene:

- *baseUri* – ograničava *URL*-ove koji se mogu koristiti u baznom elementu dokumenta.
- *connectSrc* – ograničava *URL*-ove koji se mogu učitati koristeći sučelje skripti.
- *defaultSrc* – služi kao rezerva za sve ostale smjernice koje provjeravaju izvor sadržaja za učitavanja.
- *fontSrc* – specificira izvore iz kojih se fontovi mogu učitati.

- *formAction* – ograničava *URL*-ove koji se mogu koristiti kao meta za slanje obrasca iz danog konteksta.
- *frameAncestors* – specificira koji izvori mogu na stranicu postavljati sadržaj unutar `<frame>`, `<iframe>`, `<object>`, `<embed>` ili `<applet>` oznaka. U osnovi, zaštita protiv *clickjacking* napada.
- *frameSrc* – specificira valjane izvore za učitavanje ugniježđenih konteksta koristeći elemente poput `<frame>` i `<iframe>`.
- *imgSrc* – specificira izvore iz kojih se slike mogu učitati.
- *scriptSrc* – specificira izvore iz kojih se skripte mogu učitati.
- *styleSrc* – specificira izvore iz kojih se stilovi mogu učitati [47].

Sve nabrojane smjernice kao prvu vrijednost sadrže *self* što označava da je dopušteno učitavanje određenog sadržaja iz domene te web aplikacije. To pogotovo vrijedi za smjernicu *defaultSrc* koja će tako spriječiti učitavanje određenog sadržaja izvan domene web aplikacije za koji nije definirana odgovarajuća smjernica.

3.5. Najbolja praksa programiranja u Angularu

Implementacijom najbolje prakse programiranja u *Angularu* stvorit će se čišći, jednostavniji i skalabilniji kôd za lakše održavanje i testiranje uz minimizaciju vremena za uklanjanje grešaka. Ona se sastoji od niza uputa koje se tiču sintakse, konvencija i strukture aplikacije od kojih su najbitnije:

- Struktura komponente – za svaki dio komponente koristi se iduća konvencija imenovanja: *nazivKomponente.component.ts/ html/ css/ spec*. Ovakvo imenovanje datoteka komponenti čini strukturu projekta čitljivom i sažetom.
- Jedinstvena odgovornost – oblikovni obrazac u programiranju koji nalaže da bi svaki modul, razred ili funkcija trebala imati samo jednu odgovornost. Potrebno ga je primijeniti na sve komponente, klase i ostale datoteke.
- *DRY* princip (eng. *Do not Repeat Yourself*) – oblikovni obrazac kojemu je načelo neponavljanje kôda. Korištenje istog kôda na različitim mjestima rezultira

višestrukim promjenama ako se želi promijeniti logika tog kôda. Ovakav način programiranja otežava održavanje i podložan je greškama. Zbog toga je potrebno ponavljajući kôd izlučiti na jedno mjesto te koristiti na ostalim potrebnim mjestima.

- Imenovanje – imenovanje je jedno od najvažnijih principa za održavanje čitljivosti od čega je najbitnije držati se konzistentnog načina imenovanja uz korištenje razumljivih i značajnih naziva. Zbog toga su izdvojene sljedeće konvencije:
 1. Imenovanje datoteka potrebno je obavljati po principu *značajka.tip.ts*, pri čemu se koristi crtica za odvajanje riječi (na primjer: *add-message.component.ts*).
 2. Za imena razreda koristi se *camelCase* princip imenovanja s velikim početnim slovom (na primjer: *MessageService*), za varijable, metode i selektore direktivi *camelCase* s malim početnim slovom (na primjer: *rbacAllow*) te za konstante *snake_case* s velikim slovima (na primjer: *SESSION_DURATION*).
- Struktura aplikacije:
 1. Sav kôd *Angularove* aplikacije treba se nalaziti u *src* mapi, svaka komponenta i usluga ima vlastitu datoteku, a sve datoteke treće strane nalaze se izvan *src* mape.
 2. Struktura aplikacije treba biti takva da se kôd može brzo locirati i identificirati na prvi pogled uz održavanje najsažetije strukture i uz što manje ponavljanja.
 3. Savjetuje se kreiranje mape za svaku komponentu koja ima više datoteka. Naziv mape treba predstavljati funkcionalnost koju komponenta pruža.
- Ostalo:
 1. Deklariranje varijable ili konstante – Kad god je moguće, potrebno je deklarirati varijablu ili konstantu s predviđenim tipom, nego s *any*. Korištenje opcije *any* može izazvati neželjene probleme kada se očekuje drukčiji tip varijable (ili konstante) nego što je. Uz to, deklariranje tipova unutar aplikacije stvara bolje i lakše refaktoriranje.

2. Komponente trebaju sadržavati samo logiku vezanu uz prikazivanje – Komponente su dizajnirane za prezentacijske svrhe te kontroliraju ono što bi korisnik trebao vidjeti. Zbog toga bi sva poslovna logika trebala biti unutar vlastite metode/usluge. Isto tako, nikakva logika prikaza ne bi se trebala nalaziti unutar predložaka.
3. Korištenje malih funkcija – male funkcije su funkcije do 75 linija kôda. Ako je metoda duža od toga, trebala bi biti podijeljena na više metoda ili datoteka. Male funkcije su lakše za testiranje, čitanje i održavanje te potiču ponovnu uporabu [48].

3.6. Ostale metode

Među ostalim metodama najbolje prakse za ostvarenje funkcionalne i sigurne *Angular* aplikacije ističu se sljedeće:

- Ažuriranje *Angular*ovih biblioteka – *Angular* regularno ažurira svoje biblioteke te ta ažuriranja mogu popraviti sigurnosne mane koje su otkrivene u prošlim verzijama. Nekorištenje najnovije verzije *Angulara* može dovesti do ozbiljnog sigurnosnog rizika ako potencijalni napadač odluči iskoristiti poznate ranjivosti korištene zastarjele verzije [33].
- Nemijenjanje kopije *Angulara* – Općenito je loša ideja modificirati bilo koju *Angular*ovu biblioteku jer je tada razvojnik vezan za specifičnu verziju *Angulara*. Jednom kada se biblioteka izmjeni, često se ne mogu primijeniti bitne sigurnosne nadogradnje i poboljšanja bez da se utječe na funkcionalnost izgrađene aplikacije. Najbolje rješenje je podijeliti zamišljene promjene s *Angular* zajednicom. Tako bi članovi zajednice mogli razmotriti predložene promjene te ih možda uključili u iduće izdanje verzije, naravno ako su one korisne i sigurne [33].
- Provođenje *Unit* testova – *Unit* testiranjem odlučuje se je li testirani dio kôda spreman za uporabu. Cilj je izolirati svaki dio programa i testirati rade li korektno individualni dijelovi. Svaka *Angular* aplikacija dolazi s predinstaliranim alatima za testiranje: *Jasmine* i *Karma*. *Jasmine* se koristi za kreiranje testova, dok *Karma* za njihovo izvršavanje. Osim njih, *Angular* nudi *TestBed* i *async* programska sučelja koja

olakšavaju testiranja asinkronog kôda, komponenti, direktivi i usluga. Razlozi za *Unit* testiranje brojni su, ali ističu se poboljšanje dizajna implementacije, dozvoljavanje refaktoriranja, rani pronalazak grešaka te dodavanje novih funkcionalnosti bez utjecaja na druge [7].

- Izbjegavanje komponenti s poznatim ranjivostima – Kako danas postoje brojne komponentne biblioteke treće strane, razvijanje aplikacije bez njihove pomoći čini se gotovo nemogućim. No te biblioteke možda sadrže poznate ranjivosti koje bi napadači mogli iskoristiti. Zato je biblioteke potrebno preuzimati od povjerljivog izvora te uvijek koristiti njenu najnoviju verziju. Osim toga, potrebno je raditi čestu reviziju aplikacije što se može postići s *npm*⁹ naredbom *npm audit* koja može otkriti korištenje komponenti s poznatim ranjivostima, dok naredba *npm audit fix* automatski instalira odgovarajuća dostupna ažuriranja [49].
- Prefiksiranje *JSON* odgovora – prefiksiranje *JSON* odgovora s poslužitelja prevenira *JSON* ranjivost pri kojoj napadač može čitati podatke iz *JSON* aplikacijskog programskog sučelja. Napad je jedino uspješan ako je *JSON* koji se vraća izvršiv kao *JavaScript*. Zato je na poslužitelju potrebno prefiksirati svaki *JSON* odgovor s nizom „*]]}',\n*“ koji ga čini neizvršivim. *Angularova HttpClient* biblioteka prepoznaje ovu konvenciju te automatski skida dodani niz sa svih *JSON* odgovora prije nego ih parsira [33].

⁹ Upravitelj paketa za *JavaScript* programske jezike.

4. Testiranje sigurnosti

Testiranje sigurnosti softvera proces je procjene i provjere sustava radi otkrivanja sigurnosnih rizika te ranjivosti sustava i njegovih podataka.

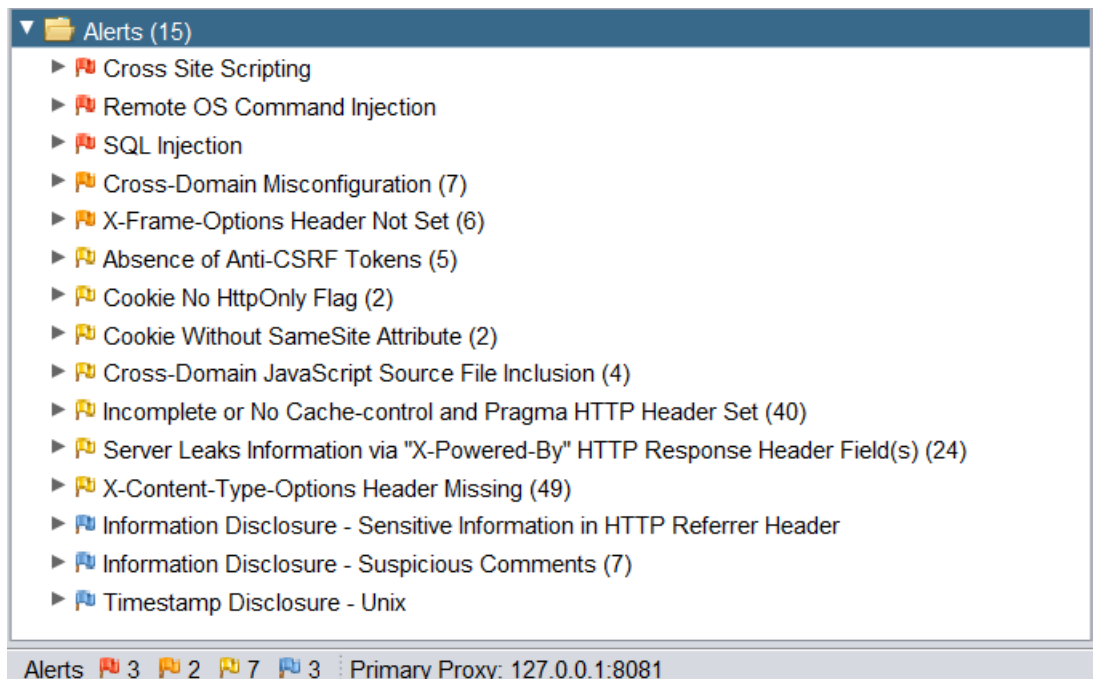
U svrhu dokazivanja poboljšanja sigurnosti testirane su dvije verzije *PhotographiusScope* aplikacije. Početna implementacija aplikacije koja predstavlja ranjivu verziju te sigurna verzija u kojoj su implementirane opisane metode zaštite i najbolje prakse. Testiranje se vršilo pomoću alata *OWASP ZAP (Zed Attack Proxy)* te online skenera za web aplikacije poput: *Mozilla Observatory*, *Checkbot: SEO, Web Speed & Security Checker* te *Security Headers*. Brojni od njih, osim prikaza mogućih ranjivosti i nedostataka od kojih testirana aplikacija pati, prikazuju i razne informacije te upozorenja koja mogu pomoći razvijatelju aplikacije pri rješavanju detektiranih problema. Rezultati testiranja ranjive i sigurne verzije aplikacije dani su u nastavku.

4.1. Testiranje OWASP ZAP alatom

OWASP ZAP besplatni je alat otvorenog kôda za penetracijsko testiranje kojega održava *OWASP*. U smislu načina rada, *ZAP* je u osnovi posrednik (eng. *proxy*) koji se nalazi između ispitivačkog preglednika i web aplikacije koje se testira tako da može presresti i provjeriti poruke poslane između preglednika i web aplikacije te ih zatim proslijediti do odredišta [50].

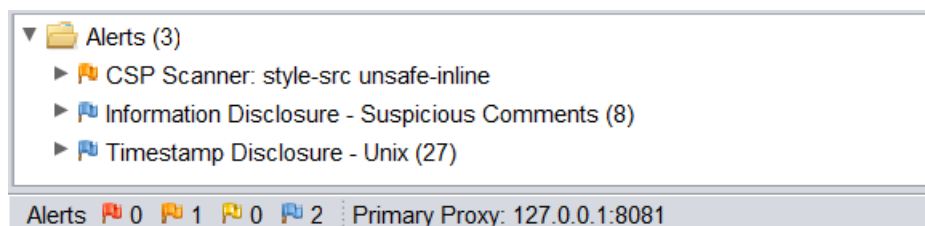
Izvještaj testiranja ranjive verzije *PhotographiusScope* aplikacije dan je na slici (Sl. 4.1). *ZAP* je za nju izdao 15 različitih upozorenja od kojih su 3 visokog rizika, 2 umjerenoga te 7 blagoga. Osim njih, tu su i 3 informativna upozorenja. Dakle *ZAP* je uspješno detektirao da je aplikacije ranjiva na *XSS*, umetanja *SQL* kôda i umetanja naredbi operacijskog sustava. Također, detektirao je da je aplikacija podložna *CSRF* napadima jer nedostaje *Anti-CSRF* token te kolačići ne sadrže *SameSite* atribut. Uz to, kolačići ne sadrže *HttpOnly* zastavicu. Što se tiče zaglavlja, detektirano je da *X-Frame-Options* i *X-Content-Type-Options* zaglavlja nisu postavljena, poslužitelj odaje informacije preko *X-Powered-By* zaglavlja što ukazuje na loše sigurnosne postavke, kao i da je *Cache-control* zaglavlje

postavljeno na *public* čime je moguće izlaganje osjetljivih podataka. Gledajući informativna upozorenja, važno je uočiti da ja *ZAP* detektirao moguće izlaganje osjetljivih informacija u *HTTP Referrer* zaglavlju. Radi se o identifikatoru sjednice postavljenom u *URL* što ukazuje na loše implementiranu autentikaciju.



Sl. 4.1 *OWASP ZAP* izvještaj testiranja ranjive verzije *PhotographiusScope* aplikacije

Nakon dodavanja opisanih mjera zaštite, kao i primjenjivanja metoda dobre prakse, obavljeno je novo testiranje na *PhotographiusScope* aplikaciji čiji je rezultat vidljiv na slici (Sl. 4.2). Sada i *ZAP* detektira da aplikacija nije više ranjiva na napade kojima je ranjiva verzija bila podložna te je većina prethodnih upozorenja postala stvar prošlosti. Jedino novo upozorenje koje se pojavilo upozorava na nesigurno *inline* stiliziranje unutar politike sigurnosti sadržaja, no ono nužno da bi se prikazali stilovi (eng. *styles*) generirani od strane *Angularovog* prevoditelja. Ostala informativna upozorenja ista su kao i ona kod ranjive verzije. Srećom, ne predstavljaju opasnost te se odnose na *Angularove* biblioteke koje se ne preporučuje modificirati.



Sl. 4.2 *OWASP ZAP* izvještaj testiranja sigurne verzije *PhotographiusScope* aplikacije

4.2. Testiranje Checkbot: SEO, Web Speed & Security Checker skenerom

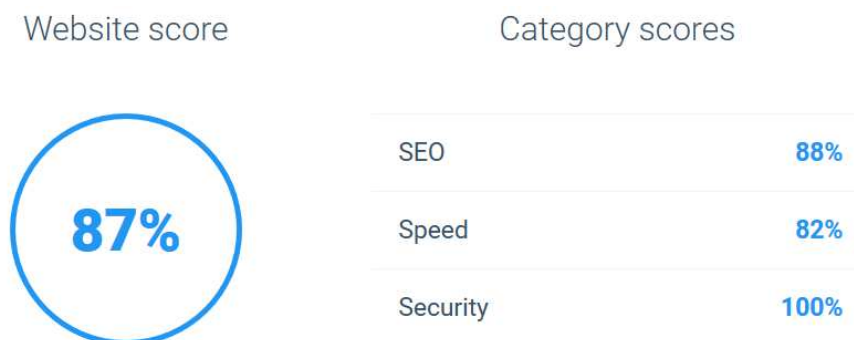
Checkbot je proširenje za *Google Chrome* preglednike koje djeluje kao *online* skener web aplikacija te pritom provjerava i ocjenjuje optimizaciju pretraživanja (eng. *SEO – Search Engine Optimization*), brzinu te najbitnije sigurnost. Pri ocjenjivanju navedenih elemenata web aplikacije, koristi preporuke brojnih web stručnjaka kao što su *Google*, *Mozilla*, *W3C* te *OWASP* [51].

Poražavajući rezultati skeniranja ranjive verzije *PhotographiusScope* aplikacije prikazani su na slici (Sl. 4.3). Vidljivo je da je ukupni rezultat aplikacije 57 %, dok je sigurnost na samo 23 %. Skener je uočio da web aplikacija ne koristi *HTTPS* kao ni *HSTS* te da se lozinke prenose *HTTP* protokolom. Osim toga, primijećeni su i već spomenuti problemi s *X-Frame-Options* i *X-Powered-By* zaglavljima, ali i nedostatak *X-XSS-Protection* zaglavlja.



Sl. 4.3 *Checkbot* rezultati skeniranja ranjive verzije *PhotographiusScope* aplikacije

Kako su navedene ranjivosti i nedostaci popravljani u sigurnoj verziji *PhotographiusScope* aplikacije, ponovnim testiranjem dobivaju se mnogo bolji rezultati. S izvještaja na slici (Sl. 4.4) vidljivo je da je sada ukupni rezultat aplikacije 87 %, dok je sigurnost na 100 %.



Sl. 4.4 *Checkbot* rezultati skeniranja sigurne verzije *PhotographiusScope* aplikacije


4.3. Testiranje Mozilla Observatory skenerom


Mozilla Observatory još je jedan *online* skener web aplikacija koji ocjenjuje isključivo sigurnost te dodjeljuje ocjene koje rangiraju od F do A+. Testira sadrži li aplikacija obranu od *XSS* i *man-in-the-middle* napada, curenja informacija te kompromitacije kolačića i mreže za dostavljanje sadržaja [52].

Testiranjem ranjive verzije *PhotographiusScope* aplikacije dobila se negativna ocjena (Sl. 4.5). Ukupni rezultat aplikacije je 0/100 jer aplikacija prolazi samo 4 od 11 testova. Najveći dio testova odnosio se na već prethodno detektirane probleme poput nedostatka *X-Content-Type-Options*, *X-Frame-Options* i *X-XSS-Protection* zaglavlja te nekorištenje *HTTPS*-a i *HSTS*-a. Što se tiče novootkrivenih problema, *Observatory* primjećuje da politika sigurnosti sadržaja uopće nije implementirana kao ni *Referrer* zaglavlje.

Nakon što je dodana odgovarajuća zaštita uz metode najbolje prakse, situacija se promijenila. Sada sigurna verzija aplikacije ima ocjenu A+ te rezultat 115/100 (Sl. 4.5). Aplikacija je dobila i dodatne bodove jer je implementirala i neke elemente sigurnosti koji

su opcionalni pri ocjenjivanju poput korištenja kolačića koji sadrže *Secure* i *HttpOnly* zaglavlje te *SameSite* atribut.

Scan Summary	
	Host: photographius-scope.herokuapp.com
	Scan ID #: 14781583
	Start Time: June 24, 2020 2:19 PM
	Duration: 4 seconds
	Score: 0/100
Tests Passed: 4/11	


Scan Summary	
	Host: photographius-scope-2.herokuapp.com
	Scan ID #: 14781554
	Start Time: June 24, 2020 2:16 PM
	Duration: 5 seconds
	Score: 115/100
Tests Passed: 11/11	


Sl. 4.5 Rezultati testiranja ranjive i sigurne verzije aplikacije u *Mozilla Observatory*ju

Iako je aplikacija dobila A+ ocjenu, to ne jamči da je ona potpuno sigurna jer postoje brojni elementi koje *Mozilla Observatory* ne testira, od kojih su najbitnije korištenje zastarjelih softverskih verzija, umetanja *SQL* kôda, nepropisno stvaranje i pohranjivanje lozinki [52].

4.4. Testiranje Security Headers skenerom

Testiranje je još provedeno koristeći *Security Headers* online skener. *Security Headers* pri testiranju provjerava postojanje već spomenutih zaglavlja (*X-Frame-Options*, *HSTS*, *X-Content-Type-Options*, *Referrer-Policy* i *Content-Security-Policy*) te dosad neotkrivenog *Feature-Policy* zaglavlja. Ranjiva verzija *PhotographiusScope* aplikacije dobiva ocjenu F, dok sigurna A. Rezultati testiranja obje verzije aplikacije dani su na slici (Sl. 4.6). *Inline* stiliziranje onemogućuje da sigurna verzija dobije A+ ocjenu.

Security Report Summary	
	Site: https://photographius-scope.herokuapp.com/
	IP Address: 52.209.180.106
	Report Time: 24 Jun 2020 12:26:33 UTC
	Headers: <div><div>✗ Strict-Transport-Security</div><div>✗ Content-Security-Policy</div><div>✗ X-Frame-Options</div><div>✗ X-Content-Type-Options</div><div>✗ Referrer-Policy</div><div>✗ Feature-Policy</div></div>

Security Report Summary	
	Site: https://photographius-scope-2.herokuapp.com/
	IP Address: 52.211.26.224
	Report Time: 24 Jun 2020 12:30:09 UTC
	Headers: <div><div>✓ X-Frame-Options</div><div>✓ Strict-Transport-Security</div><div>✓ X-Content-Type-Options</div><div>✓ Referrer-Policy</div><div>✓ Feature-Policy</div><div>✓ Content-Security-Policy</div></div>
Warning: Grade capped at A, please see warnings below.	

Sl. 4.6 Rezultati skeniranja ranjive i sigurne verzije aplikacije koristeći *Security Headers*

Zaključak

Rezultati testiranja sigurne verzije jasno pokazuju da se implementiranjem navedenih metoda zaštite i najbolje prakse ostvarila iznimno visoka razina sigurnosti web aplikacije. S time je ostvarena i sigurnost u dubinu koja štiti da kompromitiranje jedne zaštite ne ugrožava cijeli sustav.

Testiranje je također ukazalo na neke od sigurnosnih problema *Angulara*. Politiku sigurnosti sadržaja nije bilo moguće do kraja izgraditi na siguran način zbog nesigurnog *Angularovog inline* stiliziranja. Uz to, *Angular* pretpostavljeno pokreće aplikaciju preko nesigurnog *HTTP* protokola. Trenutno je implementacija ovih nedostataka zadaća razvojnika, ali zbog sve većeg naglaska na potrebu sigurnosti, očekuje se da će u budućnosti ovi problemi biti riješeni u novijim verzijama okvira.

Ipak kroz rad se primijeti da *Angular* ima više sigurnosnih prednosti nego mana. Pruža automatsku sanitizaciju koja onemogućuje unos malicioznih skripti, automatski odstranjuje prefiksirane *JSON* odgovore, pruža klijentski dio zaštite protiv *CSRF* napada, nudi brojne validacijske funkcije kao i razrede za upravljanje greškama, pretpostavljeno koristi *offline* prevoditelj predložaka te sadrži brojne funkcionalnosti za ostvarenje dobre kontrole pristupa na klijentskoj strani.

Unatoč svim ovim pozitivnim stranama *Angularove* sigurnosti, one ipak nisu dovoljne za potpuno osiguravanje web aplikacije koja se sastoji od klijentske i poslužiteljske strane. Uz njih je potrebno implementirati i brojne metode sigurnosne prakse poput sigurnosnih zaglavlja, uporabe kolačića i odgovarajućih atributa te autentikacije i autorizacije. Osim toga, nužno je jednaku zaštitu ostvariti i na poslužitelju jer se tu nalaze svi podaci s kojima aplikacija upravlja te ih je iznimno važno zaštititi.

Angular se kao razvojni okvir pokazao izrazito praktičnim i efikasnim nudeći brojne funkcionalnosti i značajke za pisanje čitljivog, jednostavnog i održljivog kôda, kao i za općeniti razvoj web aplikacije, ali uzimajući u obzir navedene prednosti i mane, očito je da *Angular* pruža određenu razinu sigurnosti, ipak bez dodatnih implementacija, to nije nikako dovoljno za ostvarenje sigurne aplikacije.

Literatura

- [1] *Mrežna aplikacija*, Wikipedia, (2019, ožujak), Poveznica: https://hr.wikipedia.org/wiki/Mre%C5%BEna_aplikacija; pristupljeno 26. lipnja 2020.
- [2] *Single-page application*, Wikipedia, (2020, lipanj), Poveznica: https://en.wikipedia.org/wiki/Single-page_application; pristupljeno 26. lipnja 2020.
- [3] *2020 Developer Survey*, Stack Overflow, Poveznica: <https://insights.stackoverflow.com/survey/2020>; pristupljeno 26. lipnja 2020.
- [4] *Angular (web framework)*, Wikipedia, (2020, lipanj), Poveznica: [https://en.wikipedia.org/wiki/Angular_\(web_framework\)](https://en.wikipedia.org/wiki/Angular_(web_framework)); pristupljeno 26. lipnja 2020.
- [5] *TypeScript*, Wikipedia, (2020, lipanj), Poveznica: <https://en.wikipedia.org/wiki/TypeScript>; pristupljeno 26. lipnja 2020.
- [6] *TypeScript configuration*, Angular, Poveznica: <https://angular.io/guide/typescript-configuration>; pristupljeno 26. lipnja 2020.
- [7] Murray, N., Coury, F., Lerner, A. Taborda, C. *ng-book: The Complete Guide to Angular*, 5. izdanje, San Francisco: Fullstack.io, 2018.
- [8] *Structural directives*, Angular, Poveznica: <https://angular.io/guide/structural-directives>; pristupljeno 26. lipnja 2020.
- [9] *Attribute directives*, Angular, Poveznica: <https://angular.io/guide/attribute-directives>; pristupljeno 26. lipnja 2020.
- [10] *Template syntax*, Angular, Poveznica: <https://angular.io/guide/template-syntax>; pristupljeno 26. lipnja 2020.
- [11] *Introduction to services and dependency injection*, Angular, Poveznica: <https://angular.io/guide/architecture-services>; pristupljeno 26. lipnja 2020.
- [12] *In-app navigation*, Angular, Poveznica: <https://angular.io/start/start-routing>; pristupljeno 26. lipnja 2020.
- [13] *Introduction to forms in Angular*, Angular, Poveznica: <https://angular.io/guide/forms-overview>; pristupljeno 26. lipnja 2020.
- [14] *OWASP Top 10*, OWASP, Poveznica: <https://owasp.org/www-project-top-ten/>; pristupljeno 26. lipnja 2020.
- [15] Karande, C. *Securing Node Applications*, 1. izdanje, Sebastopol: O'Reilly Media, 2018.
- [16] *Command Injection*, OWASP, Poveznica: https://owasp.org/www-community/attacks/Command_Injection; pristupljeno 26. lipnja 2020.
- [17] *SQL Injection*, OWASP, Poveznica: https://owasp.org/www-community/attacks/SQL_Injection; pristupljeno 26. lipnja 2020.

- [18] *SQL Injection*, Phrack, (1998, prosinac), Poveznica: <http://phrack.org/issues/54/8.html>; pristupljeno 26. lipnja 2020.
- [19] *SQL Injection in report_xml.php through countryFilter[] parameter*, Hackerone, (2018, srpanj), Poveznica: <https://hackerone.com/reports/383127>; pristupljeno 26. lipnja 2020.
- [20] Centar informacijske sigurnosti, *Napadi umetanjem SQL koda*, 1. izdanje, Zagreb: Laboratorij za sustave i signale, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu, 2011.
- [21] *Form validation*, Angular, Poveznica: <https://angular.io/guide/form-validation>; pristupljeno 26. lipnja 2020.
- [22] *Credential stuffing*, Wikipedia, (2020, lipanj), Poveznica: https://en.wikipedia.org/wiki/Credential_stuffing; pristupljeno 26. lipnja 2020.
- [23] *Brute force attack*, Wikipedia, (2020, lipanj), Poveznica: https://en.wikipedia.org/wiki/Brute-force_attack; pristupljeno 26. lipnja 2020.
- [24] *Broken Access Control*, OWASP, Poveznica: https://owasp.org/www-community/Broken_Access_Control; pristupljeno 26. lipnja 2020.
- [25] *Router*, Angular, Poveznica: <https://angular.io/api/router>; pristupljeno 26. lipnja 2020.
- [26] *Microsoft Security Misconfiguration Exposed 250 Million Technical Support Accounts*, Redmond, (2020, siječanj), Poveznica: <https://redmondmag.com/articles/2020/01/22/microsoft-exposed-250m-support-accounts.aspx>; pristupljeno 26. lipnja 2020.
- [27] *Error Handler*, Angular, Poveznica: <https://angular.io/api/core/ErrorHandler>; pristupljeno 26. lipnja 2020.
- [28] *HttpInterceptor*, Angular, Poveznica: <https://angular.io/api/common/http/HttpInterceptor>; pristupljeno 26. lipnja 2020.
- [29] *Cross Site Scripting (XSS)*, OWASP, Poveznica: <https://owasp.org/www-community/attacks/xss/>; pristupljeno 26. lipnja 2020.
- [30] *Cross-Site Scripting (XSS) Makes Nearly 40% of All Cyber Attacks in 2019*, PreciseSecurity, (2019, prosinac), Poveznica: <https://www.precisecurity.com/articles/cross-site-scripting-xss-makes-nearly-40-of-all-cyber-attacks-in-2019/>; pristupljeno 26. lipnja 2020.
- [31] *Tik or Tok? Is TikTok secure enough?*, Check Point Research, (2020, siječanj), Poveznica: <https://research.checkpoint.com/2020/tik-or-tok-is-tiktok-secure-enough/>; pristupljeno 26. lipnja 2020.
- [32] *Mitigating Cross-site Scripting With HTTP-only Cookies*, Microsoft Developer Network, (2008, prosinac), Poveznica: [https://docs.microsoft.com/en-us/previous-versions/ms533046\(v=vs.85\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/ms533046(v=vs.85)?redirectedfrom=MSDN); pristupljeno 26. lipnja 2020.
- [33] *Security*, Angular, Poveznica: <https://angular.io/guide/security>; pristupljeno 26. lipnja 2020.
- [34] *Cross-Site Request Forgery (CSRF)*, OWASP, Poveznica: <https://owasp.org/www-community/attacks/csrf>; pristupljeno 26. lipnja 2020.

- [35] Burns, J. *Cross Site Request Forgery: An Introduction To A Common Web Weakness*, 2. izdanje, Information Security Partners, 2007.
- [36] *Security Advisory: CSRF & DNS Attacks*, DrayTek, Poveznica: <https://www.draytek.co.uk/support/security-advisories/kb-advisory-csrf-and-dns-dhcp-web-attacks>; pristupljeno 26. lipnja 2020.
- [37] *O socijalnom inženjeringu*, CERT, Poveznica: https://www.cert.hr/socijalni_inzenjering/; pristupljeno 29. lipnja 2020.
- [38] *CSRF prevention Cheat Sheet*, OWASP, Poveznica: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html; pristupljeno 26. lipnja 2020.
- [39] *HttpClientXsrfModule*, Angular, Poveznica: <https://angular.io/api/common/http/HttpClientXsrfModule>; pristupljeno 26. lipnja 2020.
- [40] *Ahead-of-time (AOT) compilation*, Angular, Poveznica: <https://angular.io/guide/aot-compiler>; pristupljeno 26. lipnja 2020.
- [41] *ElementRef*, Angular, Poveznica: <https://angular.io/api/core/ElementRef>; pristupljeno 26. lipnja 2020.
- [42] *Hypertext Transfer Protocol*, Wikipedia, (2020, lipanj), Poveznica: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol; pristupljeno 26. lipnja 2020.
- [43] *HTTPS*, Wikipedia, (2020, lipanj), Poveznica: <https://en.wikipedia.org/wiki/HTTPS>; pristupljeno 26. lipnja 2020.
- [44] *A milestone for Chrome security: marking HTTP as “not secure*, Google, (2018, srpanj), Poveznica: <https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/>; pristupljeno 26. lipnja 2020
- [45] *Serve*, Angular, Poveznica: <https://angular.io/cli/serve>; pristupljeno 26. lipnja 2020.
- [46] *Helmet*, Helmet, Poveznica: <https://helmetjs.github.io/>; pristupljeno 26. lipnja 2020.
- [47] *Content-Security-Policy*, Developer Mozilla, Poveznica: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>; pristupljeno 26. lipnja 2020.
- [48] *Template syntax*, Angular, Poveznica: <https://angular.io/guide/template-syntax>; pristupljeno 26. lipnja 2020.
- [49] *Npm-audit*, npm Documentation, Poveznica: <https://docs.npmjs.com/cli/audit>; pristupljeno 26. lipnja 2020.
- [50] *Getting started*, Zaproxy, Poveznica: <https://www.zaproxy.org/getting-started/>; pristupljeno 26. lipnja 2020.
- [51] *Checkbot*, Checkbot, (2019, ožujak), Poveznica: <https://chrome.google.com/webstore/detail/checkbot-seo-web-speed-se/dagohmlhagincbfilmkadjgmdnkjinl>; pristupljeno 26. lipnja 2020.
- [52] *Frequently Asked Questions*, Mozilla Observatory, Poveznica: <https://observatory.mozilla.org/faq/>; pristupljeno 26. lipnja 2020.

Sažetak

Sigurnost web aplikacija realiziranih programskim razvojnim okvirom Angular

Sažetak

U ovome radu istražena je sigurnost web aplikacija čiji je klijentski dio izgrađen razvojnim okvirom *Angular*. Uvodni dio dotiče se web aplikacija i razloga za njihovu sigurnost. Zatim su prikazane osnovne značajke *Angulara* te aplikacija koja je izgrađena kao praktični dio ovoga istraživanja. Ona je dizajnirana na što nesigurniji način te ne sadrži nikakvu dodanu zaštitu. U trećem poglavlju, opisane su ranjivosti i napadi na implementiranoj aplikaciji, kao i mjere zaštite za njihovu prevenciju. Nakon toga, prikazane su brojne implicitne i eksplicitne metode najbolje prakse pri radu s *Angularom* pomoću kojih se također postiže veća razina sigurnosti. Implementiranjem spomenute zaštite i metoda najbolje prakse, stvorila se sigurna verzija iste aplikacije. U petome poglavlju prikazano je paralelno sigurnosno testiranje ranjive i sigurne verzije aplikacije. Na kraju je su prikazane sigurnosne prednosti i mane razvojnog okvira *Angular* uz pripadajući zaključak.

Ključne riječi

Web aplikacija, *Angular*, *TypeScript*, *Express*, sigurnost, ranjivost, napad, zaštita, *XSS*, *CSRF*, testiranje, *HTTPS*, sigurnosna zaglavlja, politika sigurnosti sadržaja, najbolja praksa, kolačić, validacija, sanitizacija, autentikacija, autorizacija

Summary

Security of web applications implemented by the Angular software development framework

Summary

In this research, the security of web applications where the client part is built using Angular development framework, is described. Web applications and the reasons for their security are mentioned in the introductory part. Then, the basic features of Angular and the application that was built as a practical part of this research are presented. Application is designed in the most insecure way possible and does not contain any added protection. In the third chapter, vulnerabilities and attacks on the implemented application are described, as well as protection measures for their prevention. After that, a number of implicit and explicit methods of best practice in Angular are presented, which also achieve a higher level of safety. By implementing the mentioned protection and best practice methods, a secure version of the same application was created. Fifth chapter presents parallel security testing of a vulnerable and secure version of an application. Finally, the security advantages and disadvantages of the Angular framework are presented with the corresponding conclusion.

Keywords

Web application, *Angular*, *TypeScript*, *Express*, security, vulnerability, attack, protection, *XSS*, *CSRF*, testing, *HTTPS*, security headers, content security policy, best practice, cookie, validation, sanitization, authentication, authorization