Sigurnost mikroservisa

Košmrl, Simon

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:037856

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-17**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij informatike, smjer poslovna informatika

Simon Košmrl

Sigurnost mikroservisa

Diplomski rad

Mentor: v. pred. dr. sc. Vedran Miletić

Rijeka, 10. rujna 2021.

Sažetak

Arhitektura web aplikacija i usluga bazirana na mikroservisima u današnje vrijeme veoma je popularna. Mikroservisi su manje, zasebno odvojene aplikacije smještene u klusteru. Njihovo zajedničko djelovanje čini cjelovitu aplikaciju. Kali Linux je najpopularnija distribucija Linuxa za testiranje sigurnosti web aplikacija. Kako bi korisnicima web aplikacije pružili najbolju moguću uslugu, potrebno je testirati sigurnost sustava. Ciljevi informacijske sigurnosti su steći povjernjivost, integritet i konstantno biti u mogučnosti pružati uslugu. U radu se bavimo metodama ispitivanja i poboljšanja sigurnosti mikroservisa.

Ključne riječi

mikroservisi, web aplikacije, softver, napadi na web aplikacije, testiranje sigurnosti

Sadržaj

1. Uvod

Proteklih 20 godina pristup Internetu postao je dostupan skoro u svakom kućanstvu i na svakom pametnom telefonu. Razvili su se mnogi uređaji koji podržavaju komunikaciju putem interneta te se kao posljedica toga stvorio internet stvari (*Internet of Things*). Razmjena podataka i informacija u realnom vremenu promijenila je našu svakodnevicu. Mnoge korporacije prepoznale su prilike i mogućnosti koje nudi ovakva tehnologija te su se time razvile i novi načini poslovanja. Ideja o kupovini putem interneta pridonijela je nove načine trgovanja te se stoga stekla potreba za stvaranjem sigurnog interneta i pouzdanih web aplikacija.

Već u ranim danima razvoja interneta kakvog poznajemo danas, pojedinci su iskorištavali ovu

tehnologiju kako bi došli do informacija i podataka koje inače ne bi smjeli posjedovati. Takve pojedince nazvali smo hakerima. Hakeri su osobe koje dobro poznaju infomacijske sustave, kao i tehnologije kojima se koriste. Oni su računalni eksperti koji svoje znanje o hardveru i softveru koriste kako bi došli do cilja ili kako bi zaobišli prepreku unutar računalnog sustava pritom koristeći se neuobičajenim mjerama. Ovisno o njihovim ciljevima, razpoznajemo 3 vrste hakera. Hakeri sa bijelim šeširom (*White hat hackers*) svoje vještine upotrebljavaju kako bi podatke zaštitili od drugih vrsta hakera te kako bi unaprjedili sigurnost računala i informacijskih sustava. Ciljevi hakera crnih šešira (*Black hat hackers*) su zlonamjerni. Često kradu i prodaju informacije koje su stečene na ilegalne načine. Hakeri sivih šešira (*Gray hat hackers*) hakiraju iz zabave, njihova dijela također su često ilegalna, no njihovi motivi nisu financijski orjentirani.

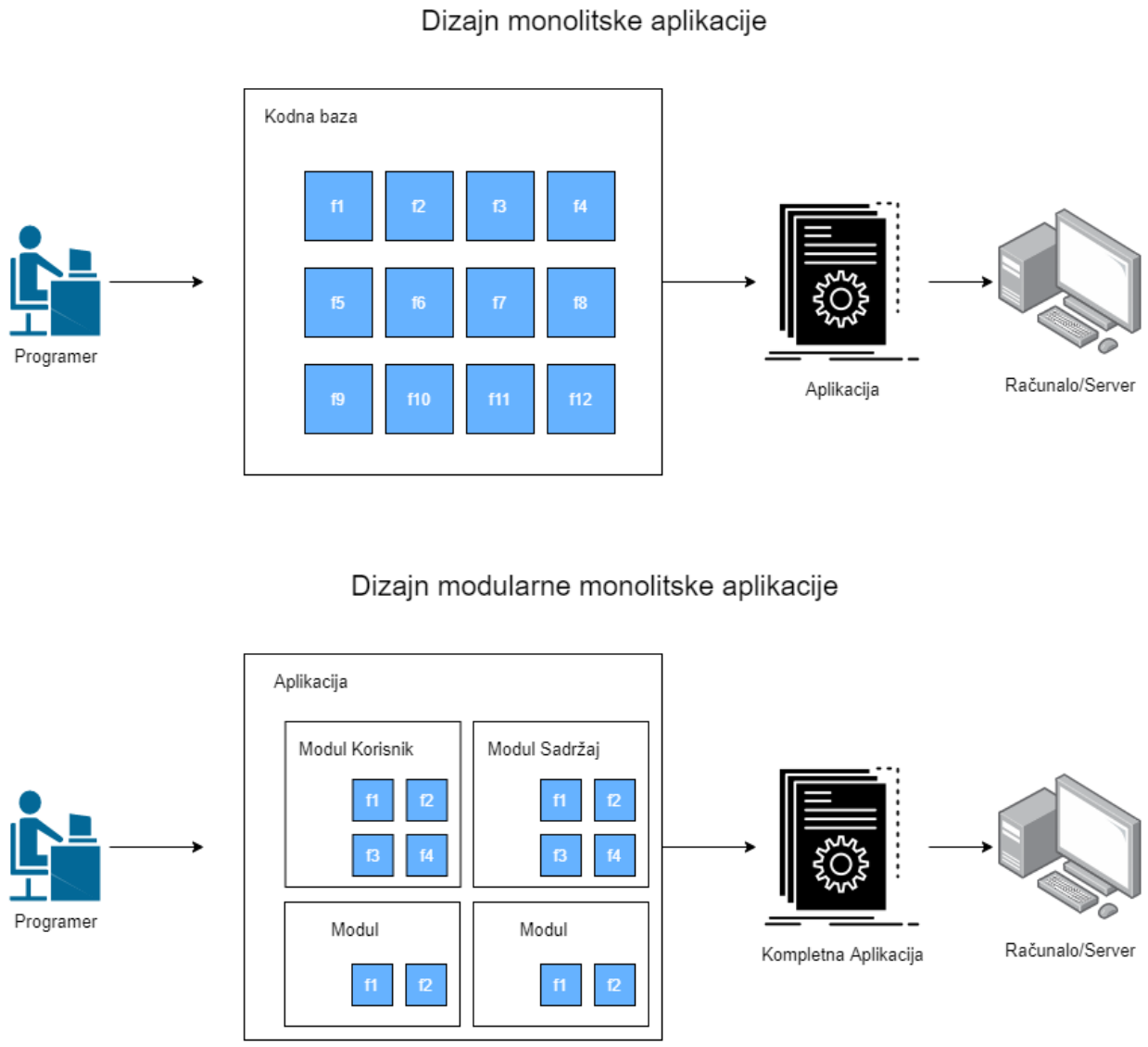
U ovome diplomskom radu istražene su i opisane prijetnje kojima se susreću aplikacije na webu. Kao praktični dio diplomskog rada, izrađen je i opisan proces izrade jednostavne web aplikacije bazirane na mikroservisima. Razvijena aplikacija sadrži osnovne funkcionlanosti koje posjeduje većina društvenih platformi, a to su prijava, registracija i objavljivanje sadržaja. Nakon izrade, opisani su procesi postavljanja aplikacije na web server kako bi ta aplikacija postala dostupna javnosti. Također opisani su alati i metode kojima se koriste hakeri kako bi izvršili napade na takve aplikacije. Tim metodama i alatima ispitana je sigurnost razvijene aplikacije. Kako bi ova aplikacija postala sigurnija, predložene su mjere sigurnosti za svaki od testiranih napada.

2. Mikroservisi

Kako bi shvatili smisao mikroservisa, potrebno je prvo opisati probleme kojima su se programeri zatekli prilikom korištenja tradicionalnog načina izrade web aplikacija. Arhitektura aplikacija prije pojave mikroservisa bila je monolitskog tipa (grč. *monolithos*: *mono* – jedan + *lit* – kamen). Takva vrsta arhitekture obilježava implementaciju web aplikacija kao jedne cjeline i uveliko se koristila proteklih 20 godina.

Početkom razvoja monolitskih web aplikacija pisani kod bio bi pohranjen u jednu kodnu bazu, te kad bi aplikacija zahtijevala novu funkcionalnost taj kod bi se gomilao na jednom mjestu. Ovakav pristup razvoja softvera nije povoljan za ponovno iskorištavanje napisanog koda. Da bi se funkcionalnosti jedne aplikacije ponovno iskoristile u nekoj drugoj, kod bi se iz jedne kodne baze kopirao ili ponovno napisao u drugoj. Ovakav pristup programiranju otežava održavanje aplikacija iz razloga što bi se svaka dodatna promjena funkcionalnosti trebala promijeniti u svim kodnim bazama koje je sadržavaju. Kao posljedica toga, programeri su našli način kako iskoristiti jednu kodnu bazu određenih funkcionalnosti unutar više projekata, odnosno aplikacija. Takve kodne baze nazivaju se modulima.

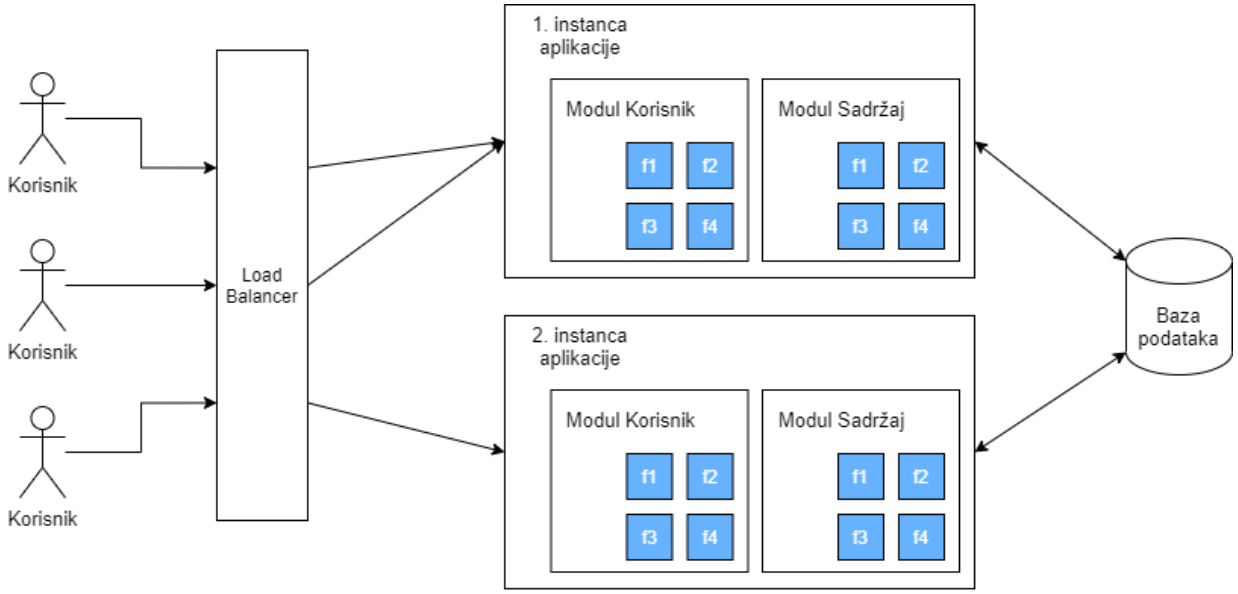
Modularno programiranje je arhitektura softvera koja naglašava razdvajanje pojedinih skupina funkcionalnosti na neovisne module za koji svaki od modula ima definirani aspekt funkcionalnosti koje podržava. Kao primjer možemo uzeti module „Korisnika” i „Sadržaja”. Modul korisnika zadužen je za funkcionalnosti prijave i registracije korisnika, dok je modul sadržaja zadužen za dohvaćanje i objavu sadržaja. Aplikacije izrađene na modulima lakše su za održavanje te se svaki od modula može dodijeliti razvojnom timu koji je zadužen za njega. Na ovakav način precizno su specificirane odgovornosti pojedinih timova što dodano olakšava održavanje modula i aplikacija. Važno je za shvatiti da je konačna verzija softvera baziranog na modulima i dalje monolitskog tipa, neovisno o tome što se kodne baze modula i aplikacije nalaze na različitim mjestima. Prilikom implementacije softvera na računalo ili web, svi moduli spojeni su u jednu cjelinu i ponašaju se kao takvi. Slika 1. vizualno prikazuje arhitekturu monolitske i modularno monolitske aplikacije.



Slika 1: Vizualni prikaz strukture monolitskih i modularno monolitskih aplikacija

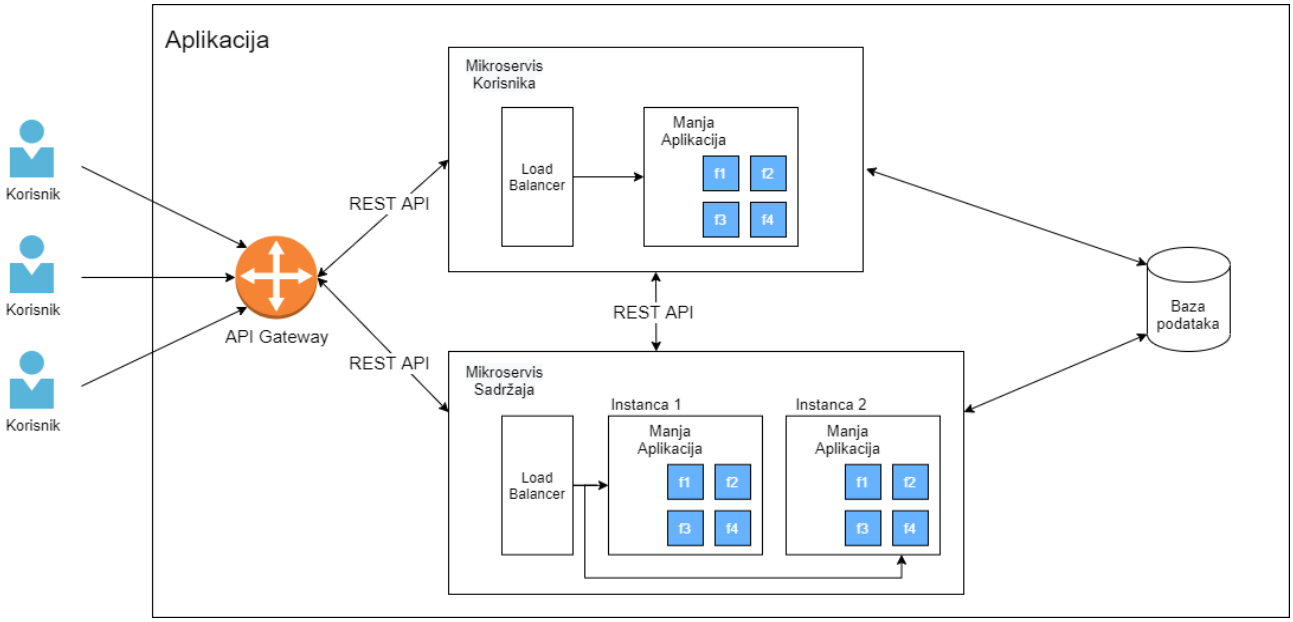
Ideja o mikroservisima započela je zbog problema kojima su se monolitske aplikacije susrele u produkcijskom okruženju. U početku razvoja interneta, web aplikacije pokretale su se i izvršavale na jednom serveru. Povećanje broja korisnika web aplikacije zahtijevalo je i povećanje resursa - procesora, radne memorije i spremišta (vertikalno skaliranje). Svi korisnici pristupaju istom serveru i koriste iste resurse. Kada bi se iskoristili svi dostupni resursi jednog servera, web aplikacija uopće ne bi bila dostupna nekim korisnicima, a drugim korisnicima rad aplikacije bi se drastično usporio. Programeri su rješenje dostupnosti resursa pronašli u horizontalnom skaliranju web aplikacija, tj. korištenjem nečeg što se naziva balanser opterećenja (*load balancer*). *Load balancer* naziv je za softver ili hardver čija zadaća je raspodjela zadatka na određeni skup resursa. Imitiranjem ponašanja obrnutog proxy-a, promet web aplikacija distribuiran je na više instanci aplikacija te kao posljedica toga aplikacija podržava puno veći mrežni promet. Instance aplikacija mogu se pokretati na više servera, virtualnih mašina ili danas popularnim – *Docker* kontejnerima koji se pokreću kao servis na operacijskom sustavu. Primjer horizontalnog skaliranja modularno monolitske web aplikacije

predstavljen je na Slici 2. Prikazane su dvije instance jedne aplikacije koje su pokrenute na različitim serverima te koriste istu bazu podataka. Raspodjelom prometa upravlja *load balancer* kojeg pozivaju korisnici. Ovisno o dostupnosti resursa spajaju se na određenu instancu aplikacije.



Slika 2: Horizontalno skaliranje monolitske aplikacija

Broj instanci aplikacija postavlja se po potrebi, ovisno o prometu aplikacije i dostupnosti računalne snage. Skaliranje monolitske web aplikacija ima jednu veliku manu. Neovisno o razlogu zbog čega je potrebno skaliranje aplikacije, novostvorene instance sadržavaju sve module koje aplikacija zahtjeva za normalan rad. Ako naša aplikacija stekne veći broj korisnika, te ti korisnici uveliko objavljuju i pregledavaju različite sadržaje, modul Sadržaja tada će imati puno veći promet i zahtijevati će više resursa. Kao posljedica toga aplikaciju ćemo skalirati. Stvoriti ćemo novu instancu aplikacije koja će uz modul Sadržaja imati i modul Korisnika kojeg nije potrebno skalirati jer nema povećani promet. Kao rješenje tog problema nastao je koncept mikroservisa. Dizajn aplikacije mikroservisa bazira se na odvajanju modula u zasebne servise. Ti servisi se pokreću na različitim serverima te su zajedno spojeni na lokalnoj mreži. Aplikacija tada postaje skup servisa koji međusobno mogu komunicirati. Mikroservisi su manje aplikacije čiji skup predstavlja cjelovitu aplikaciju. Pristup funkcionalnostima mikroservisa prolazi kroz glavnu točku protoka – *API gateway*, koji preusmjerava pozive korisnika prema različitim servisima, ovisno o radnjama koje su potrebne. Slika 3 vizualno prikazuje dizajn aplikacije baziranih na mikroservisima.



Slika 3: Dizajn aplikacije mikroservisa

2.1. Prednosti i nedostaci mikroservisa

Jedna od najvećih prednosti mikroservisa je neovisnost tehnologije na kojoj se izvršavaju. Kako je komunikacija između servisa bazirana na *API* pozivima, nismo ograničeni na jednu tehnologiju za izradu tih mikroservisa. Kada bi na našem primjeru aplikacije željeli dodati servis koji predviđa relevantne sadržaje koje zanimaju pojedinog korisnika, za razvoj nismo nužni koristiti samo JavaScript jezik. Za realizaciju takvog servisa vjerojatno bi koristili Python, iz razloga što posjeduje bolje biblioteke za izgradnju umjetne inteligencije neuronskih mreža, na čemu su mnogi takvi servisi i bazirani. Sukladno tome, svaki od mikroservisa može posjedovati svoju bazu podataka. Time se još više naglašava neovisnost između mikroservisa. Jedan servis može imati relacijsku bazu podatak, dok drugi mikroservisi mogu imati nerelacijske baze podataka. Druga prednost mikroservisa je mogućnost skaliranja servisa koji zahtjeva više resursa i ima povećani aplikacijski promet. Ostale servise koji tada ne ovise o povećanom prometu nije potrebno skalirati. Također, mikroservise je lakše za održavati. Unutar organizacije, zaduženje za jedan mikroservis može se dodijeliti jednom razvojnom timu ili pojedincu. Isporuka novih funkcionalnosti jednog mikroservisa na produkcijsku okolinu ne ovisi o ostatku aplikacije jer je svaki od servisa pokrenut u zasebnoj okolini. Time se olakšava i ažuriranje aplikacije. Prilikom ažuriranja ponovno pokrećemo samo servise koji zahtijevaju promjene, za razliku od monolitske aplikacija u kojima se ponovno pokreće cijela aplikacija. Dodavanje novih servisa također nije problem. Implementacijom novih mikroservisa ne utječemo na dosadašnji rad i razvoj aplikacije. Razvoj novih mikroservisa odvojen je od ostalih. Najveća prednost mikroservisa je eliminacija totalnog pada sustava. Korisnicima je omogućena usluga, iako nešto pođe po zlu. U slučaju da se dogodi kvar na jednom od servisa koji

nije dio glavnih procesa aplikacije, tada će sustav nastaviti svoj rad. Za razliku od monolitske aplikacije koja bi se u takvom slučaju kompletno zaustavila.

Arhitektura mikroservisa je poprilično kompleksna, što je jedan od najvećih nedostataka. Takva arhitektura aplikacija zahtjeva više planiranja i nadzora nad svakim od servisa. Često je potrebna optimizacija aplikacije kako bi se pružila najbolja korisnička usluga. Kako mikroservisi međusobno komuniciraju putem API poziva, može doći do usporenja veze i sporog učitavanja podataka. U tom slučaju dolazi do lošeg korisničkog iskustva. Mikroservisi zahtijevaju dobro definirane pozive i odgovore kako bi integracija sa drugim servisima bila jednostavnija. Dobro definirani podaci i pozivi te vođenje liste zavisnosti mikroservisa jednih o drugima olakšava testiranje i integraciju novih promjena svakog mikroservisa.

Jedan od standarda mikroservisa je da svaki od njih posjeduje i komunicira sa njemu određenom bazom podataka. Kada bi svi servisi komunicirali sa jednom bazom podataka te kada bi ta baza pala, svi mikroservisi prestali bi raditi. Time bi baza podataka postala kritična točka pada cijele aplikacije, što koncept mikroservisa želi izbjeći. Razdvajanjem baza podataka često se stječe potreba za dohvaćanjem podataka iz više bazi, te se time može stvoriti ovisnost između mikroservisa. Postoje 2 načina komunikacije između mikroservisa. Servisi mogu međusobno komunicirati direktno, putem API poziva. Taj način nije pogodan jer se stvara ovisnost između servisa. Ako jedan od mikroservisa padne, drugi servisi koji ovise o njemu neće moći pristupiti tim podacima. Drugi način komunikacije između mikroservisa, koji se više koristi, je putem događaja (*event hub*). Ovaj način komunikacije stvara duplicirane podatke. Podaci koji su potrebni jednom servisu da radi, neovisno o drugim servisima, spremljeni su u njegovoj bazi i u bazi koja ih je primarno spremila. Često jedan mikroservis treba podatke iz više baza drugih mikroservisa. Kako bi smanjili kritične točke pada poziva, možemo stvoriti novi mikroservis koji komunicira sa svim ostalima servisima putem događaja. Taj mikroservis, zadužen je za spremanje svih podnesenih radnji događaja. Time se svi ostali mikroservisi, koji su u nekom trenutku pali, mogu sinkronizirati kako bi posjedovali najnovije podatke. Postoje mnoge prednosti i mane, no mnogi smatraju kako prednosti nadmašuju sve nedostatke mikroservisa.

3. Izrada web aplikacije

Kao praktični dio diplomskog rada izrađena je jednostavna web aplikacija koja podržava funkcionalnosti prijave i registracije korisnika te funkcionalnosti objavljivanja i pregledavanja objavljenog sadržaja. Prilikom izrade web aplikacije, korištene su sljedeće tehnologije: Angular, NodeJS, Express, MongoDB. Ovaj skup tehnologija poznat je još pod imenom *MEAN stack* (Mongo, Express, Angular i NodeJS). Korištenjem ovih tehnologija obuhvaća se cjelokupni razvoj potreban za izradu web aplikacija (*full-stack development*).

3.1. Opis korištenih tehnologija

Za razvoj vizualnog dijela aplikacije (*front-end development*) poslužio je Angular u kombinaciji sa Bootstrapom. Angular je programska biblioteka i razvojna platforma za izradu učinkovitih i sofisticiranih web i mobilnih aplikacija temeljenih na jednoj web stranici. Platforma je besplatna i otvorenog izora te je pod vodstvom Googleovog tima i zajednice zainteresiranih pojedinaca. Za izgled elemenata stranice korišten je Bootstrap, najpoznatiji alat za stiliziranje web stranica.

Za razvoj funkcionalnosti i pozive na mikroservise (*backend development*) korišteni su NodeJS/Express te MongoDB kao baza podataka. MongoDB je nerelacijska baza podataka u koju se podaci ne pohranjuju u tablice već u dokumente. Jedna od prednosti nerelacijskih baza u odnosu na relacijske baze je učinkovitost pohrane velike količine podataka, što je čini pogodnom za pohranu sadržaja ove aplikacije. NodeJS je *runtime enviorment* otvorenog izvora baziran na JavaScriptu. Pokreće ga V8 JavaScript engine što mu omogućuje da bude jako učinkovit i da se pokreče na više platformi. Izvršava se na jednoj jezgri u jednom procesu, a pomoću petlje događaja, koje omogućuje JavaScript, kod se prilikom svog izvršavanja ne blokira. Za razvoj API poziva korišten je Express. Express je fleksibilna i minimalistička biblioteka za NodeJS koja pruža mnoge funkcionalnosti za stvaranje robusnih API poziva koje koriste skoro sve aplikacije na webu.

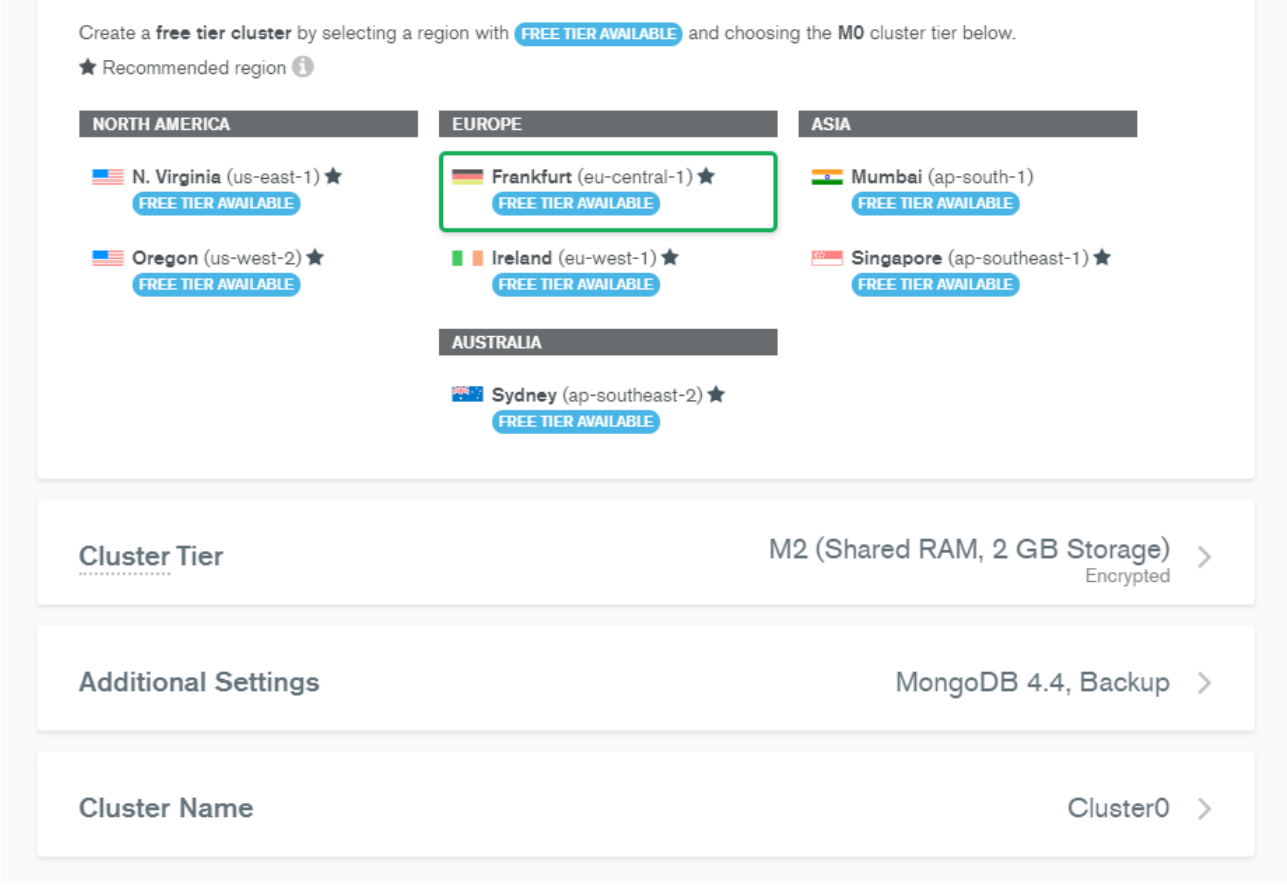
3.2. Konfiguracija baze podataka – MongoDB Atlas

MongoDB Atlas je internet servis koji nudi MongoDB baze u oblaku. Do nekih granica potrošnji usluge servisa su besplatne. Besplatno korištenje servisa obuhvaća stvaranje do 100 različitih baza podataka, uz limitirano spremište od 5GB. Razlog zbog kojeg je odabran ovaj servis su mogućnosti koje se nude plaćanjem usluga Atlasa. Jedna tih usluga je stvaranje sigurnosnih kopija baza podataka, što je jedna od bitnih stavki sigurnosti baza podataka. Atlas koristi TLS/SSL

kriptografske protokole za sigurno spajanje na bazu. Pristup bazi određen je na dva načina – postavljanjem IP adrese koja može pristupiti bazi, ili postavljanje virtualne privatne mreže unutar koje se omogućava pristup. Prijavom u nadzornu ploču Atlasa, olakšano je stvaranje novih korisnika baze kojima se dodjeljuje pristup. Pri stvaranju novih korisnika moguće je ograničiti pojedine radnje unutar baze podataka kao što su pisanje, čitanje i brisanje. Unutar nadzorne ploče postoji alat za pregled povijesti spajanja na bazu. U tom alatu moguće je raspoznati sve uspješne i neuspješne pokušaje pristupa. Na taj način saznajemo sve nedozvoljene pristupe bazi te sukladno tome, ukoliko je potrebno, možemo reagirati sa novim mjerama sigurnosti.

3.2.1. Izrada klastera

U ovome radu korištena je besplatna verzija servisa. Klaster baze podataka smješten je u Frankfurtu i korištene su zadane postavke koje se ne plaćaju (Slika 4).

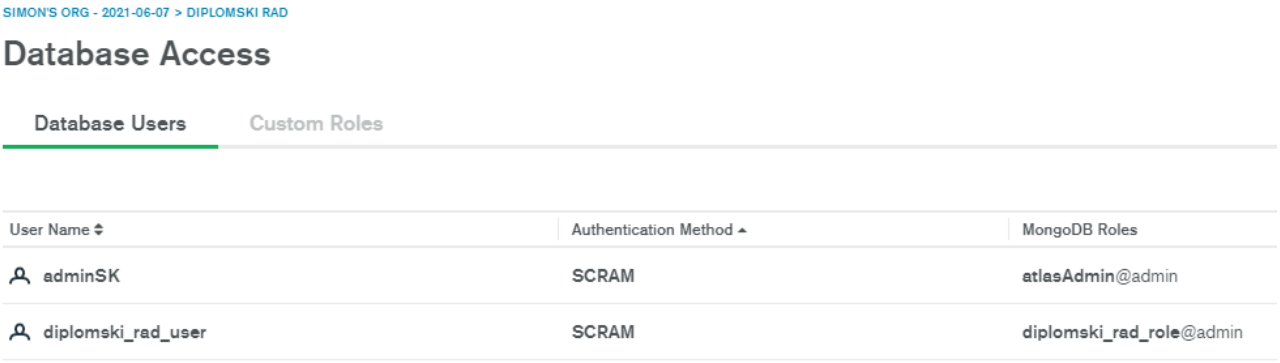


Slika 4: Postavke klastera

3.2.2. Postavljanje pristupa bazi

Baza koja će se koristiti kao spremište podataka nazvana je „diplomski_rad” te je sukladno tome stvoren novi korisnik „diplomski_rad_user”. Korisniku je dodijeljena prilagođena uloga koja mu

daje dopuštenje čitanja, pisanja i brisanja podataka iz te baze podataka. Razvijena aplikacija koristiti će ovog korisnika za identifikaciju i autorizaciju pristupa bazi.



Slika 5: Lista korisnika baze podataka

Pristup bazi omogućen je cijelom internetu. Postavljanje pristupa na IP adresu *0.0.0.0/0*, bazi je omogućen pristup sa bilo koje IP adrese. Takvom konfiguracijom ugrožava se sigurnost baze, no zbog olakšanja izrade ovog diplomskog rada ta opcija je omogućena. Kada bi baza podataka bila namijenjena za produkcijsko okruženje, ta opcija ne bi bila uključena. U produkcijskom okruženju pristup bazi ograničio bi se korištenjem *VPC (Virtual Private Cloud) peering-a* ili postavljanjem privatnih pristupačnih točki *(Private endpoint)*. Korištenjem tih metoda, spajanje bazi dozvoljeno je se samo unutar definiranih privatnih mreža što poboljšava sigurnost baze.

Slika 6: Mrežne postavke baze podataka - lista dozvoljenih IP adresa

3.3. Opis izrađenih mikroservisa

Pozadinski dio aplikacije razvijen je u tri dijela, odnosno tri mikroservisa: *Gateway, Users, Posts*. Mikroservis *Gateway,* primarni je servis kroz kojeg prolazi cijeli promet aplikacije. Ukoliko je

potrebno izvršiti neku od funkcionalnosti drugih mikroservisa, *Gateway* preusmjerava poziv korisnika prema mikroservisu koji podržava traženu funkcionalnost. Mogućnost preusmjeravanja poziva omogućena je pomoću biblioteke „express-http-proxy”. Prema određenoj konfiguraciji, *Gateway* prepoznaje putanje do svih mikroservisa te korištenjem obrnutog proxy-a preusmjerava API pozive prema traženim servisima.

```
const SERVICES = {
    "users": {
        proxy: httpProxy("http://srv-users:3001"),
        protected: false,
    },
    "posts": {
        proxy: httpProxy("http://srv-posts:3002"),
        protected: true
    },
}
```

Pozivi prema mikroservisima prolaze kroz rutu „/api”. Sustav naknadno iz URL-a saznaje kojem servisu se želi pristupiti. Ukoliko se u konfiguraciji nalazi traženi servis provjeravamo dali je za pristup potrebna autorizacija. Ovakvim pristupom sa lakoćom možemo kontrolirati dali neki korisnik ima pravo pristupiti operacijama mikroservisa ili ne. Ukoliko je pristup servisu dozvoljen, poziv korisnika je preusmjeren prema traženom servisu.

```
// API GATE WAY
this.app.use("/api", async (req, res, next)=> {
       ...

const paths = reqUrl.split("/"); // [ "", *IME SERVISA (eg. "posts
")*, *POZIV FUNKCIJE* ]
       ...
       // provjera ako je servis registriran
       const service = SERVICES[paths[1]];
       if(!service)
           return res.sendStatus(404);
       // validacija pristupa
       if(service && service.protected) {
           try {
               const token = req.headers.authorization;
               if(!
token) return res.status(401).send('Access Denied');
               const data: any = jwt.verify(token, JWT_SECRET);
               ...
               }
           }
         ...
       }
       service.proxy(req, res, next); // Preusmjeravanje poziva
});
```

Osim preusmjeravanja poziva i kontrole autorizacije, *Gateway* također poslužuje web stranicu na glavnoj ruti aplikacije. NodeJS i Express nude opciju posluživanja statičkih datoteka unutar aplikacije. Ova opcija iskorištena je umjesto standardnog posluživanja web stranica korištenjem Apache ili Nginx poslužitelja.

```
// posuživanje angular aplikacije
// posluživanje svih datoteka web stranice
this.app.use(express.static(path.join(__dirname, '../public')));
// angular radi na jednoj stranici – svi pozivi se upućuju na
index.html
this.app.get('/*', async (req, res) => {
    res.sendFile(path.resolve(__dirname, '../
public', 'index.html'));
});
```

Gateway aplikacija, kao i ostali mikroservisi, napisani su pomoću TypeScripta. TypeScript je sintaktički pristup pisanja koda, on je super-skup JavaScripta. Koristi se za u razvoju velikih aplikacija. Napisani kod kompajlira se u JavaScript.

```
    constructor(){
        this.app = express();
    }

    async init(){
        const port = 3000;
        this.setup();
        this.route();

        this.app.listen(port, () => {
            console.log(`App running on port: ${port}`)
        });
    }

    setup() {
        this.app.use(express.json());
        this.app.use(express.urlencoded());
        this.app.use(cors());
    }
```

Mikroservis *Gateway* pokrenut je na portu 3000. Express aplikacije svih mikroservisa koriste standardne postavke parsiranja podataka zahtjeva i postavljanja header-a poziva.

Mikroservis *Users* pokrenut je na portu 3001. Zadužen je za prijavu i registraciju korisnika. Bitna stavka sigurnosti prilikom registracije korisnika je da se unesena šifra korisnika *hashira*. Ukoliko netko upadne u bazu podataka, neće moći prepoznati lozinke korisnika. *Hashirana* šifra može se usporediti prilikom prijave, na taj naći znamo dali je upisana šifra jednaka onoj unesenoj prilikom registracije. U ovoj aplikaciji korištena je *bycript* biblioteka koja nudi razne algoritme *hashiranja*.

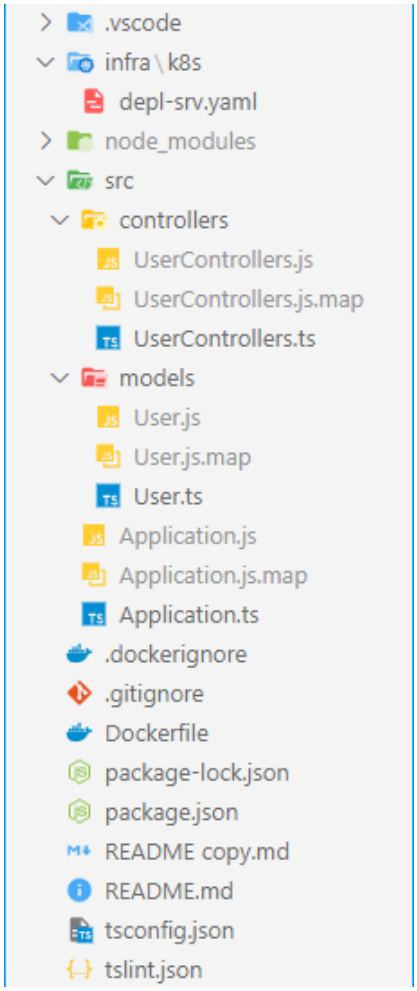
```
async hashPassword(){
        this.password = await bcrypt.hash(this.password, 10);
        await this.save();
}
```

Prilikom prijave korisnika generira se JSON Web Token (JWT). Json Web Tokeni industrijski su standard (rfc7519) za sigurno predstavljanje korisnika na serveru. Podaci zapisani u tokenu mogu se provjeriti i može im se vjerovati jer su digitalno potpisani. JWT podržava mnoge algoritme digitalnih potpisa kao što su HMAC, RSA ili ECDSA. Prilikom prijave na ovu aplikaciju, generira se token koji u sebi sadrži informacije o ID-u korisnika i njegovom korisničkom imenu. Ukoliko je

uspješna prijava token se šalje korisniku. Spremanjem tokena u lokalno spremište klijenta izbjegava se potreba za prijavu prilikom svakog novog pristupa stranici.

```
const token = jwt.sign(
                { identification: user._id, username: user.usernam
e },
                UserController.JWT_SECRET, options);
```

Mikroservis *Posts* pokreće se na portu 3002. Za pristup ovom servisu potreban je *Authorization* header sa validnim JWT tokenom. Provjera tokena izvršava se u mikroservisu *Gateway. Posts* servis zadužen je za pohranu i za dohvaćanje spremljenih postova. Mikroservisi *Users* i *Posts*, za razliku od *Gateway-a*, imaju pristup bazi i njihova struktura projekta prati MVC (Model, View, Controller) strukturu raspodjele direktorija. Slijedeća slika prikazuje cijeloukupnu strukturu projekta mikroservisa *Users*. Jednaku strukturu projekta koristi i mikroservis *Posts*.

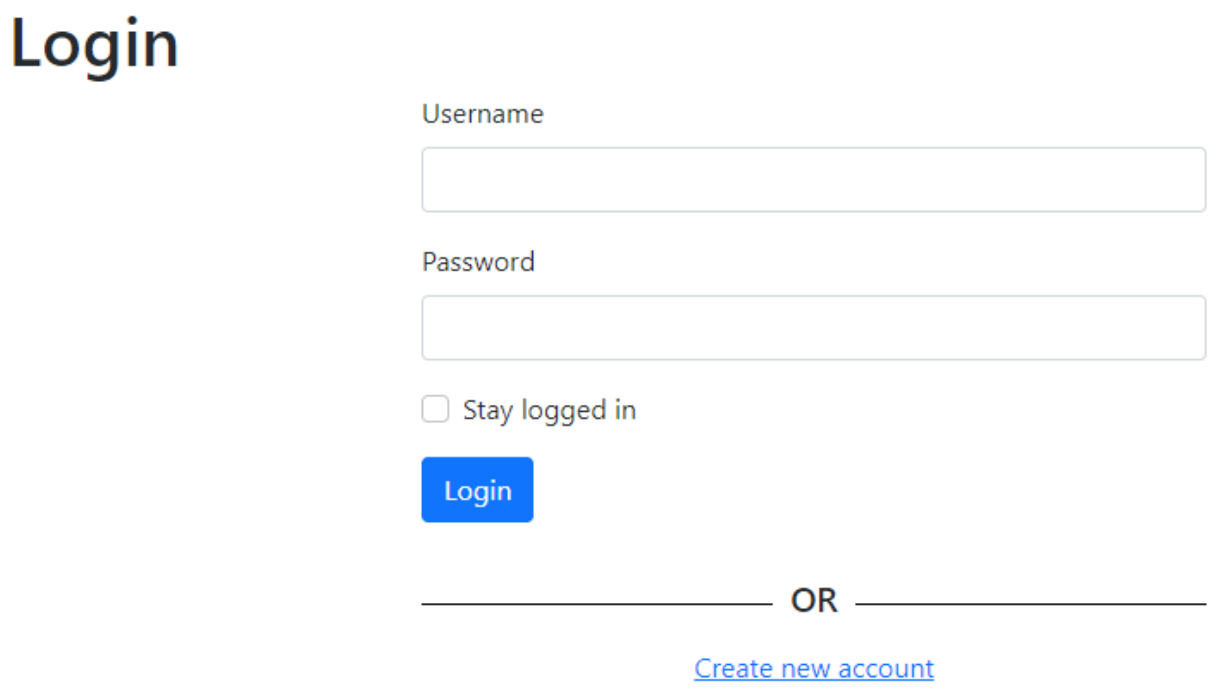


Slika 7: Struktura projekta mikroservisa Users

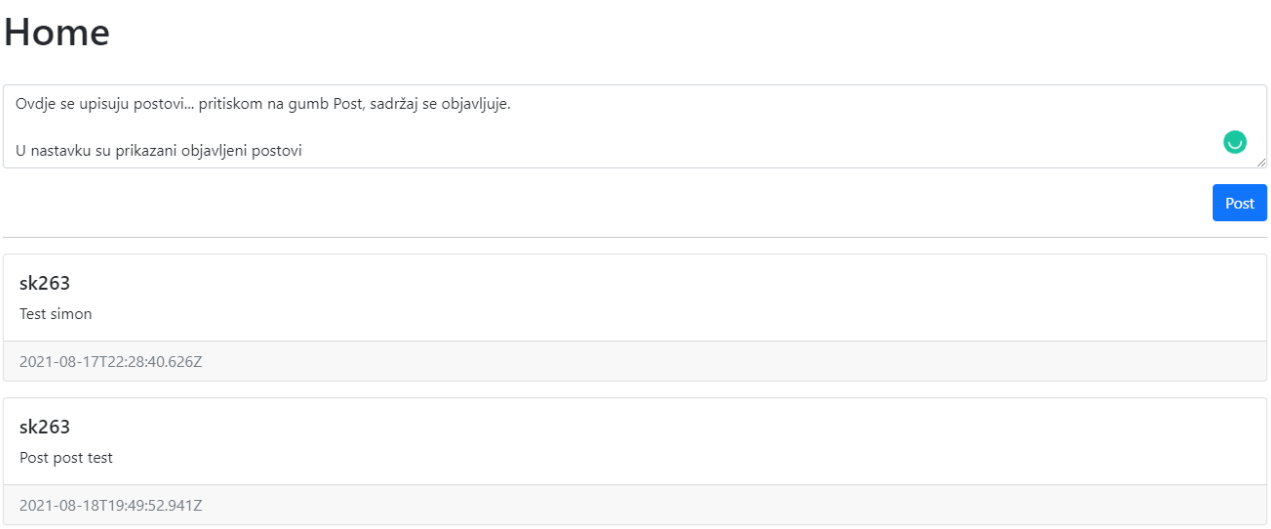
3.4. Korisničko sučelje

Korisničko sučelje aplikacije napravljeno je u Angularu. Sastoji se od 3 stranice: *Login*, *Registration* i *Main.* Pristup stranici *Main* (Slika 9), omogućen je samo prilikom uspješne prijave i validnog JWT tokena. Ovu funkcionalnost omogućuje nam Angular *CanActivate guard.* Ukoliko korisnik nije prijavljen i nema validan token, *guard* preusmjerava korisnika na *Login* stranicu (Slika 8). Na stranici *Main* prikazuju se i objavljuju postovi.

Izrađena web stranica poslužuje se na glavnoj ruti *Gateway* mikroservisa.



Slika 8: Prikaz stranice Login

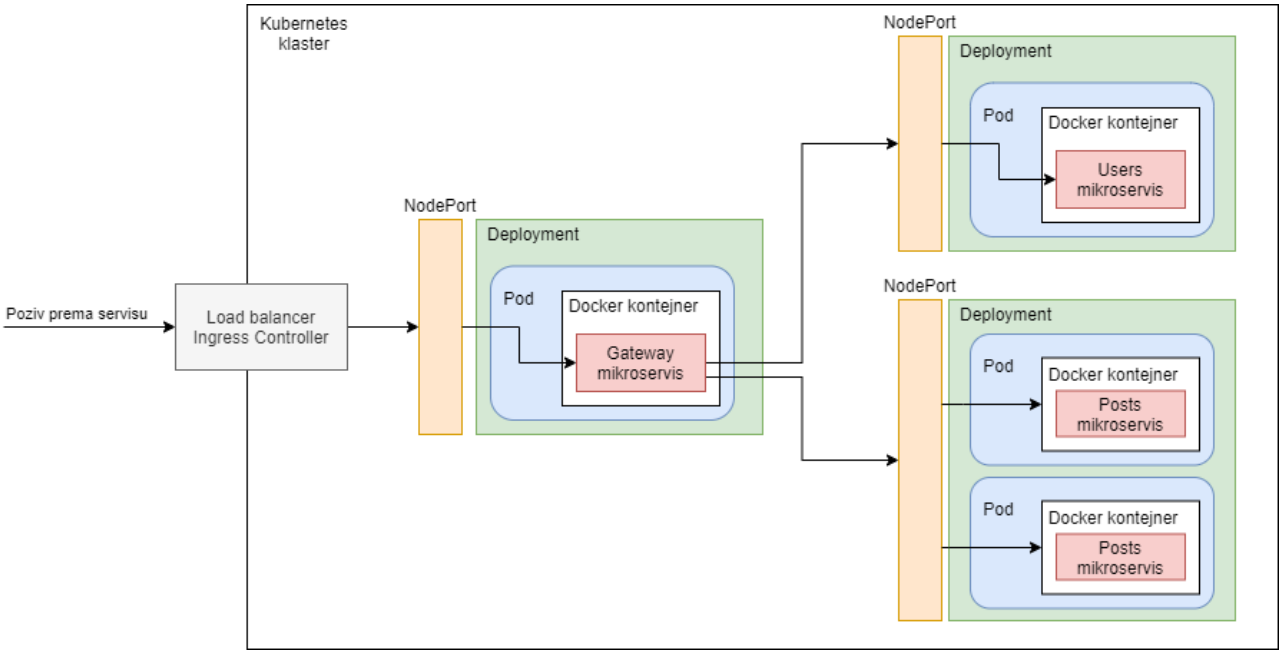


Slika 9: Prikaz stranice Main

4. Postavljanje mikroservisa

Najbolja praksa pokretanja mikroservisa je da se svaki od servisa pokreče u jedinstvenoj okolini. Jedna od bitnih karakteristika mikrosevisa je njihov brzi razvoj i brza implementacija na produkcijsku okolinu. Korištenjem Dockera upravo to i postižemo. Docker je otvorena platforma za razvoj, distribuciju i pokretanje aplikacija. On omogućava izvršavanje aplikacija na bilo kojoj platformi koja podržava Docker. Aplikacije se pokreću u virtualnim okruženjima koje nazivamo kontejnerima (eng. *Containers*). Kako bismo postavili (eng. *deploy*) aplikaciju u kontejner, prvo moramo napraviti Docker sliku (eng. *Image*). Docker slika sadrži sve informacije, konfiguracije i podatke koji su potrebni da bi naša aplikacija radila unutar kontejnera.

Prilikom horizontalnog skaliranja pojedinog mikroservisa imati ćemo više kontejnera koji pokreću taj servis. Da bi korisnik znao na koju instancu aplikacije se može spojiti, koristiti ćemo Kubernetes. Kubernetes, poznat po imenu K8S, je sustav otvorenog izvora koji služi za deployanje, skaliranje i kontrolu kontejneriziranih aplikacija. Primarna zadaću mu je orkestracija kontejnera, što ga čini povoljnim za upravljanje mikroservisima. Cilj nam je pokrenuti jedan mikroservis po Docker kontejeru, a služeći se alatima Kubernetesa olakšano nam je skaliranje i upravljanje tih kontejnera. Slika 10. vizualno prikazuje arhitekturu aplikacije unutar Kubernetes klastera. Pristup servisima unutar klastera omogućava *Ingress Controller.* Njegova zadaća je dozvoliti promet koji dolazi izvan klastera i po potrebi pozvati odgovarajuće servise unutar klastera. Pristup svim mikroservisima moguć je samo unutar samog klastera, zbog toga je potrebno definirati *Ingress Controller,* koji je zapravo *Load Balancer,* kako bi se omogućili pozivi na servise i izvan klastera. U primjeru ove aplikacije, *Ingress Controller* preusmjerava promet samo prema jednom servisu – *Gateway-u.* Na slici možemo primijetiti da svi pozivi prema mikroservisima prolaze kroz *NodePort. NodePort* je Kubernetes-ov servis koji nam omogućava otvaranje portova. Komunikacija na *NodePort* dostupna je samo unutar klastera. Taj servis također služi kao *LoadBalancer.* Ukoliko postoji više instanci jednog mikroservisa (više *Pod*-ova), *NodePort* određuje kojoj instanci aplikacije će poziv biti dodijeljen. *Pod*-ovi u Kubernetesu su slični Docker kontejnerima. To je okruženje koji pokreće jednog ili više Docker kontejnera. *Pod*-ovima upravlja *Deployment.* On određuje koliko instanci aplikacije želimo imati, te u slučaju da jedna od njih padne, automatski stvara novu. Kao poslužitelja ove aplikacije odabrao sam DigitalOcean. Razvijena aplikacija dostupna je na adresi: https://simon-kosmrl-diplomski.xyz/



Slika 10: Arhitektura aplikacije unutar Kuberentes klastera

5. Testiranje sigurnosti web aplikacije

Testiranje sigurnosti je proces pronalaženje prijetnji i slabosti softvera sa ciljem zaustavljanja zlonamjernih napada. Svrha testova sigurnosti je identifikacija svih mogućih propusta i slabosti softvera koje bi prouzročile gubitak informacija, prihoda ili ugleda organizacije i njihovih zaposlenika. Postoji 7 različitih tipova testiranja sigurnosti:

1. Skeniranje ranjivosti – korištenjem automatiziranog alata sa ciljem prepoznavanja ranjivosti sustava.
2. Procjena rizika – analiza rizika sigurnosti. Rizici su rangirani po veličini moguće štete: nizko rizični, srednje rizični, visoko rizični. Preporučuju se postupci kontrole rizika, kao i mjere ublažavanja.
3. Skeniranje sigurnosti – automatiziranim ili manualnim alatom utvrđuju se slabosti mreže i sustava.
4. Penetracijsko testiranje – simuliranje zlonamjernih napada. Analiza potencijalnih slabosti sustava na vanjske prijetnje.
5. Revizija sigurnosti – inspekcija unutarnjih dijelova sustava - potencijalni nedostatci i slabosti internog sustava.
6. Etičko hakiranje – unajmljivanje stručnjaka koji pokušava hakirati sustav ili mrežu sa ciljem otkrivanja nedostataka i slabosti sigurnosti.
7. Procjena držanja – kombiniranje etičkog hakiranja, skeniranja sigurnosti i procjene rizika kako bi se prikazala cijela slika sigurnosti.

Praktički dio diplomskog rada pretežito će se bazirati na penetracijskom testiranju web aplikacije. To je način na koji etički hakeri dolaze do saznanja nedostataka sigurnosti. Penetracijsko testiranje bazira se na simuliranju napada na sustav, na način kako bi to uradili i zli hakeri.

Testiranje se dijeli u 5 glavnih faza:

1. Prikupljanje informacija (*Footprinting and Reconnaissance*) – Ovo je prva faza penetracijskog testiranja i smatra se najbitnijom fazom. Cilj ove faze je prikupljanje općenitih informacija o ciljanoj meti. Pretraživanje ove faze može biti aktivno i pasivno. Aktivno istraživanje odnosi se na prikupljanje informacija direktno od ciljanog sustava (slanjem poziva). Pasivno istraživanje je prikupljanje informacija indirektno (pretraživanjem web-a, Google pretraživanja i slično). Prikupljanjem podataka o meti lakše je pronaći

slabosti sustava i pokušati probiti sigurnost. Ova faza smatra se najbitnijom jer se u njoj prikupljaju sve informacije o meti kako bi se isplanirao napad. Ukoliko su prikupljene informacije lažne, napad se neće moći izvršiti. Prikupljene informacije ove faze su: komponente mreže, informacije o uređajima, informacije o otvorenim portovima i točkama pristupa, informacije o operacijskom sustavu i slično.

2. Skeniranje mreže (*Scanning*) – U ovoj fazi koriste se jedan ili više alata skeniranja kako bi se skupile informacije o meti. Korištenjem alata poput port skenera, skenera mreže (*network mapper*) i skenera ranjivosti (*vulnerability scanner*), dolazimo do boljeg razumijevanja strukture sustava. Prikupljene informacije služe za istraživanje ranjivosti sustava te se napad može izvesti na vrlo sofisticiran način. U ovoj fazi dolazimo do saznanja ranjivosti sustava te sukladno njima stvaramo plana napada.
3. Izvedba napada (*Exploitation/Gaining Access*) – U ovoj fazi penetracijski testeri pokušavaju dobiti neovlašteni pristup sustavu i iskoristiti ranjivosti pronađene u prošloj fazi. Cilj ove faze je prikupljanje osjetljivih podatka i stjecanje što veće ovlasti nad sustavom, pritom koristeći se različitim alatima.
4. Održavanje pristupa (*Maintaining Access*) – U ovoj fazi pokušava se zadržati pristup sustavu. Stvaranjem trajne veze moguće je pronaći skrivene ranjivosti sustava. Trajnu vezu sa sustavom moguće je stvoriti tehnikom stražnjih vrata (*backdoor*). Vrlo bitno je da se nakon penetracijskog testiranja uklone alata stvaranja trajne veze, jer u suprotnom sustav ostaje ranjiv na ovu vrstu napada.
5. Prekrivanje tragova – Zadnja faza testiranja je uklanjanje tragova hakiranja. Brisanjem zapisa i otisaka puštenih tokom napada, administratorima sustava otežano je prepoznavanje nedozvoljenog prisustva. Ovo pomaže pen testerima da razmišljaju kao hakeri i prekriju svoje tragove.

U ovome diplomskom radu, za testiranje sigurnosti aplikacije korišten je Kali Linux – vodeća Linux distribucija za testiranje prodora, etičko hakiranje i procjenu sigurnosti mreže. Ova distribucija Linux-a nudi nam skoro sve alate koji će nam poslužiti za izvršavanje napada na našu aplikaciju. Kali Linux ovog diplomskog rada instaliran je na virtualnoj mašini koja se pokreće na Windowsima (VirtualBox). Korištena verzija je Kali Linux 2021.2.

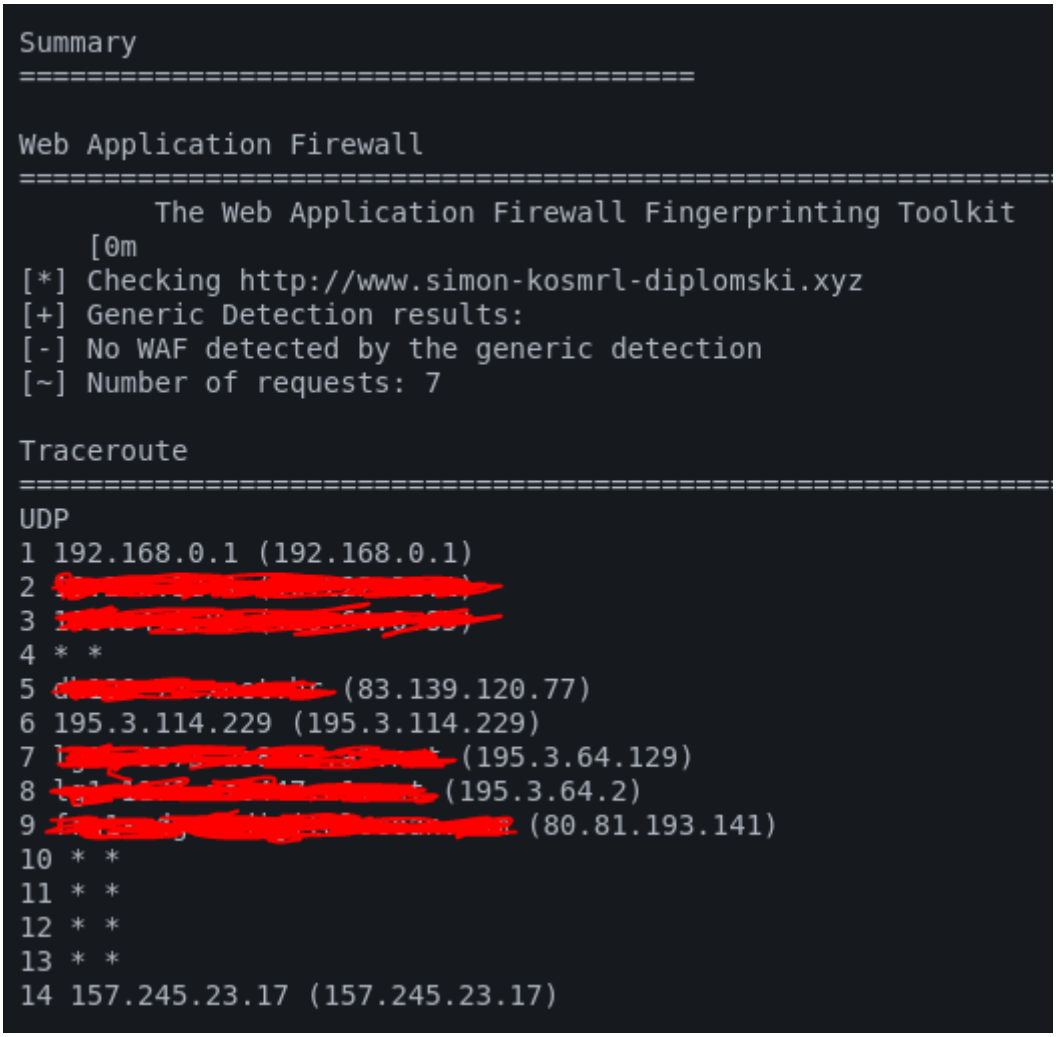
5.1. Prikupljanje informacija o web aplikaciji

5.1.1. Prikupljanje generalnih podataka

Postoje mnogi alati i načini prikupljanja informacija o našoj web aplikaciji. Znamo da se aplikacija nalazi na adresi „simon-kosmrl-diplomski.xyz” te samim time možemo započeti istraživanje.

Jedan od alata koji ćemo iskoristit za prikupljanje informacija o web aplikaciji je *Discover.* *Discover* alat nam nudi automatizirani proces aktivnog prikupljanja informacija o aplikaciji. Baziran je na komandnoj liniji i jednostavan je za korištenje. Za početak prikupljanja informacija, u alatu odaberemo koje metode želimo koristiti za pretraživanje (pasivno ili aktivno prikupljanje podataka) i upišemo domenu za koju želimo prikupiti informacije.

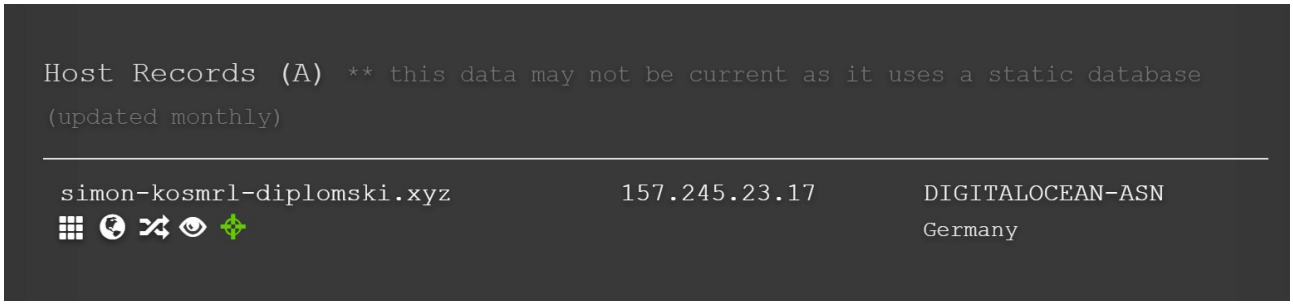
Korištenjem *Discover-a* prikupljene su informacije o vatrozidu (*firewall*), DNS zapisa i putanji paketa do web aplikacije (*traceroute*). Na slici 11 prikazano je generirano izvješće. Iz izvješća saznajemo da nije detektiran WAF (*Web Application Firewall*). Također vidljiva je IP adresa servera na kojem je postavljena aplikacija.



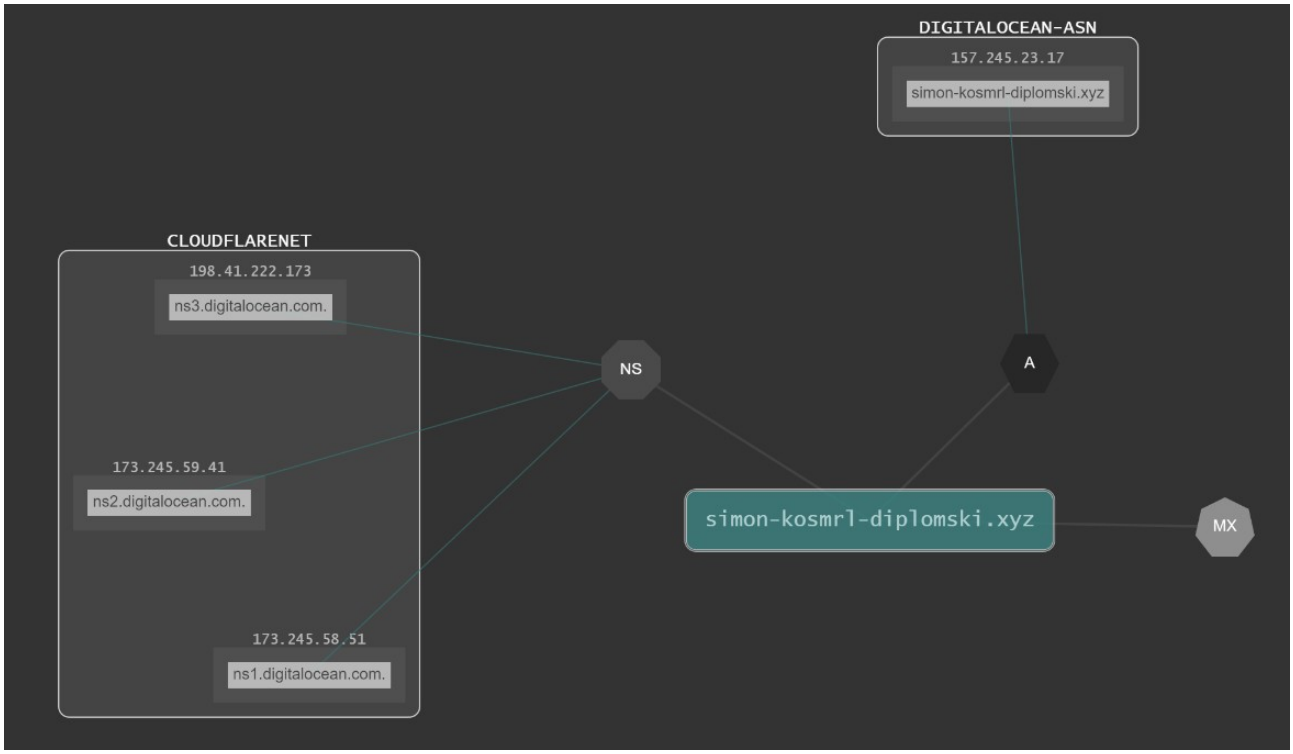
Slika 11: Dio izvješća istraživanja informacija pomoću Discover alata

Osim aktivnog istraživanja, *Discovery* može pronaći i informacije pasivnog istraživanja, kao što su liste email adresa domene koja se istražuje, informacije o društvenim mrežama (linkove i statistiku popularnosti). Naša domena nema prijavljeni email adresa i nije dostupna na društvenim mrežama, stoga alat nije pronašao takve podatke.

Osim *Discover-a*, obavljeno je istraživanje *DNSdumpster-om*. *DNSdumpster* je besplatan online alat za istraživanje domene. Korištenjem alata saznajemo da je poslužitelj aplikacije *DigitalOcean*. Server je smješten u Njemačkoj. Alat je stvorio i vizualni prikaz mreže povezivanja domene sa IP adresama i DNS zapisima.



Slika 12: Izvješće DNSdumpster-a - Zapisi poslužitelja

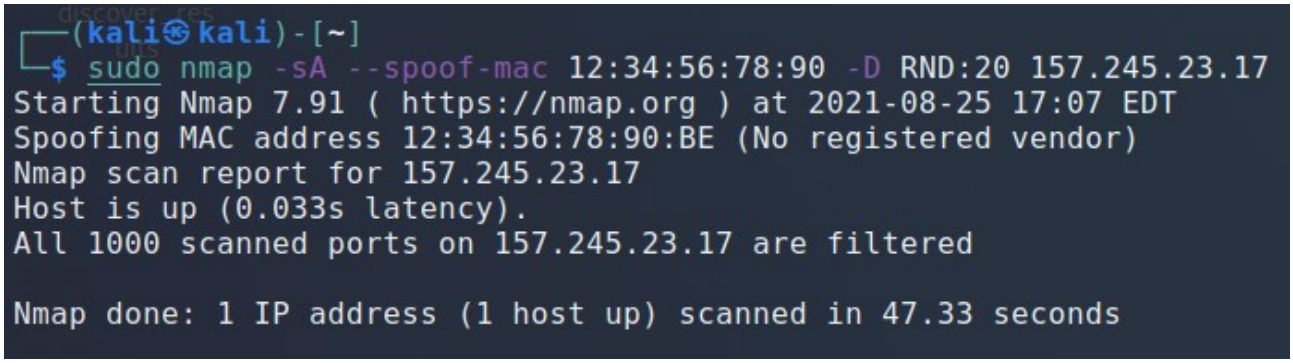


Slika 13: Izvješće DNSdumpstera - Vizualni prikaz lokacije DNS zapisa domene

5.1.2. Skeniranje mreže aplikacije

Za skeniranje mreže aplikacije koristiti će se alat *nmap*. *Nmap (Network mapper)* je besplatan alat otvorenog izvora koji nam nudi mogućnosti otkrivanja korisnih informacija o vatrozidu koji stoji ispred aplikacije, informacije o otvorenim i zatvorenim portovima te saznanje servisa koji se pokreću na određenim portovima.

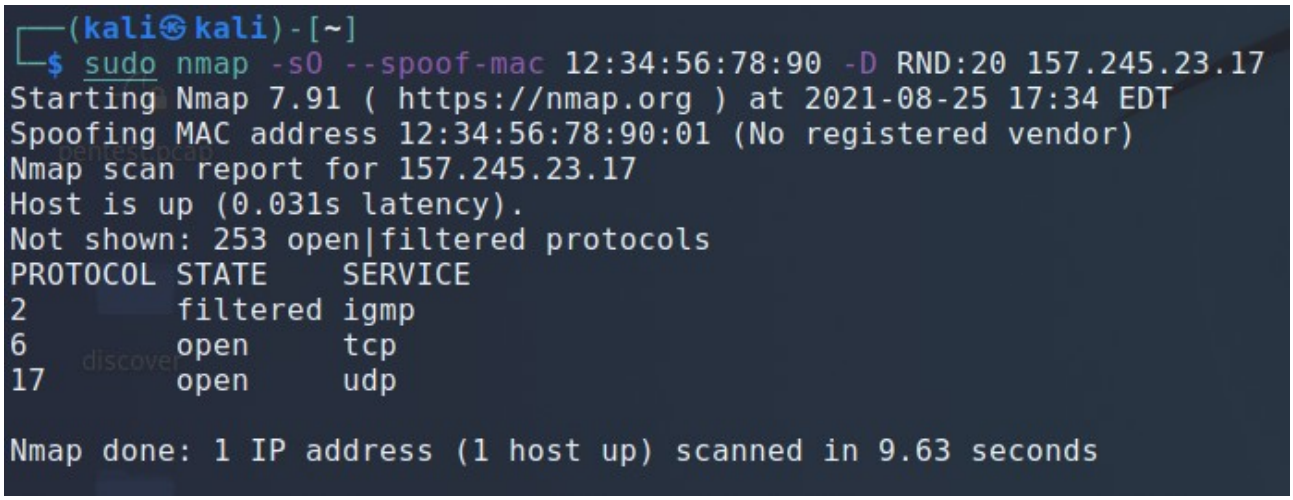
Skeniranje ćemo započeti istraživanjem vatrozida (*firewall*). Prekriti ćemo našu pravu MAC i IP adresu koristeći --spoof-mac (*MAC address spoofing*) i -D opcije (koristi će se 20 nasumično odabranih IP adresa). Opcijom -sA izvršiti će TCP ACK skeniranje (Slika 14).



Slika 14: Skeniranje postavki vatrozida - nmap

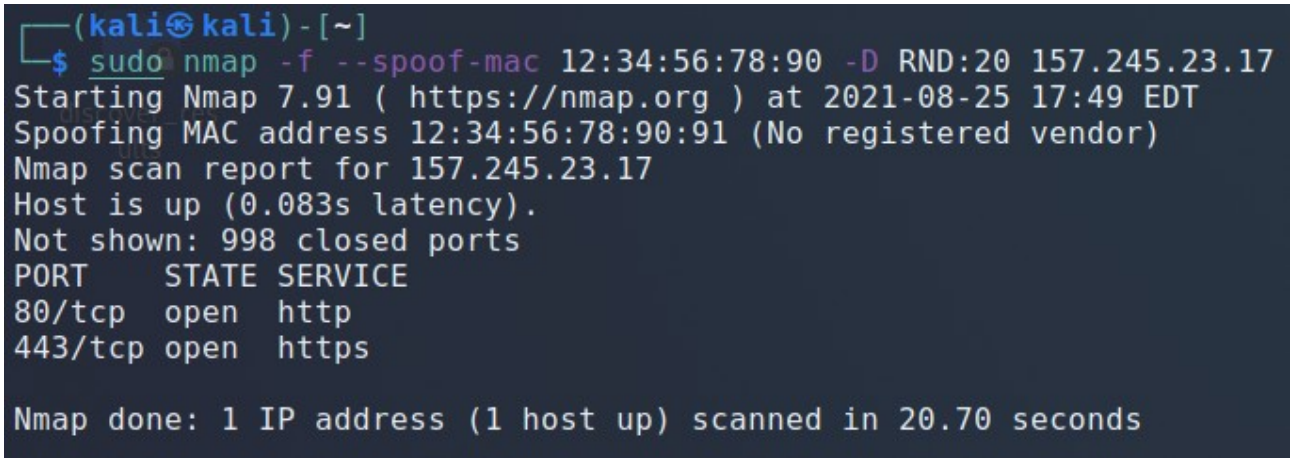
Sa slike možemo primijetiti kako je skeniranje trajalo 47.30 sekundi. Skenirano je 1000 portova i svi portovi su filtrirani. Ovo skeniranje ne može prepoznati koji portovi su zapravo otvoreni, no činjenica da su svi filtrirani govori nam o tome da smo naišli na vatrozid ili na određen set pravila rutera koju su filtrirala pakete.

Alatom možemo skenirati koji IP protokoli su podržani. To činimo opcijom -sO sa kojom saznajemo da su otvoreni portovi TCP-a i UDP-a (Slika 15).



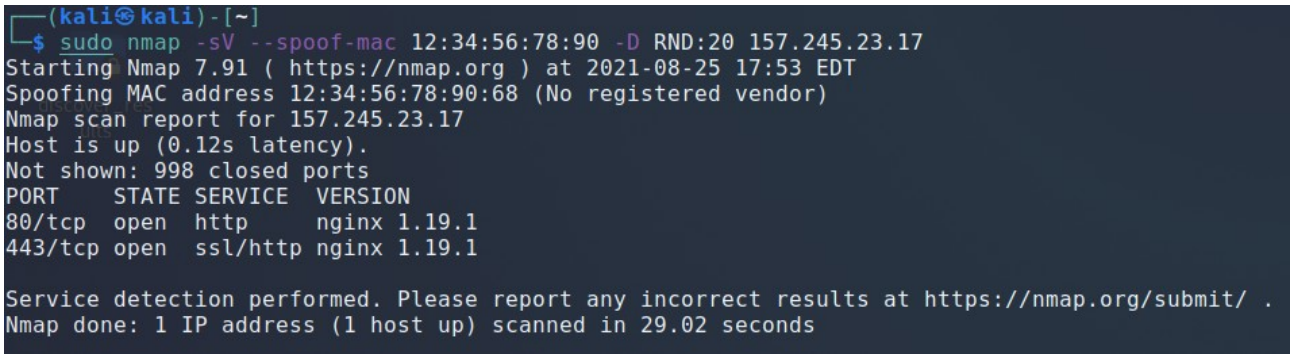
Slika 15: Skeniranje podržanih IP protokola - nmap

Skeniranje alatima kao nmap često može biti detektirano vatrozidom, no postoje načini na koje možemo to izbjeći. Korištenjem opcije -f skeniranje će biti izvršeno malim fragmentiranim IP paketima.



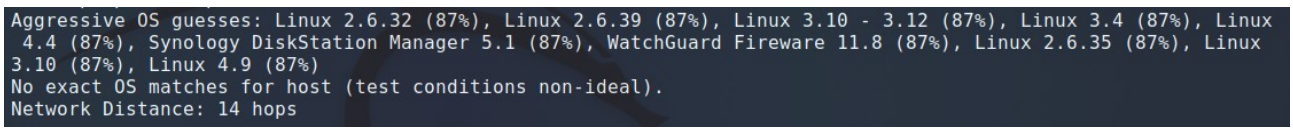
Slika 16: Fragmentirano skeniranje - nmap

Ovim skeniranjem saznajemo da su otvoreni portovi 80 i 443, koji su standardni portovi HTTP i HTTPS servisa. Nmap alat nudi nam opciju skeniranja verzija tih servisa. Opcijom -sV dolazimo do saznanja da su servisi nginx verzije 1.19.1 (Slika 17).



Slika 17: Skeniranje verzija servisa - nmap

Posljednje skeniranje ovim alatom, omogućiti će nam da otprilike saznamo na kojem operacijskom sustavu je pokrenuta naša aplikacija. Alat je 87% siguran da je u pitanju jedna od navedenih verzija Linux operacijskog sustava. Na listi se također nalaze Synology DiskStation Manager i WatchGuard Fireware.



Slika 18: Skeniranje operacijskog sustava - nmap

5.2. Traženje postojećih ranjivosti servisa

Skeniranjem mreže naše aplikacije saznali smo da servisi na portovima 80 i 443 koriste Nginx 1.19.1. Korištenjem alata *searchsploit* možemo pretražiti postojeće ranjivosti Nginx-a.

```
┌──(kali㉿kali)-[/usr/…/exploitdb/exploits/cgi/webapps]
└─$ searchsploit nginx
------------------------------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                                            |  Path
------------------------------------------------------------------------------------------ ---------------------------------
Nginx (Debian Based Distros + Gentoo) - 'logrotate' Local Privilege Escalation            | linux/local/40768.sh
Nginx 0.6.36 - Directory Traversal                                                        | multiple/remote/12804.txt
Nginx 0.6.38 - Heap Corruption                                                            | linux/local/14830.py
Nginx 0.6.x - Arbitrary Code Execution NullByte Injection                                 | multiple/webapps/24967.txt
Nginx 0.7.0 < 0.7.61 / 0.6.0 < 0.6.38 / 0.5.0 < 0.5.37 / 0.4.0 < 0.4.14 - Denial of S     | linux/dos/9901.txt
Nginx 0.7.61 - WebDAV Directory Traversal                                                 | multiple/remote/9829.txt
Nginx 0.7.64 - Terminal Escape Sequence in Logs Command Injection                         | multiple/remote/33490.txt
Nginx 0.7.65/0.8.39 (dev) - Source Disclosure / Download                                  | windows/remote/13822.txt
Nginx 0.8.36 - Source Disclosure / Denial of Service                                      | windows/remote/13818.txt
Nginx 1.1.17 - URI Processing SecURIty Bypass                                             | multiple/remote/38846.txt
Nginx 1.3.9 < 1.4.0 - Chuncked Encoding Stack Buffer Overflow (Metasploit)                | linux/remote/25775.rb
Nginx 1.3.9 < 1.4.0 - Denial of Service (PoC)                                             | linux/dos/25499.py
Nginx 1.3.9/1.4.0 (x86) - Brute Force                                                     | linux_x86/remote/26737.pl
Nginx 1.4.0 (Generic Linux x64) - Remote Overflow                                         | linux_x86-64/remote/32277.txt
PHP-FPM + Nginx - Remote Code Execution                                                   | php/webapps/47553.md
------------------------------------------------------------------------------------------ ---------------------------------
Shellcodes: No Results
```

Slika 19: Pretraživanje ranjivosti Nginx-a - searchsploit

Kali Linux posjeduje mnoge skripte za izvršavanje postojećih napada. Pretraživanjem *searchsploit*-a nismo pronašli poznatu ranjivost naše verzije Nginx-a. Primjećujemo kako prijašnje verzije servisa posjeduju neke ranjivosti koje se mogu iskoristiti. *Searchsploit* za svaku od poznatih ranjivosti verzija prikazuje lokaciju na disku kojoj može pristupiti kako bi došli do skripte za izvršavanje napada ili tekstualne datoteke koja sadrži opisa ranjivosti i postupak napada. Za sigurnost web aplikacija vrlo je važno da se koriste najnovije verzije servisa i alata na kojima su pokrenute. Redovitim ažuriranjem verzija korištenog softvera možemo smanjiti rizik od uspješnog izvršavanja zlonamjernog napada.

Osim *searchsploit*-a, određene ranjivosti možemo pretraživati i izvršavati preko *Metasploit framework-a*. *Metasplot* je najveća baza skripti i programa kojima se mogu ispitati postojeće ranjivosti servera i mreža. Vezija 6.0.45 sadrži 592 različitih setova zlonamjernog koda te više od dvije tisuće *exploit-a* koji su organizirani po platformama (Android, Windows, Linux, PHP i slično). Alat je baziran na konzoli te se preko njega mogu pokretati napadi ili pročitati dobro definirane dokumentacije sa svaku poznatu ranjivost koju želimo iskoristiti.

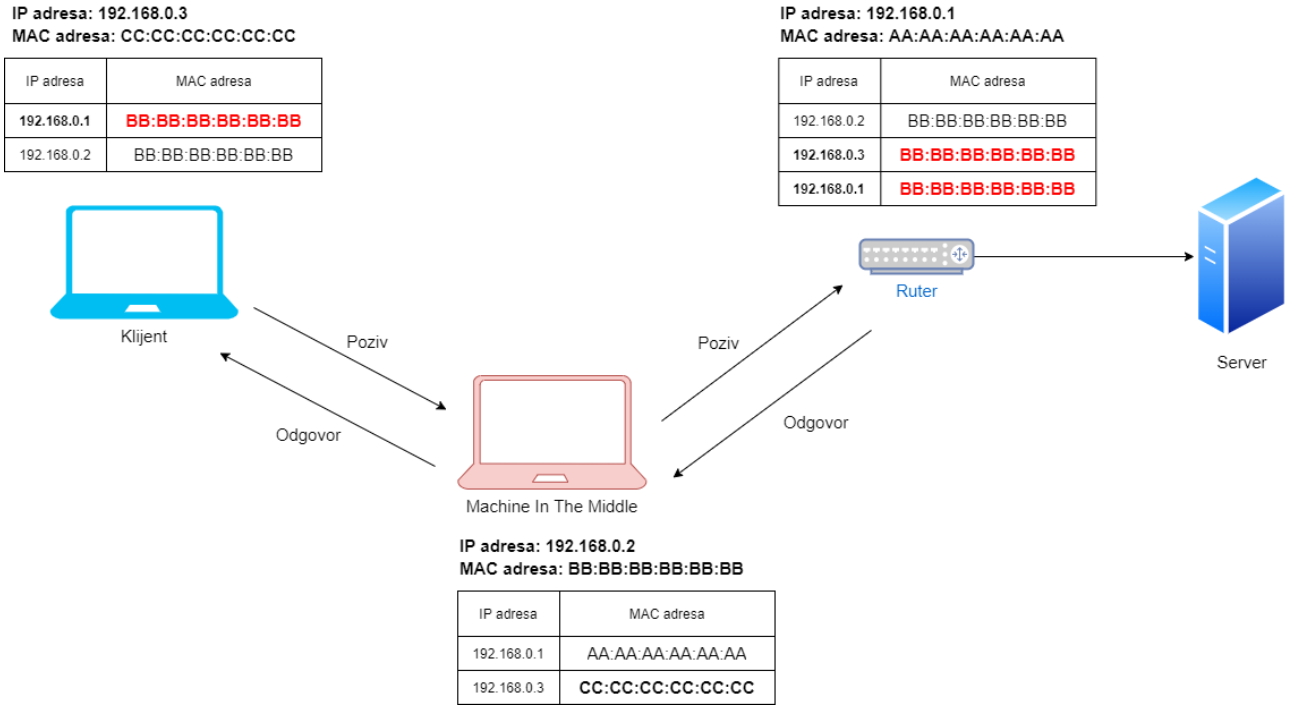
```
       =[ metasploit v6.0.45-dev                          ]
+ -- --=[ 2134 exploits - 1139 auxiliary - 364 post        ]
+ -- --=[ 592 payloads - 45 encoders - 10 nops             ]
+ -- --=[ 8 evasion                                       ]
```

Slika 20: Metasploit kolekcija

5.3. Mrežni napadi aplikacije

5.3.1. Man In The Middle

Testiranje ranjivosti aplikacije u oblaku započeti će napadom koji je prouzročio potrebu stvaranja sigurne veze na internetu. *Man in the Middle* napad bazira se na presretanju komunikacije između klijenta i servera. Najčešća varijanta ovog napada izvršava se na lokalnoj mreži. Ukoliko neko računalo dobije pristup lokalnoj mreži (najčešće probijanjem *wireless* zaštite) može prisluškivati sve komunikacije koje se odvijaju unutar te mreže. Napad se bazira na *ARP spoofing-u*. *ARP (Address Resolution Protokol)* je protokol koji spaja IP adrese sa pripadajućim MAC adresama. Lažiranje podataka tog protokola omogućuje prolazak svih podataka mreže kroz jedno računalo.

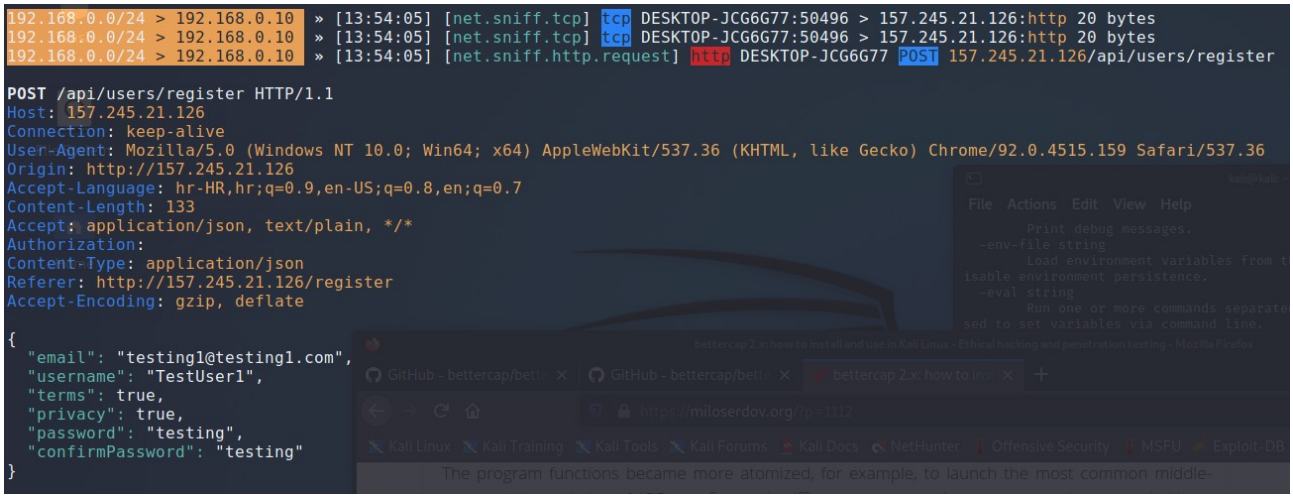


Slika 21: Vizualni prikaz ARP spoofinga - Man In The Middle napad

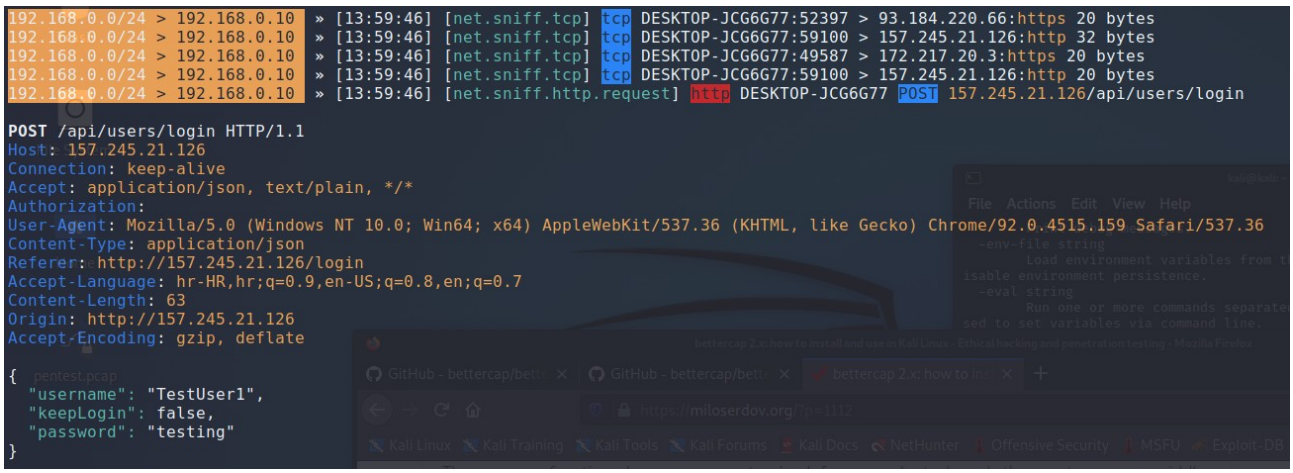
Ovakav napad je moguće izvršiti korištenjem *Bettercup-a*. *Bettercup* je softver koji nam omogućuje prisluškivanje podataka na mreži. Alat je baziran na konzoli te je za njegov rad nužno upisivati komande. Napad započinje konfiguracijom servisa koji nam ovaj alat nudi. Prvo uključimo *ARP spoofing* lokalne mreže pokretanjem naredbe `arp.spoof on`, te zatim možemo početi prisluškivati promet mreže pokretanjem naredbe `net.sniff on`.

Korištenjem HTTP protokola klijent pristupa našoj stranici. Sve potrebne datoteke za rad te stranice preuzimaju se i generiraju u pregledniku. Prilikom prijave na našu stranicu klijent upisuje svoje podatke i šalje poziv prema serveru kako bi se uspješno prijavio u aplikaciju. Poziv prema serveru

također se odvija HTTP protokolom. Podatke koji se šalju putem ovog protokola moguće je presretnuti. Slike 22. i 23. predstavljaju presretanje podataka *Man In The Middle* napada korištenjem *Bettercap* alata. Na slikama su prikazani svi podaci poslanog poziva (*Http Headers*) i izlistani su svi podaci uneseni od strane korisnika (*Body*).



Slika 22: Man In The Middle - poziv registracije korisnika



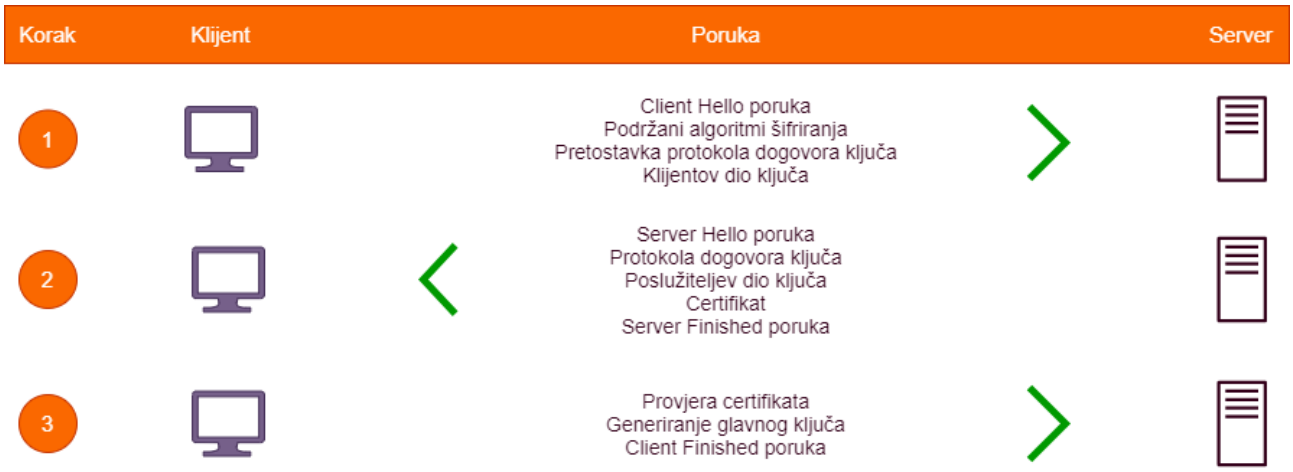
Slika 23: Main In The Middle - poziv prijave u aplikaciju

5.3.1.1. SSL/TSL

Kao obrana od ovog napada, stvoren je novi protokol – SSL (*Secure Socket Layers*). Zadaća SSL-a je pružiti sigurnu komunikaciju na internetu. To ostvaruje šifriranjem podataka HTTP-a, te se u tom slučaju taj protokol naziva HTTPS. Šifriranje podataka sprječava njihovo čitanje ukoliko ih netko presretne – samo klijent i server mogu dešifrirati poslane poruke i saznati njihovo pravo značenje. Presretačima sadržaj tih poruka izgleda kao nasumično generirani niz znakova bez ikakvog značenja.

Protokol je implementiran kao omotač oko HTTP protokola. Na OSI modelu nalazi se iznad transportnog sloja (TCP/UDP) i služi kao omotač aplikacijskog sloja. Na taj način mogu se osigurati komunikacije i ostalih protokola koji se izvršavaju na aplikacijskom sloju (SMTP, FTP i sl.). SSL kriptografski protokol započeo je Netscape 1993. godine. Organizacija IETF (*Internet Enginering Task Force*) je 1999. godine predložila novi standard kriptografije – TLS (*Transport Layer Security*). TLS je nasljednik SSL3.0 uz manje razlike poboljšanja sigurnosti.

Najnovija verzija TLS protokola koja se koristi u vrijeme pisanja ovog diplomskog rada je TLS v1.3. Princip rada te verzije protokola prikazan je na sljedećoj slici. Protokol započinje nečim što nazivamo TLS rukovanje (eng. *handshake*). Korištenjem principa simetričnog i asimetričnog šifriranja podataka ostvarena je sigurna veza. Kako bi se potvrdila autentifikacija identiteta servera, koristi se asimetrična kriptografija – certifikat izdan od strane CA (*Certificate Authority*) i privatan ključ posjeduje samo server. Korištenjem javnog ključa (certifikata) podaci se šifriraju, a kako bi se dešifrirali, moramo posjedovati privatni ključ koji pripada tom javnom ključu. Na taj način asimetrična kriptografija pripomaže pri rukovanju klijenta i servera u stvaranju sigurne razmjene dijelova ključeva između njih. Kada obje strane razmjene sve potrebne ključeve, stvara se glavni ključ koji je kombinacija svih razmijenjenih ključeva. Glavi ključ koristi se za šifriranje svake druge komunikaciju između servera i klijenta nakon ostvarivanja uspješnog rukovanja (simetrična kriptografija). U TLSv1.3 rukovanje se izvađa u 3 koraka, što ga čini efikasnijim od prethodne verzija koja koristi više koraka i poziva kako bi ostvarila sigurnu vezu.



Slika 24: TLS v1.3 Handshake između klijenta i servera

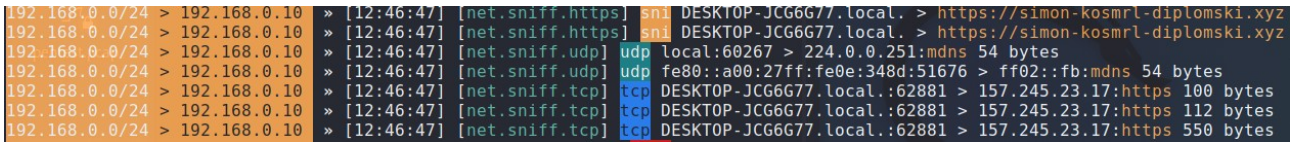
Prvi korak: Klijent šalje *Client Hello* poruku serveru. U toj poruci sadrže se svi paketi za šifriranje koje podržava klijent, predviđa se koji će protokol dogovora ključeva poslužitelj vjerojatno odabrati i šalje se klijentov dio ključeva.

Drugi korak: Server vraća poruku klijentu. U toj poruci odabran je protokol dogovor ključa. Šalje se

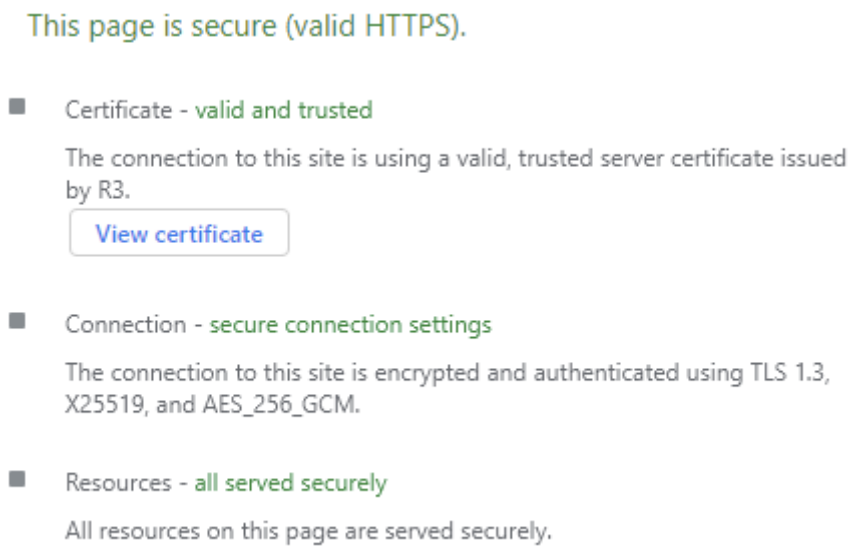
poslužiteljev dio ključa, njegov certifikat i poruka *Server Finished* ukazujući da je server generirao glavni ključ te da ima sve potrebne podatke za ostvarivanje sigurne veze.

Treći korak: Klijent provjerava dobiveni certifikat i generira glavni ključ vraćajući serveru *Client Finished* poruku. Poruke završetka bitne su za provjeru lažiranja podataka. Sadrže svu prošlu komunikaciju te ako je netko prilikom slanja poruka promijenio njihov sadržaj, klijent i server neće imati zapise komunikacije. U tom slučaju veza je otkazana.

Za stvaranje sigurne veze na Kubernetes cluster, poslužio je *cert-manager*. Korištenjem *cert-manager-a* dodani su svi certifikati i ključevi potrebni za rad TLS-a. Stvoreni certifikat izdan je od strane *Let's Encript Certificate Authority-a.* Nakon što se konfigurirao TLS, komunikacija sa serverom je sigurna te se podaci unutar sigurne veze ne mogu lako dešifrirati. Na slici 25 primjećujemo da se uspostavila sigurna HTTPS veza za serverom te *Bettercap* ne može raspoznati koji podaci su poslani i zaprimljeni . Aplikacija u oblaku podržava najnoviju verziju TLS-a (v1.3). Slika 26 prikazuje pregled sigurnosnih protokola i algoritama korištenih za stvaranje sigurne veze između servera i preglednika Google Chrome.



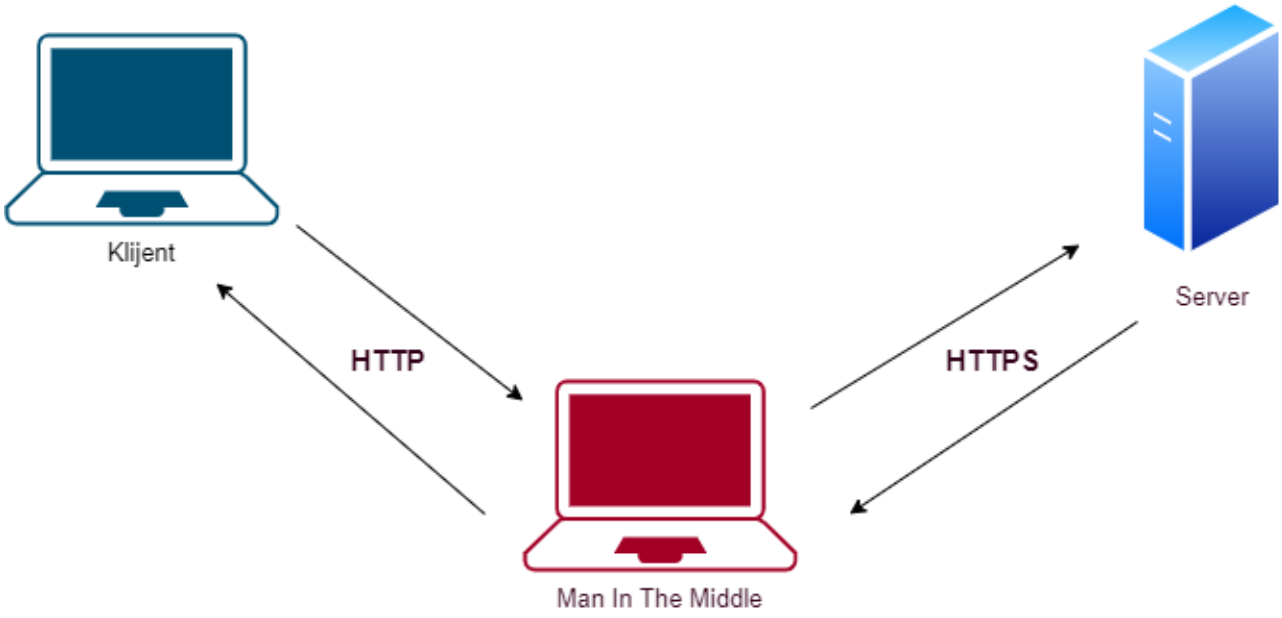
Slika 25: Presretanje sigurne veze



Slika 26: Pregled sigurnosti veze - Google Chrome

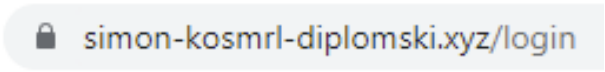
5.3.1.2. SSL Strip i HSTS hijack napadi

Preglednici weba imaju jednu veliku manu. Prilikom posjećivanja stranice putem upisa web adrese preglednici pokušavaju uspostaviti HTTP vezu. Ovo može prouzročiti veliki problem jer ukoliko netko prisluškuje promet mreže može izvršiti *SSL Strip / HSTShijack* napad na klijenta. Napad funkcionira na način da se web stranica poslužuje klijentu preko HTTP veze, dok mašina koja sluša promet mreže, uspostavlja sigurnu vezu sa poslužiteljem/serverom te web stranice. Slika 27 vizualno predstavlja navedeni napad.



Slika 27: SSL Strip - HSTS hijack napad

Postoji velika mogućnost da klijent ne vidi da uspostavljena veza nije sigurna jer je web stranica u njegovom pregledniku normalno učitana. Za obranu od ovog napada potrebno je postaviti pravilnu HSTS (*HTTP Strict Transfer Security*) politiku korištenja sigurne mreže. Jednom kada preglednik dohvati navedenu politiku svaki naknadni poziv prema tom serveru biti će pokrenut na HTTPS protokolu. Kao posljedica toga, za ovaj napad da bi bio uspješan veoma je bitno da klijent prvi put pristupa nekoj web stranici i to preko HTTP veze. Moderni preglednici posjeduju listu domena za koje je poznato da podržavaju i preferiraju HTTPS vezu te se prilikom spajanja na te stranice automatski uspostavlja HTTPS veza. Za prijavljivanje stranice koja preferira sigurnu vezu potrebno je postaviti *preload* politiku HSTS-a te prijaviti domenu na stranici https://hstspreload.org/ gdje je moguće saznati ako je stranica kvalificirana za prijavu na listu. Ukoliko se dobije potvrdna poruka, domena će se dodati na listu. U modernim preglednicima stranica će se i na prvom spajanju pokušati spojiti na HTTPS, što u potpunosti sprječava ovaj napad. Za starije preglednike, moguće je obavijestiti klijenta da provjeri ako se komunikacija odvija na sigurnim kanalima.

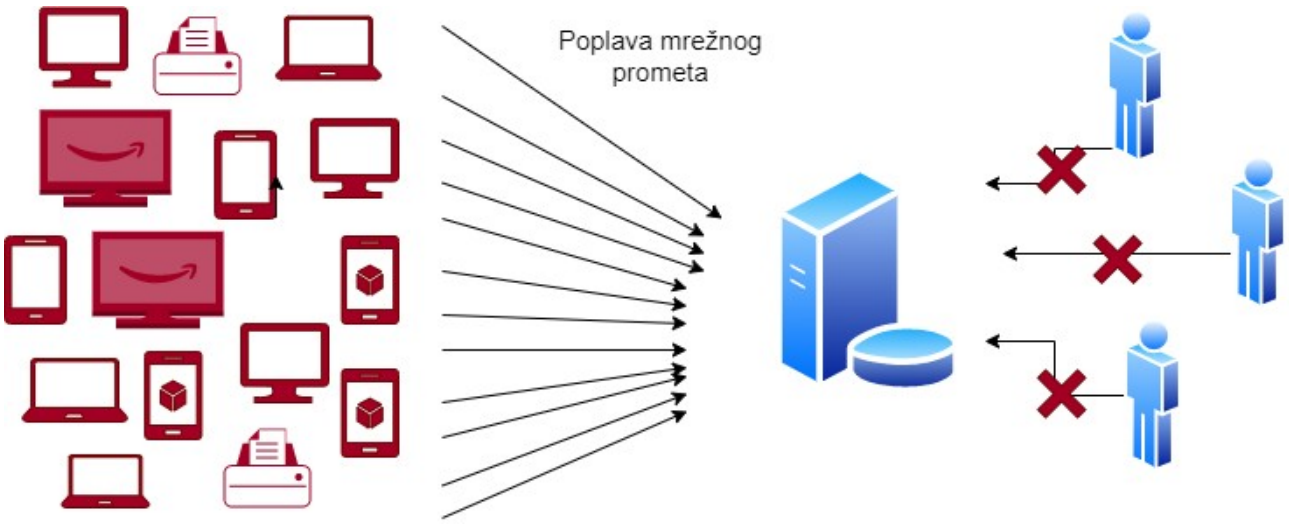


Slika 28: Prikaz sigurne veze – ikona lokota

5.3.2. DDoS – Denial of Service napadi

Napad uskraćivanjem resursa (*DoS*) je zlonamjerni pokušaj prekidanja normalnog prometa podatka nekog servera, usluge ili aplikacije u oblaku. Poplavom mrežnog prometa pokušava se preopteretiti ciljani sustav. Ukoliko poplave mrežnog prometa proizlaze iz više izvora, takav napad postaje distribuirani napad (DDoS). Uspješnim izvršavanjem *DDoS* napada pristup korisnicima usluge u oblaku je onemogućen.

Pojavom *IoT*-a (*Internet of Things*) mnogi uređaji dobili su pristup internetu. Zlonamjerni softver koji prouzročuje distribuirane napade mreže može se dostaviti na mnogobrojne uređaje. Zbog brojnosti uređaja koji imaju pristup internetu, *DDoS* napadi postaju sve opasniji. Jedan od napada korištenjem *IoT*-a zabilježen je 2016. godine. Napad se izvršio na *DNS* provajder *Dyn*. Pomoću zlonamjernog softvera nazvanog Miriai, izvršen je *DDoS* napad vojskom uređaja kao što su kamere, pametne televizije, printeri i drugo. Kako je *DDoS* napad baziran na mreži, njegova efikasnost se broji u prometu bitovima u sekundi. Jedan od najvećih zabilježenih napada izvršen je na *Google Cloud* uslugu. Zabilježena efikasnost tog napada bila je 2.54 Tbps.



Slika 29: Vizualni prikaz DDoS napada

Zaraženi uređaji softverom koji mogu prouzročiti poplavu mrežnog prometa nazivaju se botovi.

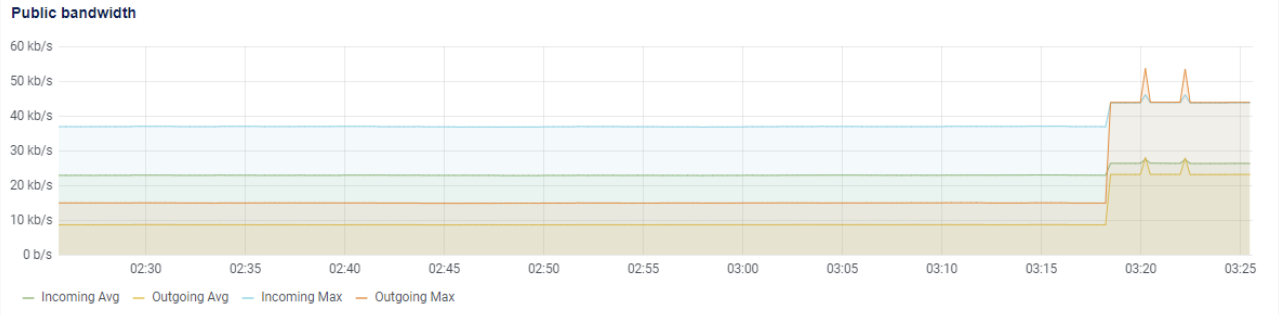
Skup takvih uređaja naziva se botnet. DDoS napadi mogu prouzročiti kaos na više slojeva OSI modela, no najčešće botovi napadaju 3. sloj (*Network*), 4. sloj (*Transport*) i 7. sloj (*Application*) OSI modela. Napadi se često dijele na 3 glavne kategorije: Volumetrijski napadi, Protokol napadi i Aplikacijski napadi.

Volumetrijski napadi koriste velike količine prometa kako bi onemogućile propusnost prometa ciljanog sustava. Ovakve vrste *DDoS* napada koriste tehnike pojačavanja prometa. Promet koji prouzročuje ovaj napad, uveliko je manji od količine prometa usmjerenog prema targetiranom sustavu. Pretrpavanjem prometa može doći do kompletnog blokiranja pristupa ciljanog sutava. Neki od primjera napad ove vrste su: *NTP Amplification*, *DNS Amplification*, *UDP Flood*, *TCP Flood*. Protok napadi iskorištavaju mane 3. i 4. sloja OSI modela. Takvi napadi preopterećuju resurse ciljanog sustava, što može proizvesti poremećaj u radu kritičkih točaka kao što je *firewall.* Primjeri takvih napada su: *SYN Flood* i *Ping of Death.*

Aplikacijski napadi usmjereni su na mane 7. sloja OSI modela. Ovi napadi su veoma sofisticirani te ih je teže eliminirati i prepoznati od drugih vrsta DDoS napada. Uspostavljanjem veze sa ciljanim serverom iscrpljuju njegove resurse uzimajući veći dio procesne snage i prometa mreže. Primjeri ovih napada su: *HTTP Flood* i *DNS Service Attacks.*

Vrlo važno je na vrijeme prepoznati DDoS napade. Za to je potrebno imati dobro postavljene sustave kontrole – nadzor prometa i potrošnje resursa. U kontrolnoj ploči *DigitalOcean-a* omogućen je nadzor nad svim resursima Kubernetes klastera. Postavljanjem upozorenja možemo biti obavješteni o promjenama normalnog rada sustava. Na slici 30 predstavljen je blagi porast prometa proizveden hping3 alatom koji je svake 100 milisekunde poslao jedan ping poziv.

Povećanjem mrežnog prometa moguće je stvoriti više instanci svakog od našeg mikroservisa, te time ublažiti DDoS napad. Horizontalno skaliranje klastera povećavanjem broja *node*-ova i pokrenutih instanci mikroservisa omogućiti će nam da naša usluga bude dostupna čak i za vrijeme DDoS napada, no ova prednost Kubernetesa dosta je skupa.



Slika 30: Porast prometa pinganjem web servera

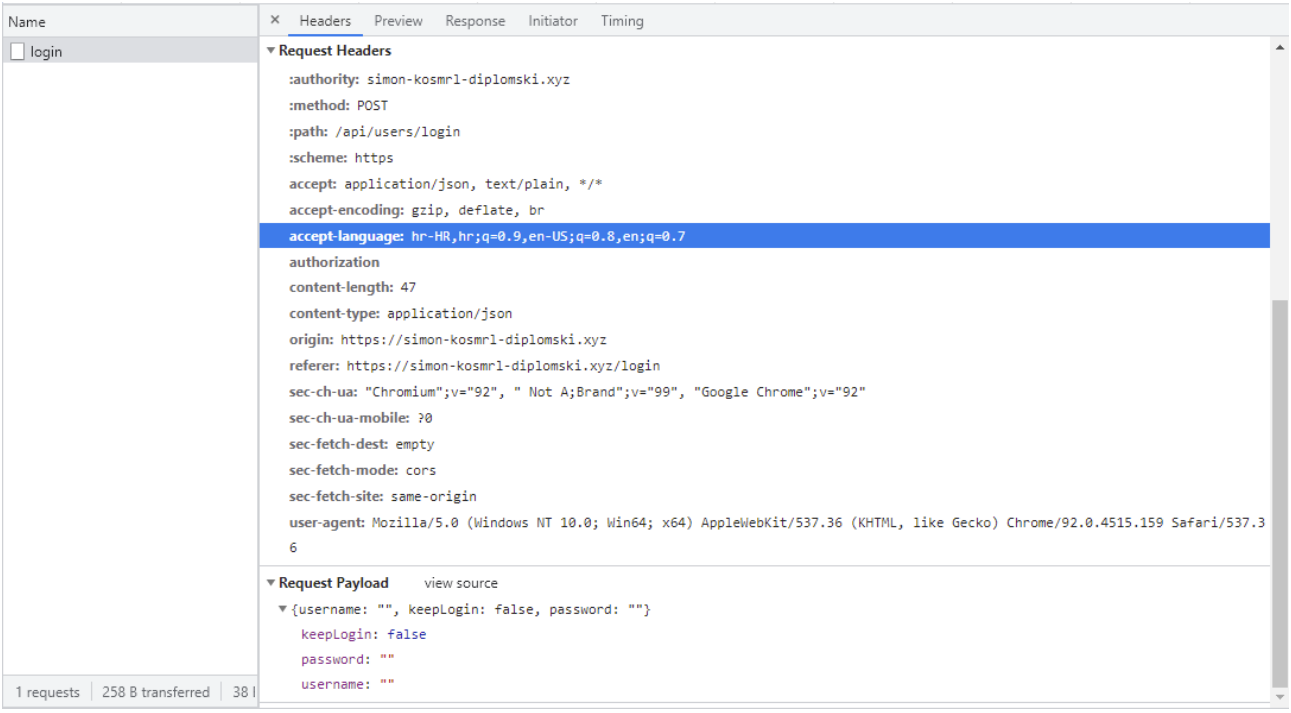
Za zaštitu od DDoS napada postavlja se IDS (*Intrusion Detection System*), čija zadaća je analiziranje prometa mreže, i IPS-a (*Intrusion Pervention System*) koji analizira zaprimljene pakete, te ovisno o njihovom sadržaju ih blokira, ukoliko je prepoznao izvršavanje napada. Dobra praksa pri obrani od DDoS napada je zaustaviti neželjeni promet prije nego što dođe do našeg servera. To čine servisi kao *Cloudflare* koji dobro raspoznaju dobar i loše promet. Ovakvi servisi mogu činiti prvu liniju obrane od DDoS napada. Kao dodatna linija obrane u Kubernetes klasteru može se konfigurirati Ingress kontroler na način da se limitira broj trenutnih veza aplikacije te da se ograniči maksimalan broj poziva unutar nekog vremenskog perioda.

5.4. Direktni napadi web aplikacije

5.4.1. Brute-force napadi

Korisnicima je za prijavu u aplikaciju potrebna registracija. Prilikom prijave potrebno je upisati registrirano korisničko ime i lozinku koja je povezana na taj račun. Ukoliko nemamo postavljenu dobru politiku prijave, web aplikacija je ranjiva na *brute-force* napade. *Brute-force* napad bazira se na isprobavanju svih mogućih kombinacija korisničkog imena i lozinke dok se ne pronađe valjana kombinacija. Za izvedbu ovog napada možemo koristiti *Burp suite. Burp* ili *Burp suite* je set alata koji se koriste za testiranje sigurnosti web aplikacija. *Intruder* je jedan od alata *Burp suita.* Koristi za simuliranje *brute-force* napada.

Prvi korak *brut-force* napada je istraživanje. Potrebno je saznati koji podaci su potrebni za prijavu te koje su karakteristike poziva koji se šalje za provjeru prijave. *Burp suite* posjeduje preglednik web stranica koji presreće pozive te detaljno prikazuje informacije svakog poziva. Iste informacije moguće je pregledati u svakom pregledniku, korištenjem alata koji najviše pomažu programerima, a to su *Browser Developer Tools. DevTools* dostupan je u svakom modernom pregledniku. Pritiskom tipke F12 unutar preglednika otvaraju se ti alati. U kartici alata *Network*, moguće je vidjeti sve pozive koje je preglednik obavio. Sljedeća slika prikazuje obavljeni poziv prijave naše aplikacije.

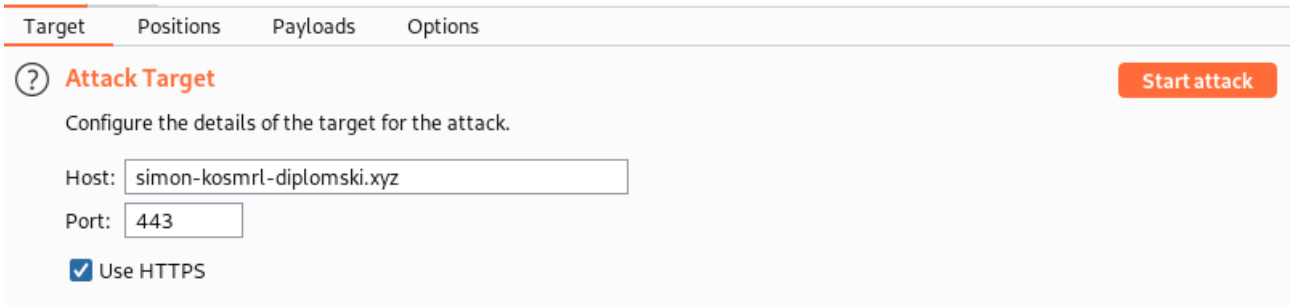


Slika 31: Alati preglednika - Primjer poziva prijave

Iz poziva primjećujemo kako je metoda POST, na putanji „/api/users/login” te za prijavu u aplikaciju šalju se sljedeći JSON podaci:

```
{
     "username":"",
     "keepLogin":false,
     "password":""
}
```

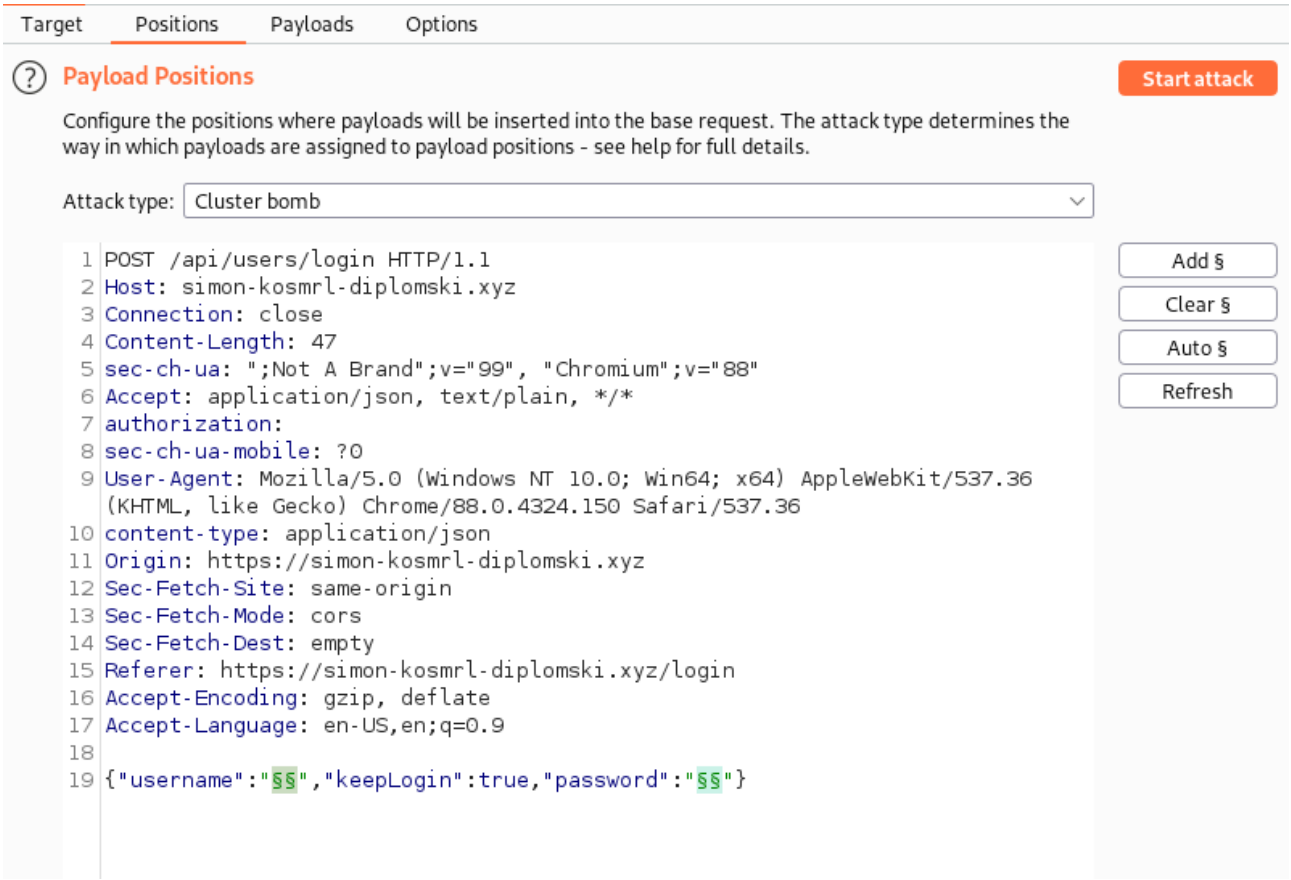
Prikupili smo sve potrebne informacije potrebne za *brute-force* napad i možemo započeti konfiguraciju *Intruder-a* koji će nam pomoći za automatiziranje API poziva. Prvo odredimo domenu i port na kojem će se napad izvesti.



Slika 32: Intruder postavke - Određivanje domene

U sljedećem koraku određujemo vrstu napada i karakteristike poziva. Vrsta napada određena je na

Cluster bomb. Ovaj napad isprobavati će sve kombinacije određenih korisničkih imena sa svim određenim lozinkama. U liniji 19 slijedeće slike primjećujemo kako su polja „*username*” i „*password*” popunjena vrijednostima „§§”. Vrijednosti tih varijabli biti će zamijenjene vrijednostima korisničkih imena i lozinki definiranih u sljedećem koraku.

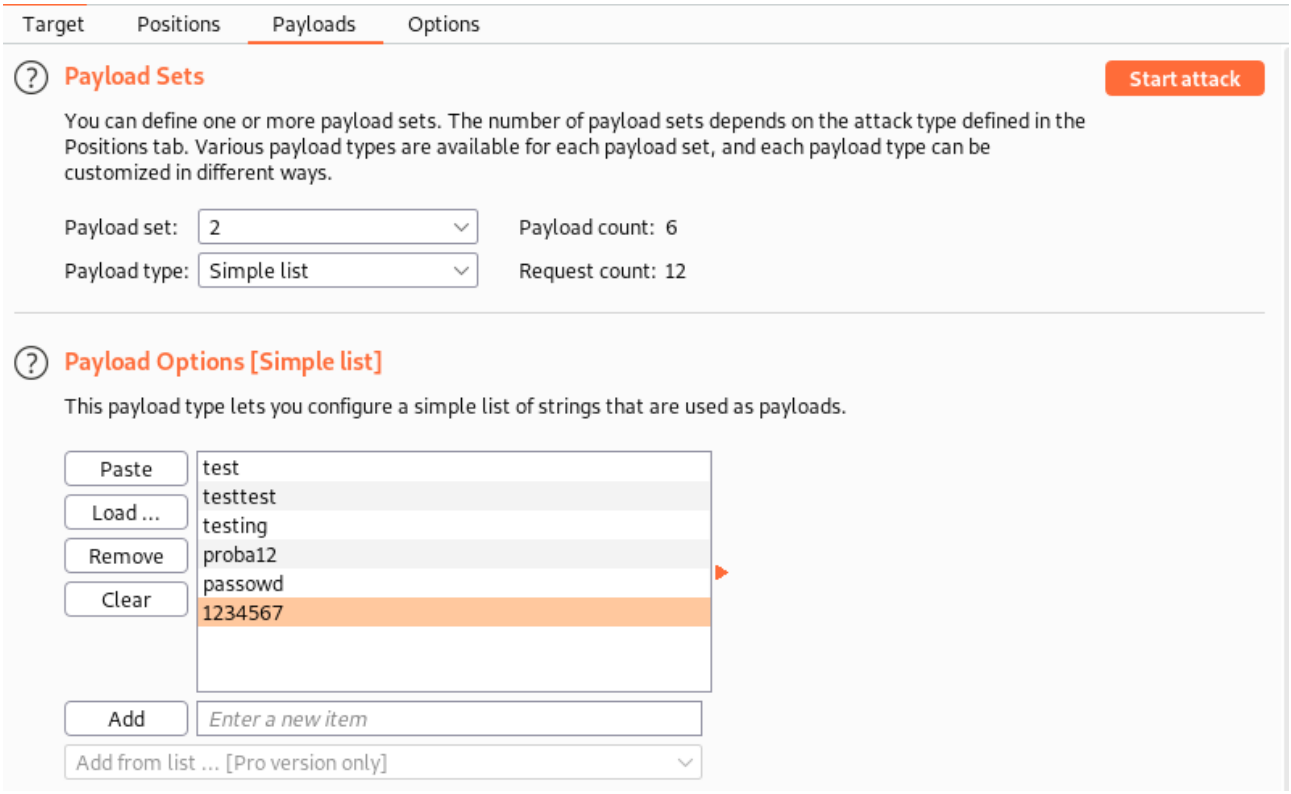


Slika 33: Intruder postavke - postavljanje karakteristika poziva

Postavljanje liste korisničkih imena i lozinki u *Burp suite-u* određuje se setovima. Set 1 sadržavati će liste korisničkih imena (TestUser1 i TestUser2), dok će set 2 sadržavati listu lozinki. Lista lozinki generirana je ručno i u njoj se nalazi 6 primjera lozinki od kojih je jedna točna. U pravom napadu na web aplikaciju lista korisnika i lista lozinki bila bi puno veća te bi sukladno tome bilo bi obavljeno puno više poziva. Broj poziva ovog testa je 12 (2 - broj korisničkih imena * 6 – broj lozinki = 12 – broj poziva sa svaku kombinaciju). Sljedeća slika prikazuje listu seta 2, odnosno listu lozinki koje će biti isprobane za prijavu korisnika TestUser1 i TestUser2.

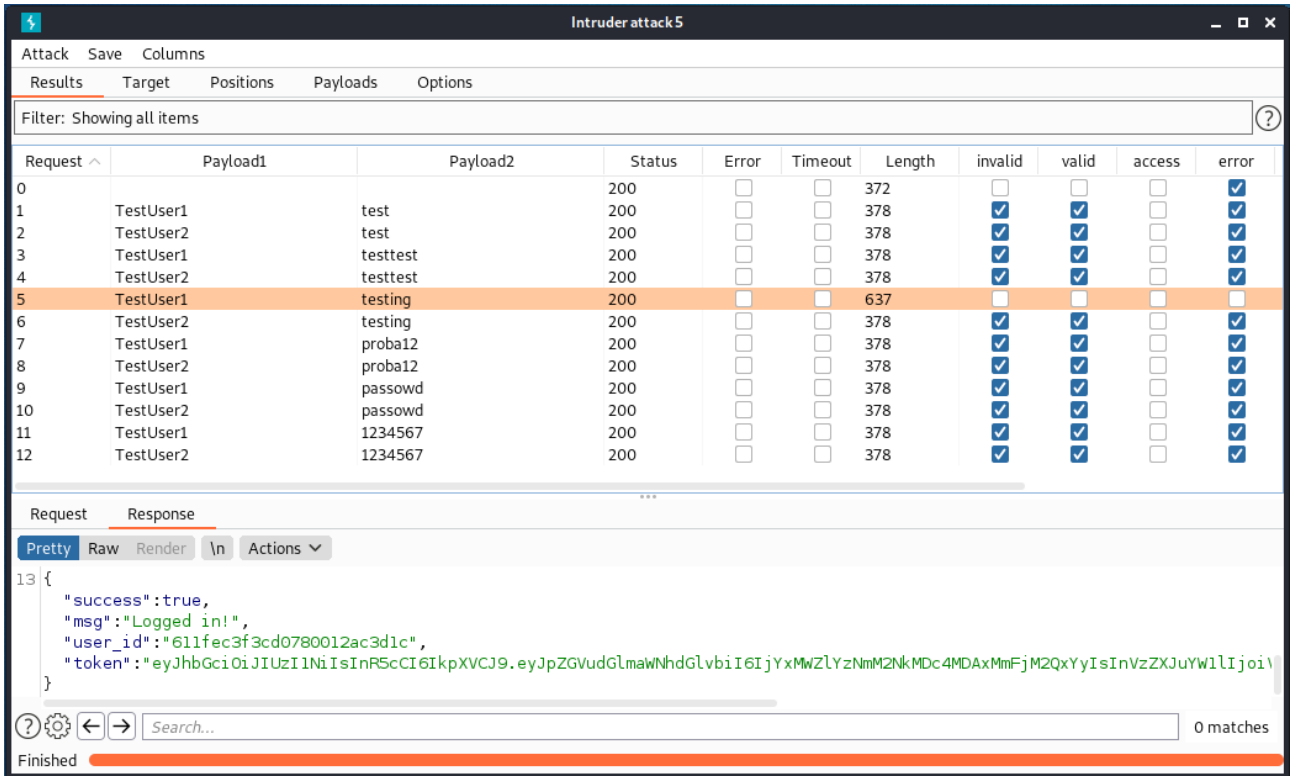
U ovom koraku izvršavanja napada najprije bi bile korištene informacije prikupljeni u prvom koraku testiranja (Prikupljanje podataka alatom *Discover*). Ukoliko su pronađene neke od e-mail adresa targetirane domene napravila bi se lista tih adresa. Ako već unaprijed znamo email adresu ili korisničko ime za koje želimo dobiti pristup, set će sadržavati samo jedno korisničko ime ili email te će se kao posljedica toga broj poziva smanjiti.

Za listu lozinki najčešće se koriste liste koje postoje u Kali Linux distribuciji. Jedna od najpoznatijih lista lozinki naziva se „*rockyou.txt*” i sadrži više od 14 milijuna različitih lozinki korištenih u 32 milijuna različitih računa.



Slika 34: Indruder postavke – postavljanje liste lozinki

Pritiskom na tipku *Start attack* izvršavamo napad. *Indruder* za nas automatizira pozive te za svaki od njih sprema odgovor servera. Odgovore je moguće filtrirati te za svaki od poziva možemo vidjeti rezultat svake kombinacije korisničkog imena i lozinke. Slika 34 prikazuje listu automatiziranih poziva koje je obavio *Indruder.* Na listi možemo primijetiti sve kombinacije koje su odrađene. Svaki poziv na listi je veličine 378 bajtova, osim jednog poziva koji je uspješan. Kombinacija korisničkog imena i lozinke tog poziva je TestUser1 i lozinka testing. Poziv vraća poruku „*Logged in*!” sa validnim tokenom koji služi za prijavu u aplikaciju. Primjećujemo kako taj poziv nema postavljenih kvačica na *invalid* i *valid* stupcima. Ukoliko posjedujemo veću listu poziva, pomoću sortiranja po tim stupcima lako možemo isfiltrirati kombinacije koje su postojeće u sustavu.



Slika 35: Intruder napad - prikaz rezultata

Jedan način obrane od *brute-force* napada je praćenje broja krivo upisanih lozinki svakog korisnika. Ukoliko se upiše 5 krivih lozinka, račun korisnika možemo zamrznuti. Svaka naknadna prijava u tom slučaju biti će onemogućena i za prijavu u aplikaciju biti će potrebno ispuniti *CAPCHA* test. *Capcha* test je vrsta autentifikacije čiji cilj je raspoznati ljude od računala. *Capcha* je engleska skraćenica za *Completly Automated Public Turing test to tell Computers and Humans Apart*. Test je izrađen sa pretpostavkom da ga računalo ili neki softver nije u stanju riješiti, no većina ljudi bi trebala uspješno riješiti ovaj test. Ukoliko se test uspješno izvrši, možemo dozvoliti dodatne pokušaje prijave u aplikaciju. Najpoznatiji primjer *reCAPTCHA* testa je besplatni Googlov test „Nisam robot”.

Slika 36: Googlov ReCAPTCHA test

Osim obrane od *brute-force* napada prijave, ovaj test možemo iskoristiti i za obranu od pretrpavanja baze podataka korisnika. Slično kao i kod *brute-force* napada, možemo izraditi skriptu koja će

registrirati nove korisnike na svakom pozivu, koristeći nasumično generirane podatke. Kao posljedica toga imati ćemo hrpu zapisa korisnika koji neće koristiti aplikaciju. Time bi se rad aplikacije mogao usporiti i dodatno bi plaćali mjesto spremišta baze podataka. Test *reCaptche* štiti nas od ove vrste napada ukoliko je za svaku registraciju potrebno riješiti test. Prilikom prijave također je bitno da sustav ne javlja odvojeno greške krivog upisanog korisničkog imena i krivo upisane lozinke. Ukoliko odvojeno šaljemo greške, napadač može skupiti listu korisničkih imena koje postoje. Kao posljedica toga *brute-force* napad može biti više efikasniji. Bolja praksa je slanje nekih generičkih poruka greške kako bi dali što manje bitnih informacija prilikom napada. Ista stvar vrijedi i prilikom registracije, no u većini slučajeva dovoljno je postaviti *Captcha* test. Napadač za pohranu liste korisničkih imena također može stvoriti mapu stranica korisničkog profila. Ukoliko takve stranice postoje, veoma je bitno da se u URL-u stranice nalazi korisničko ime, a u slučaju da je profil sakriven, greška skrivenih i ne postojećih profila trebala bi biti ista.

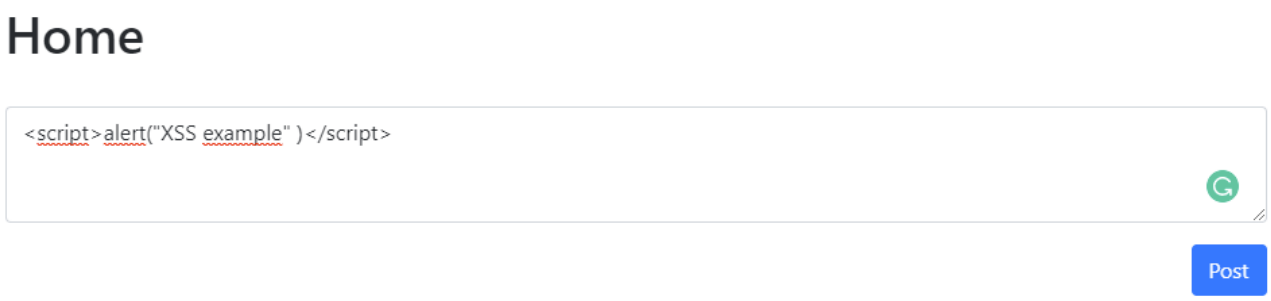
Ovisno o vrsti aplikacije, može se koristiti registracija korisnika pomoću vanjskih sustava kao što su *Facebook, Google Sign In* i *LinkedIn.* Takva vrsta registracije je veoma jednostavna i bezbrižna za korisnika. Proces registracije odvija se u par koraka i nakon uspješne registracije nije potrebno zapamtiti loziku. Veoma je bitno da sustav sam po sebi prilikom registracije ima dobro definirane kriterije prema kojima se određuju lozinke. Ako web aplikacija sadrži osjetljive podatke, dobra je praksa odrediti duljinu lozinke na najmanje 8 znakova, kombiniranjem malih i velikih slova, znakova i brojeva. Postavljanje kriterija sigurnijih lozinki dodatno povećava kompleksnost potrebnu za uspješno izvršavanje *brute-force* napada. Neke aplikacije zaključavaju prijavu korisnika sve dok administracija ne odobri ponovnu prijavu. U slučaju da je korisnička lozinka ugrožena, aplikacije za povećanu sigurnost prilikom prijave u sustav koriste i 2FA (*Two Factor Authentication*). Korištenjem 2FA za prijavu u aplikaciju nije potrebna samo lozinka, nego i neka dodatna informacija za koju smo sigurni da je posjeduje samo osoba koja je izradila račun. Za dodatnu sigurnost ove prijave šalje se pristupni kod sa pretpostavkom da će samo korisnik imat pristup tom kodu. Kod može biti poslan na e-mail adresu korisnika, slanjem poruke na mobitel ili korištenjem *Google 2-Step Verification-a.* Veoma je bitno da se *2FA* prijava omogući na kratko vrijeme, jer suprotnom ova metoda prijave nije veoma efektivna i može se probiti.

5.4.2. XSS – Cross-site scripting

Cross site scripting (XSS) je vrsta napada kojom napadač ubacuje zlonamjerne skripte unutar web stranice. Skripte se šalju drugim korisnicima čiji preglednici nisu u stanju prepoznati da li se skripta treba izvršiti ili ne. Postoje 3 vrste *cross site scripting* napada, a to su pohranjeni XSS (*Stored XSS*), reflektirani XSS (*Reflected XSS*) i XSS napadi temeljeni na DOM-u.

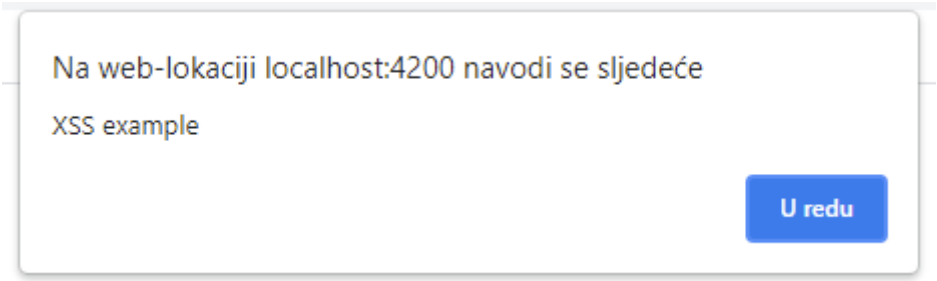
Stored XSS napadi izvršavaju se spremanjem zlonamjernih skripti u bazu podataka. Te skripte se kasnije mogu učitati u preglednicima drugih korisnika. Za ovu vrstu napada je potrebno ispuniti jedno polje unosa (*input field*), nakon čega je potrebno da se ti podaci spreme u bazu podataka te se ponovnim posjećivanjem stranice prikazuju.

Princip rada naše web aplikacije odgovara tom profilu. Unosom skripte u polje objave sadržaja, možemo spremiti skriptu koja će se učitavati svim korisničkim preglednicima.



Slika 37: Primjer ubacivanja skripti

Kada objavimo sadržaj sa slike, u svakom pregledniku koji posjećuje ovu web stranicu izvršiti će se skripta koja će otvoriti prozor u kojem piše *XSS example*. Sljedeća slika prikazuje djelovanje skripte.



Slika 38: Primjer izvršavanja cross site scripting napada

Pravi napad *cross site skripting-a* bio bi korišten za stjecanje osjetljivih podataka, kao što su *session tokeni* i druge bitne informacije pohranjene u pregledniku korisnika.

Za razliku od *Stored XSS*, skripte *Reflected XSS-a* nisu pohranjene u bazu podataka. Za ovaj napad je potrebno da se sadržaj polja za unos nakon objave ponovno vrati na stranicu. Ovo je najčešće slučaj kod vrsti tražilica koje za svoj unos šalju poziv pohranom podataka u parametrima stranice, te nakon izvršavanja poziva prikazuju podatke iz tih parametara. Izvršavanje ove vrste napada bazira se na slanju poveznice koja sadrži zlonamjernu skriptu koja će se pokrenuti prilikom učitavanja stranice. Napad ovisi o korisniku, posjećivanje poveznice aktivirati će skriptu.

DOM XSS vrlo je sličan *Reflected XSS-u* na način da se najčešće izvršava ubacivanjem skripti u poveznice. Za razliku od *Reflected XSS-a, DOM cross site scripting* ne šalje poziv serveru, nego se skripte izvršavaju direktno u pregledniku, koristeći JavaScript. Ovu vrstu napada teško je pronaći i nije toliko učestala kao ostali napadi.

XSS napadi izvršavaju se ubacivanjem JavaScript koda. Napad *cross site scripting-om* omogućava širenje „crva" (*worms*) na stranicama društvenih mreža, otimanje sesije, krađu identiteta, *DDoS* napade i krađu osjetljivih podataka kao što su lozinke. Za obranu ove vrste napada, potrebno je izbjeći sav dinamički sadržaj koji dolazi iz baze podataka na način da ga preglednik tretira kao sadržaj HTML-a, umjesto da ga tretira kao neobrađeni dio HTML-a. Većina modernih *framework-a* automatski izbjegava dinamički sadržaj. Izbjegavanjem dinamičkog sadržaja na ovaj način, zaustavljamo zlonamjerni kod od njegovog izvršavanja. To se najčešće čini zamjenom specifičnih simbola u kodirane HTML entitete (npr. „<" se pretvara u „&#60" i „>" se pretvori u „&#62"). Ukoliko web stranica podržava unos generiranja HTML-a, sadržaj takvih unosa potrebno je provjeriti i ukloniti skripte. Moderni preglednici posjeduju opciju postavljanja *Content-Security Policy* zaglavlja. Ta politika dopušta vlasniku web stranice da kontrolira od gdje se učitava i izvršava JavaScript kod. Postavljanjem ove politike u zaglavlje odgovora na poziv stranice, preglednik možemo kontrolirati u smislu da nikad ne izvršava inline JavaScript kod i da se skripte stranice mogu učitavati samo iz definiranih domena. Ova politika može se postaviti i kao sadržaj <meta> oznake.

```
Content-Security-Policy: script-src 'self' https://simon-kosmrl-diplomski.xyz
```

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self' https://simon-kosmrl-diplomski.xyz">
```

5.4.3. Krađa tokena

Najveći problem web aplikacija je prepoznavanje korisnika i utvrđivanje da li je neki korisnik onaj što tvrdi da je. Za autentifikaciju korisnika u našoj aplikaciji koriste se JWT (JSON Web Tokeni). Prilikom uspješne prijave, generiramo token i pohranjujemo ga u lokalno skladište korisnika (*local storage*). Spremanje podataka u lokalno skladište nije sigurno iz razloga što se njemu može pristupiti korištenjem JavaScripta. Izvršavanjem *XSS* napada na korisnika, informacije u lokalnom skladištu mogu se lako izvući. Bolja opcija spremanja tokena je korištenjem kolačića (*Cookies*). Spremanje kolačića preporučuje se postavljanjem *secure* opcije, koja šalje kolačić samo preko SSL-a, te *http-only* opacije, koja sprječava JavaScript od čitanja kolačića. Postavljanjem *SameSite* opcije aplikaciju štitimo i od *Cross-site request forgery* ranjivosti. *Cross-site request forgery* je vrsta

napada kojom se validnim tokenima poziv prema serveru može uputiti iz bilo koje domene. To može prouzročiti da se sadržaj aplikacije prikazuje i na drugim web stranicama. Najprije ćemo opisati kako JSON Web Tokeni funkcioniraju i spomenuti neke od njegovih ranjivosti, te ćemo nakon toga pokušati još više zaštititi web aplikaciju u slučaju da je neki od tokena ukraden.

JSON Web Tokeni sastavnjeni su od 3 dijela, a to su zaglavlje (*header*), sadržaj (*payload*), i potpis (*signature*). Zaglavnje tokena tokena prestavlja base64 format korištenog alogritma za digitalno potpisivanje samog tokena. U ovoj web aplikaciji korištena je *jsonwebtoken* JavaScript biblioteka za generiranje *JWT-a*. Biblioteka podržava mnoge algoritme potpisivanja tokena, od kojih se najčešće koristi HS256 algoritam. Primjer zaglavlja JSON Web Tokena predstavljen je sljedećim kodom:

```
header = '{"alg":"HS256","typ":"JWT"}'
```

Drugi dio tokena, odnosno sadržaj tokena, je base64 format sadržaja koji posjeduje token. Sadržaj tokena je JSON oblika.

```
payload = {
     "identification": "611fec3f3cd0780012ac3d1c",
     "username": "TestUser1",
     "iat": 1630342754,
     "exp": 1630429154
}
```

Zadnji dio predstavlja potpis prethodnog sadržaja tokena. Potpis je generiran algoritmom predstavljenom u zaglavlju tokena:

```
key = 'secretkey'
unsignedToken = encodeBase64(header) + '.' +
encodeBase64(payload)
signature = HMAC-SHA256(key, unsignedToken)
```

Validan *JSON Web Token* predstavlja spoj svih dijelova tokena koji su odvojeni točkama:

```
token = encodeBase64(header) + '.' + encodeBase64(payload) + '.' +
encodeBase64(signature)

# token :

EyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZGVudGlmaWNhdGlvbiI6IjYxM
WZlYzNmM2NkMDc4MDAxMmFjM2QxYyIsInVzZXJuYW1lIjoiVGVzdFVzZXIxIiwiaWF
0IjoxNjMwMzQyNzU0LCJleHAiOjE2MzA0MjkxNTR9.aGPPSINXy6Qj_m33C-
froowFxPEBURo5A5dHg7hnwTs
```

JSON Web Tokeni štite nas od manipulacije sadržaja tokena, no vrijedno je spomenuti da je se taj base64 sadržaj može lako pročitati. Stoga je veoma važno da se u sadržaj tokena ne spremaju bitni podaci. U našem primjeru, sadržaj tokena posjeduje *ID* i korisničko ime korisnika. Ovo je loša praksa jer se iz tokena mogu pročitati osjetljive informacije. Polje „identification" je *MongoDB-ov* način spremanja *ObjectID-a*, stoga možemo zaključiti da aplikacija koristi baš tu bazu za pohranu podataka korisnika. Ovo može kasnije dobro poslužiti jer znamo da je svaki sljedeći *ObjectID*, inkrement predhodnog *ObjectID-a.* Bolja praksa korištenja ove metode autentifikacije bila bi pohrana nasumičnog generiranog sadržaja, koji klijentu nema nikakvo značenje, no naša aplikacija bi trebala biti sposobna raspoznati o kojem korisniku je riječ.

Jedna od mana koju može posjedovati *JSON Web Token* je korištenje algoritma „*None*". Tokeni ne trebaju biti potpisani ukoliko se koristi „None" algoritam u zaglavlju tokena. Veoma je važno da web aplikacija ne dopušta validaciju tokena bez njegovog potpisa. U suprotnom sadržaj tokena može se promjeniti te bi tada napadači stekli pristup svim računima korisnika. Najbolja praksa je korištenje *HS256* ili *HS512* algoritama potpisa. Za potpis, preporuka je koristiti ključ iste ili veće veličine kao i rezultat algoritma sažimanja (256 bitova za HS256). Ključevi manjih veličina ranjivi su na *brute-force* napade.

Kako bi umanjili štetu ukoliko je neki od tokena korisnika ukraden, važno je postaviti datum isteka tokena. Kraći datum valjanosti tokena može uveliko umanjiti štetu koja može biti nanesena. Korištenjem sesije (eng. *session)* možemo dodatno osigurati web aplikaciju. *Session* je još jedan od načina na koji možemo autentificirati korisnike. *JSON Web Tokene* nije potrebno spremiti u bazu podataka kako bi potvrdili njihovu valjanost, za razliku od *session-a* kojem je potrebna neka vrsta baze kako bi utvrdili valjanost *session ID-a*. Ukoliko se valjani token ukrade, moguće je pristupiti računu korisnika iz bilo kojeg drugog preglednika. Stoga korištenjem *session-a* možemo spremiti neke od podataka kojima se naš korisnik primarno prijavio u aplikaciju. Ti podaci specifični su za korisnika i povezani su sa tokenom. Time ćemo znati o kojem je korisniku riječ, a pohranjene karakteristike poslužiti će za prepoznavanje valjanosti poziva. Korištenjem ovih praksi, dodatno osiguravamo autentifikaciju korisnika.

5.5. Ostali napadi na web aplikacije

5.5.1. SQL Injection/NoSQL Injection

Napadi *SQL* i *NoSQL Injectiona* iskorištavaju ranjivosti aplikacijskog koda. Ti napadi izvađaju se dinamičkim ubacivanjem komandi upita u bazu podataka. Svrha ovog napada je manipulirati rezultat kojeg vraća upit za bazu. Primjer ovog napada na našoj aplikaciji može se izvršiti prilikom

slanja poziva za prijavu. Moguće je manipulirati poziv prijave na način da se umjesto teksta korisničkog imena šalje određeni regularni izraz, koji će dohvatiti jednog korisnika za kojeg ne trebamo znati korisničko ime. U nastavku nalazi se kod aplikacije i primjer regularnog izraza koji će pronaći prvog korisnika čije ime počinje sa „Test”.

```
const {username, password, keepLogin} = req.body; // username =
/Test/
if(!username || !password)
     return res.send({success: false, error: "Wrong data"});

let user = await User.findOne<User>({username});
```

Ukoliko znamo koju vrstu podataka očekujemo, dobro je provjeriti da li je zaprimljen dobar tip podataka te u slučaju da nije odbijamo poziv ili sami promijenimo tip podataka (npr. korištenjem toString() JavaScript funkcije koja stvara tekst).

5.5.2. Clickjacking i Phishing napad

Clickjacking je vrsta napada koja iskorištava mogućnost implementiranja jedne web stranice unutar druge stranice koristeći *iframe* oznaku HTML-a. Takvi napadi iskorištavaju popularan sadržaj stranica, te skrivenim elementima stranice pokreću radnju koju korisnik nije namjeravao izvršiti. Ovaj napad ne ugrožava direktno sigurnost web aplikacije, no ugrožava korisnike. Postavljanjem *X-Frame-Options* ili već spomenutog *Content-Security-Policy* zaglavlja, pregledniku dajemo do znanja dali se web stranica može učitati pomoću *frame*, *iframe* ili *object* oznaka unutar drugih web stranica. Izgled web stranice može se kopirati, te samim time korisnik može imati osjećaj da posjećuje poznatu stranicu. Kopiranje izgleda stranice prijave, česti je oblik napada pri kojem se podaci uneseni od strane korisnik pohranjuju. Time se mogu steći podaci prijave za pravu web aplikaciju. Ovaj napad se naziva *Phishing* i može izbjeći podsjećanjem korisnika da provjeri domenu na kojoj se nalazi.

5.5.3. Prijenos zlonamjernih datoteka

Ukoliko web aplikacija posjeduje mogućnost prijenosa datoteka, napadači mogu prenijeti zlonamjerne skripte. Ako se te skripte mogu pokrenuti na serveru može doći do velike štete. Primjer takvog napad je prijenos PHP skripte kojom se pokreću cmd komande na serveru. Napadači u tom slučaju stječu pristup konzoli, što može prouzročiti mnogobrojne probleme.

5.6. Praćenje ponašanja aplikacije

Bilježenje (eng. *Logging*) i praćenje sustava (eng. *Monitoring*) vrlo je važno iz razloga što saznajemo što naša aplikacija radi i koje procese izvršava. Ispisivanjem informacija u konzoli aplikacije ili spremanjem zabilješki (eng. *logs*) u bazu podataka daje nam mnoga saznanja o aplikaciji. Na ovaj način možemo saznati greške aplikacije i način na koji korisnici komuniciraju sa njom. U slučaju napada možemo izbaciti upozorenje, tragovi napada ostati će zapisani. Standard je da se bilješki označuju jednom od 4 vrste poruka, a to su *DEBUG*, *INFO*, *WARNING* i *ERROR*. Osim vrste bilješke, dobro je ispisati i vrijeme ispisa. Ispis vremena omogućuje nam da proučimo stanje resursa sustava za vrijeme ispisa poruke. Također, dobro je ispisati ime datoteke i liniju koda koja je ispisala pojedinu bilješku. Kako aplikacija može ispisati mnogo bilješki, ove informacije su dobro došle prilikom filtriranja. Prilikom ispisa bilješki, važno je da ne ispisujemo osjetljive podatke kao što su lozinke, ključevi šifriranja, podaci o bazama i osobni podaci korisnika. Stvari koje je poželjno bilježiti su:

- URL-ove sa IP-adresom i kodom statusa odgovora
- Radnje pokrenute od strane korisnika
- Pokušaje prijave (uspješne i neuspješne), odbijanje tokena i sesije
- Pozive na druge servise izvan aplikacije
- Prihvaćanje pravila privatnosti i pravila korištenja
- Informacije o pokretanju i gašenju
- Greške aplikacije i događaje sustava

Ove bilješke trebaju biti osigurane, samo admini trebaju imat pristup njima. Osim aplikacijskih bilješki, web servisi kao *Nginx* bilježe promet mreže te nam daju na uvit ponašanje prometa aplikacije. Kako bi na vrijeme saznali ako je nešto pošlo po zlu, dobro je imati postavljene sustave praćenja koji nas obavještavaju o problemima.

6. Zaključak

Struktura aplikacija mikroservisa može biti veoma kompleksna. Aplikacija izrađena kao praktični dio diplomskog rada osnovni je primjer mikroservisa. Dodavanjem novih mikroservisa neće se utjecati na dosadašnji rad aplikacije. Razvojna tehnologija novih mikroservisa može se razlikovati od dosadašnje tehnologije ove web aplikacije, što tu činjenicu čini jednom od najvećih prednosti mikroservisa. Korištenjem orkestratora Kubernetes omogučeno nam je jednostavno i praktično skaliranje web aplikacija. Razvijena aplikacija postavljena je u pogon i može joj se pristupiti putem interneta.

Nad aplikacijom je izvršeno osnovno testiranje sigurnosti prilikom kojeg su pronađene neke od mana sustava. Penetracijsko testiranje odvija se u nekoliko faza: prikupljanje podataka, skeniranje, izvršavanje napada, zadržavanje pristupa i prekrivanje tragova. Postoje mnogi napadi koji prijete web aplikacijama, opis postupka izvršavanja nekih od napada i pritom korišteni alati iznesi su u ovom diplomskom radu. Sigurna komunikacija između klijenta i poslužitelja bazira se na šifriranju podataka. Ovo čini sastavni dio sigurnosti web aplikacija. Razvojem informacijske tehnologije, mnogi uređaji su dobili pristup internetu. Kao posljedica toga, napadi uskraćivanjem resursa aplikacije postali su sve opasniji. Za većinu napada može se spremiti plan obrane, a neke od napada moguće je totalno uskratiti korištenjem dobrih praksi prilikom razvoja web aplikacije. Praćenje stanja sustava i bilježenje događaja unutar aplikacije veoma je bitno i može pripomoći pri otkrivanju napada na web aplikacije. Rezultat testiranja sigurnosti, web aplikacije izrađene uz ovaj diplomski rad, upućuje na važnost pružanja sigurne usluge. Glavni ciljevi informacijske sigurnosti su povjerljivost, integritet te dostupnost sustava, mreže i podataka.

7. Literatura

[1] “2020-01_Attacking_and_Securing_JWT.pdf.” Accessed: Sep. 01, 2021. [Online]. Available: https://owasp.org/www-chapter-vancouver/assets/presentations/2020-01_Attacking_and_Securing_JWT.pdf

[2] “2020-01_Attacking_and_Securing_JWT.pdf.” Accessed: Sep. 01, 2021. [Online]. Available: https://owasp.org/www-chapter-vancouver/assets/presentations/2020-01_Attacking_and_Securing_JWT.pdf

[3] “Angular - Introduction to the Angular Docs.” https://angular.io/docs (accessed Sep. 01, 2021).

[4] “caplets/hstshijack at master · bettercap/caplets,” *GitHub*. https://github.com/bettercap/caplets (accessed Sep. 01, 2021).

[5] “Critical vulnerabilities in JSON Web Token libraries,” *Auth0 - Blog*. https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/ (accessed Sep. 01, 2021).

[6] “Cross Site Scripting (XSS) Software Attack | OWASP Foundation.” https://owasp.org/www-community/attacks/xss/ (accessed Sep. 01, 2021).

[7] “Docker overview,” *Docker Documentation*, Aug. 31, 2021. https://docs.docker.com/get-started/overview/ (accessed Sep. 01, 2021).

[8] “DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf.” Accessed: Sep. 01, 2021. [Online]. Available: https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf

[9] “Express - Node.js web application framework.” https://expressjs.com/ (accessed Sep. 01, 2021).

[10] “Famous DDoS attacks | Biggest DDoS attacks,” *Cloudflare*. https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/ (accessed Sep. 01, 2021).

[11] “Hacker,” *Wikipedia*. May 16, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hacker&oldid=1023461039

[12] “HTTP Strict Transport Security,” *Wikipedia*. Aug. 25, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=HTTP_Strict_Transport_Security&oldid=1040572075

[13] “Introduction :: bettercap.” https://www.bettercap.org/intro/ (accessed Sep. 01, 2021).

[14] “Introduction to Node.js,” *Introduction to Node.js*. https://nodejs.dev/learn (accessed Sep. 01, 2021).

[15] “Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution,” *Kali Linux*. https://www.kali.org/ (accessed Sep. 01, 2021).

[16] “Koja je razlika između web dizajnera i web developera,” *Safran Design | Wordpress Development*, Aug. 05, 2018. https://safrandesign.hr/razlika-izmedu-web-dizajnera-i-web-developera/ (accessed Sep. 01, 2021).

[17] “Kubernetes Best Practices: Security.” https://www.altostack.io/company/blog/kubernetes-best-

practices-security/ (accessed Sep. 01, 2021).
[18] “Kubernetes Documentation,” *Kubernetes*. https://kubernetes.io/docs/home/ (accessed Sep. 01, 2021).
[19] L. Baird, *leebaird/discover*. 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://github.com/leebaird/discover
[20] “Load Balancer.” https://www.f5.com/services/resources/glossary/load-balancer (accessed Sep. 01, 2021).
[21] “Logging and Monitoring,” *Hacksplaining*. https://www.hacksplaining.com/prevention/logging-and-monitoring (accessed Sep. 01, 2021).
[22] “Looking Beyond the Hype: Is Modular Monolithic Software Architecture Really Dead? | by Md Kamaruzzaman | Towards Data Science.” https://towardsdatascience.com/looking-beyond-the-hype-is-modular-monolithic-software-architecture-really-dead-e386191610f8 (accessed Sep. 01, 2021).
[23] “Metasploit | Penetration Testing Software, Pen Testing Security,” *Metasploit*. https://www.metasploit.com/ (accessed Sep. 01, 2021).
[24] “Microservices vs Monolith: which architecture is the best choice for your business?” https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (accessed Sep. 01, 2021).
[25] “Microservices vs Monolithic architecture,” *MuleSoft*. https://www.mulesoft.com/resources/api/microservices-vs-monolithic (accessed Sep. 01, 2021).
[26] “Modular programming,” *Wikipedia*. Jul. 17, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Modular_programming&oldid=1034099174
[27] “Nmap: the Network Mapper - Free Security Scanner.” https://nmap.org/ (accessed Sep. 01, 2021).
[28] “Offensive Security’s Exploit Database Archive.” https://www.exploit-db.com/ (accessed Sep. 01, 2021).
[29] A. Team, “Phases of Penetration Testing,” *All About Testing*, Mar. 17, 2018. https://allabouttesting.org/phases-of-penetration-testing/ (accessed Sep. 01, 2021).
[30] “Protecting Against SQL Injection,” *Hacksplaining*. https://www.hacksplaining.com/prevention/sql-injection (accessed Sep. 01, 2021).
[31] “Protecting Your File Uploads,” *Hacksplaining*. https://www.hacksplaining.com/prevention/file-upload (accessed Sep. 01, 2021).
[32] “Protecting Your Users Against Clickjacking,” *Hacksplaining*. https://www.hacksplaining.com/prevention/click-jacking (accessed Sep. 01, 2021).
[33] “Protecting Your Users Against Cross-site Scripting,” *Hacksplaining*. https://www.hacksplaining.com/prevention/xss-stored (accessed Sep. 01, 2021).
[34] “reCAPTCHA,” *Google Developers*. https://developers.google.com/recaptcha?hl=hr (accessed Sep. 01, 2021).
[35] “Secure Treatment of Passwords,” *Hacksplaining*.

https://www.hacksplaining.com/prevention/password-mismanagement (accessed Sep. 01, 2021).
[36] “Security Features and Setup — MongoDB Atlas,” *https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt*. https://docs.atlas.mongodb.com/setup-cluster-security/ (accessed Sep. 01, 2021).
[37] A. Hauser, “The lesser known XSS Technique.” https://www.scip.ch/en/?labs.20171214 (accessed Sep. 01, 2021).
[38] “The most popular database for modern apps,” *MongoDB*. https://www.mongodb.com (accessed Sep. 01, 2021).
[39] “The Seven Types of Security Testing | test IO.” https://test.io/blog/the-seven-types-of-security-testing/ (accessed Sep. 01, 2021).
[40] “Three Common Types of DDoS Attacks?” https://www.thousandeyes.com/blog/three-types-ddos-attacks (accessed Sep. 01, 2021).
[41] “TLS 1.3 Handshake: Taking a Closer Look,” *Hashed Out by The SSL Store™*, Mar. 20, 2018. https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/ (accessed Sep. 01, 2021).
[42] “Transport Layer Security,” *Wikipedia*. Aug. 31, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Transport_Layer_Security&oldid=1041611876
[43] “Types of DDoS attacks explained.” https://cybersecurity.att.com/blogs/security-essentials/types-of-ddos-attacks-explained (accessed Sep. 01, 2021).
[44] “Types of XSS | OWASP.” https://owasp.org/www-community/Types_of_Cross-Site_Scripting (accessed Sep. 01, 2021).
[45] “TypeScript,” *Wikipedia*. Sep. 01, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=TypeScript&oldid=1041710710
[46] J. P. Updated, “What is a Brute Force Attack? Definition | Varonis,” *Inside Out Security*, Oct. 16, 2018. https://www.varonis.com/blog/brute-force-attack/ (accessed Sep. 01, 2021).
[47] “What is a distributed denial-of-service (DDoS) attack?,” *Cloudflare*. https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (accessed Sep. 01, 2021).
[48] J. P. Updated, “What is Metasploit? The Beginner’s Guide - Varonis,” *Inside Out Security*, Aug. 14, 2019. https://www.varonis.com/blog/what-is-metasploit/ (accessed Sep. 01, 2021).
[49] “What is Penetration Testing | Step-By-Step Process & Methods | Imperva,” *Learning Center*. https://www.imperva.com/learn/application-security/penetration-testing/ (accessed Sep. 01, 2021).
[50] “What Is Phishing? Examples and Phishing Quiz,” *Cisco*. https://www.cisco.com/c/en/us/products/security/email-security/what-is-phishing.html (accessed Sep. 01, 2021).
[51] T. Hamilton, “What is Security Testing? Types with Example.” https://www.guru99.com/what-is-security-testing.html (accessed Sep. 01, 2021).

8. Popis slika

Table of Figures