Primjena sigurnosnih mehanizama u razvoju mrežnih aplikacija za automobilsku industriju

Živković, Josip

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

Sveučilišni studij

PRIMJENA SIGURNOSNIH MEHANIZAMA U RAZVOJU MREŽNIH APLIKACIJA ZA AUTOMOBILSKU INDUSTRIJU

Diplomski rad

Josip Živković

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 15.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Josip Živković
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D-57ARK, 11.10.2020.
OIB studenta:	30052009550
Mentor:	Prof. dr. sc. Marijan Herceg
Sumentor:	,
Sumentor iz tvrtke:	Davor Kedačić
Predsjednik Povjerenstva:	Doc. dr. sc. Denis Vranješ
Član Povjerenstva 1:	Prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	izv. prof. dr.sc. Josip Job
Naslov diplomskog rada:	Primjena sigurnosnih mehanizama u razvoju mrežnih aplikacija za automobilsku industriju
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	Kod razvoja programske podrške općenito, a posebno kod one koja se koristi u mrežnom okruženju i predviđena je za upotrebu od strane većeg broja različitih korisnika neophodno je od samog početka voditi brigu o svim aspektima sigurnosti. Zadatak ovog rada je primijeniti sigurnosne mehanizme u razvoju mrežne aplikacije za provođenje udaljenog ispitivanja programske podrške u automobilskoj industriji. Potrebno je omogućiti sigurno autenticiranje i autoriziranje korisnika mrežne aplikacije, definirati i implementirati potrebne mehanizme za prikaz sadržaja na koje korisnik, prema pripadnosti određenoj korisničkoj grupi, ima pravo. Osim međusobne komunikacije između različitih korisnika, komunikacija se vrši i između testnih stanica i
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	15.09.2022.

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 26.09.2022.

Ime i prezime studenta:	Josip Živković
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-57ARK, 11.10.2020.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena sigurnosnih mehanizama u razvoju mrežnih aplikacija za automobilsku industriju**

izrađen pod vodstvom mentora Prof. dr. sc. Marijan Herceg

i sumentora ,

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

Potpis studenta:

SADRŽAJ

1. UVOD

Testiranje softvera predstavlja metodu kojom se nastoji provjeriti odgovara li stvoreni softverski proizvod zahtjevima i očekivanjima koja su bila postavljena i dogovorena u početku. Provođenje testiranja je bitno kako bi se osiguralo da softverski proizvod nema nedostataka. Testiranje softvera podrazumijeva izvođenje komponenti softvera korištenjem ručnih ili automatiziranih alata za procjenu jednog ili više svojstava od interesa. Sama svrha testiranja softvera je identificirati pogreške, nedostatke ili zahtjeve koji nedostaju u odnosu na dogovorene zahtjeve [1]. Kako softver pišu i stvaraju ljudi nerealno je očekivati da u razvijenom softveru neće postojati pogreške i problemi, zato je provođenje testiranja softvera nužno. Testiranje softvera je bitno jer ukoliko postoje bilo kakve pogreške u softveru, cilj je da se te pogreške identificiraju u što ranijoj fazi razvoja, odnosno prije isporuke softverskog proizvoda jer se na taj način dodatno smanjuju troškovi samog testiranja. Pravilno testiran softver pruža veću sigurnost, osigurava pouzdanost i visoke performanse što rezultira uštedom vremena, isplativošću i većim zadovoljstvom krajnjih korisnika. Da je testiranje softvera bitno pokazuju i neki primjeri iz prakse u kojima je vidljivo da softverske greške ili zanemarivanje kvalitetnog testiranja može biti vrlo skupo i imati opasne posljedice. U travnju 2015. godine, *Bloombergov* terminal u Londonu je pao zbog softverske pogreške što je utjecalo na više od 300.000 trgovaca i stvorilo velike financijske probleme. Sljedeći primjer je *Nissan* koji je bio prisiljen povući više od milijun automobila s tržišta zbog softverskog kvara u senzorskim detektorima zračnih jastuka. Postoje primjeri u kojima su i izgubljeni ljudski životi. *Airbus A300 China Airlinesa* srušio se zbog softverske pogreške 26.4.1994. prilikom čega je smrtno stradalo 264 osobe. 1985. godine kanadski uređaj za liječenje *Therac-25* pokvario se zbog programske pogreške i isporučio smrtonosnu dozu zračenja pacijentima prilikom čega su tri osobe smrtno stradale i tri su bile kritično ozlijeđene [1]. Iz navedenih primjera jasno je vidljivo da se testiranje softvera, a pogotovo onog softvera koji se koristi u automobilskoj industriji i liječenju ljudi treba shvatiti jako ozbiljno. Samo testiranje softvera može se klasificirati u tri kategorije: funkcionalno testiranje, nefunkcionalno testiranje ili testiranje performansi te održavanje. Funkcionalno testiranje predstavlja vrstu testiranja softvera kojim se sustav provjerava u odnosu na funkcionalne zahtjeve. Svrha funkcionalnih testova je testirati svaku funkciju aplikacije na način da se pruži odgovarajući ulaz te se provjerava izlaz u odnosu na zahtjeve. Ovakvo testiranje provjerava korisničko sučelje, bazu podataka, sigurnost, API-je te komunikaciju između klijenta i poslužitelja. Najpoznatiji primjeri funkcionalnog testiranja su: *unit* testiranje, *smoke* testiranje, integracijsko testiranje te *white* i *black box* testiranje [2]. Nefunkcionalno testiranje predstavlja vrstu testiranja softvera za provjeru nefunkcionalnih aspekata (izvedba, pouzdanost, upotrebljivost itd.) aplikacije. Svrha nefunkcionalnog testiranja je odrediti spremnost sustava prema nefunkcionalnim parametrima. Jedan od najboljih primjera nefunkcionalnog testiranja je provjera koliko bi se ljudi moglo istovremeno prijaviti u aplikaciju. Neki od

primjera nefunkcionalnog testiranja su: testiranje performansi, testiranje opterećenja, testiranje otpornosti na stres, sigurnosno testiranje, penetracijsko testiranje, testiranje kompatibilnosti itd. [3]. Jednom kada je aplikacija napravljena i postavljena, ona može biti u procesu korištenja dugi niz godina. Česta je pojava da se sustav i okruženje tijekom tog vremena mijenjaju, ispravljaju i proširuju. Testiranje koje se provodi tijekom te faze naziva se održavanje. Održavanje se obično sastoji od dva dijela. Prvi dio se odnosi na testiranje promjena koje su napravljene zbog korekcija u sustavu ili ako je sustav proširen. Drugi dio održavanja pokrivaju regresijski testovi koji za cilj imaju dokazati da radovi na održavanju nisu utjecali na ostatak sustava [4]. U praksi postoji preko 150 vrsta testiranja, a broj se povećava iz dana u dan. Također, nisu sve vrste testiranja primjenjive na sve projekte, već ovise o tipu i veličini samog projekta, te je i to faktor o kojem treba voditi računa prije provođenja samog testiranja.

Glavni cilj ovog diplomskog rada je opisati i primijeniti sigurnosne mehanizme u razvoju mrežnih aplikacija. Diplomski rad strukturiran je u pet poglavlja. U uvodnom dijelu opisana je važnost testiranja softvera i načini provođenja testiranja. U drugom poglavlju opisane su postojeće metodologije i alati koji se koriste za ispitivanje sigurnosti softvera. Također su opisane i dobre prakse koje bi se trebale provoditi prilikom razvoja sigurnih mrežnih aplikacija. U trećem poglavlju opisane su najčešće prijetnje na mrežne aplikacije i načini prevencije tih napada. U trećem poglavlju opisana je i implementacija sigurnosnih mehanizama u *Django* radnom okviru prilikom izrade mrežne aplikacije. U četvrtom poglavlju obavljeno je testiranje funkcionalnosti implementiranih mehanizama kao i testiranje mrežne aplikacije pomoću alata za procjenu sigurnosti. U petom poglavlju dan je kratak zaključak o svemu obrađenom.

2. PREGLED POSTOJEĆIH RJEŠENJA ZA ISPITIVANJE SIGURNOSTI SOFTVERA

Sama svrha sigurnosnog testiranja mrežnih aplikacija je pronalaženje sigurnosnih nedostataka ili ranjivosti unutar aplikacije i njenog okruženja s ciljem dokumentiranja tih ranjivosti i pronalaženja načina na koje bi se te ranjivosti mogle popraviti i ukloniti. Postoji nekoliko tipova testnih metodologija. Tu spadaju sigurnosne revizije, procjena ranjivosti i penetracijski testovi. Svaka od metodologija ima različite opsege i ciljeve, te naravno prednosti i nedostatke. Metodologije testiranja mogu biti izvršene na različitim razinama automatizacije. Neka testiranja provode se potpuno automatizirano, dok su ostala testiranja izvedena manualno. Nakon što je testiranje izvršeno, sljedeći preporučeni korak je ispravljanje ranjivosti koje su uočene tijekom testiranja. Sljedeći korak, nakon što je sanacija grešaka dovršena, trebalo bi biti naknadno ispitivanje koje treba osigurati da su svi popravci uspješno obavljeni [5]. Svrha ovog poglavlja je opisati najčešće vrste sigurnosnih testiranja, metodologije i alate, opisati dobre prakse koje treba slijediti za siguran razvoj softvera te opisati razine automatizacije koje se koriste prilikom provođenja testova.

2.1. Vrste sigurnosnih testiranja

Sigurnosno testiranje predstavlja oblik nefunkcionalnog testiranja kojemu je glavni zadatak provjeriti postoje li prijetnje u softveru, te koji su rizici i ranjivosti. Dok je zadatak funkcionalnog testiranja provjera radi li softver ispravno, sigurnosno testiranje treba utvrditi je li softver dobro konfiguriran, dobro dizajniran i bez rizika. Sigurnosno testiranje usredotočeno je na hrpu elemenata kao što su: ranjivosti koje mogu izložiti sredstva napadačima, rizik koji nameću različite sigurnosne rupe, sredstva poput aplikacije ili infrastrukture kojima treba zaštita te sanacija sigurnosnih problema. Najčešće vrste sigurnosnih testiranja su: skeniranje ranjivosti, ispitivanje prodora ili penetracijsko testiranje, procjena rizika, revizija sigurnosti, pregled izvornog koda te procjena sigurnosnog stanja [6].

Skeniranje ranjivosti predstavlja automatizirani proces kojeg podjednako koriste sigurnosni inženjeri kao i napadači za prepoznavanje ranjivosti na mrežnoj aplikaciji ili mreži. Skeniranje ranjivosti uključuje testiranje ranjivosti ili slabosti koje mogu postojati u informacijskom sustavu. Skeniranje ranjivosti radi na principu da se sustav analizira u odnosu na poznate potpise ranjivosti. Rezultati samog testiranja najčešće se prikazuju u datoteci poznatoj i kao *izvješće o skeniranju ranjivosti* [7]. Skeniranje ranjivosti nadalje uključuje:

- Vanjsko skeniranje ranjivosti: Ova vrsta skeniranja koristi se za identifikaciju ranjivosti u dijelovima mreže koja su izložena internetu.
- Unutarnje skeniranje ranjivosti: Vrsta skeniranja koja se koristi za skeniranje područja mreže koja su rezervirana za internu upotrebu.

- Nenametljivo skeniranje ranjivosti: Predstavlja metodu koja utvrđuje potencijalne ranjivosti u mreži na temelju tragova okoline bez stvarnog iskorištavanja ranjivosti.
- Nametljivo skeniranje ranjivosti: U ovoj metodi napadač pokušava iskoristiti ranjivost kako bi odredio koliki rizik ta ranjivost predstavlja za mrežu [6].

Ispitivanje prodora ili penetracijsko testiranje predstavlja oblik sigurnosnog testiranja u kojem sigurnosni inženjeri simuliraju napade kako bi provjerili i pronašli ranjivosti prisutne u mrežnoj aplikaciji ili mreži. Penetracijsko testiranje doista nalikuje stvarnom hakiranju, no provodi se u sigurnim uvjetima i prema unaprijed određenim pravilima. Testiranje često provode timovi od jednog ili više ispitivača. Glavna prednost penetracijskog testiranja je dokaz rizika. No s druge strane vrijeme potrebno za provedbu penetracijskog testiranja je veliko te ne smanjuje rizik već samo potvrđuje postojanje rizika. Potencijalno bolje rješenje za većinu tvrtki bi bilo da novac ulažu u uklanjanje ranjivosti i jačanje infrastrukture umjesto da traže potvrdu o postojanju ranjivosti. No zbog izuzetno velike sposobnosti identificiranja skrivenih ranjivosti, penetracijsko testiranje predstavlja jednu od najpopularnijih vrsta sigurnosnog testiranja [5]. Sam proces penetracijskog testiranja podijeljen je u sedam faza:

- Priprema: U ovoj fazi testeri se savjetuju kako bi odredili cilj penetracijskog testa. Sam opseg testa također je definiran u ovoj fazi.
- Prikupljanje informacija: Testeri pokušavaju prikupiti što više informacija o ciljnoj mreži na način da koriste razne aktivne i pasivne tehnike.
- Otkriće: U ovoj fazi testeri skeniraju metu kako bi pronašli poznate ranjivosti.
- Analiza ranjivosti: Ranjivosti koje su identificirane u prethodnoj fazi se analiziraju i ocjenjuju u smislu ozbiljnosti i utjecaja.
- Iskorištavanje: Napadači iskorištavaju ranjivosti kako bi dobili pristup nakon čega pokušavaju eskalirati pristup. Ovo je faza u kojoj mogu odrediti koliki rizik određena ranjivost zaista predstavlja za sustav.
- Izvješće i preporuka: U ovoj fazi priprema se izvješće prethodnih faza. Sadrži popis ranjivosti, njihove rezultate te preporuke za sanaciju.
- Sanacija i ponovno skeniranje: U ovoj fazi tester surađuje s programerima kako bi uklonili ranjivosti i ponovno obavili skeniranje sustava kako bi potvrdili sigurnosni status [6].

Procjena rizika označava proces koji pomaže u prepoznavanju i ublažavanju potencijalnih prijetnji i ranjivosti u informacijskom sustavu i podacima. Jedna od funkcija je da pomaže u procjeni statusa informacijske sigurnosti u organizaciji te identificira područja rizika. Procjena rizika je sustavna i objektivna metoda procjene sigurnosti informacijskog sustava i identificiranja sigurnosnih rizika koji bi se mogli iskoristiti za nanošenje štete organizaciji [7]. Ovu vrstu sigurnosnog testiranja možemo podijeliti u četiri koraka:

- Identifikacija: Ova faza uključuje stvaranje popisa svih sredstava koji su kritični za mrežu, dijagnosticiranje podataka koje svako sredstvo prenosi ili pohranjuje te određivanje rizika za svako sredstvo
- Procjena: Imovina se provjerava na rizik u smislu iskorištavanja, utjecaja na poslovanje, potencijalan gubitak prihoda itd. Imovina se također procjenjuje u smislu kritičnosti za poslovanje kako bi se mogao odrediti prioritet.
- Ublažavanje: U ovom koraku vlasnici poduzeća zajedno sa stručnjacima za sigurnost planiraju određene mjere kako bi ublažili određene propuste.
- Prevencija: Nakon što se rizik ublaži, provode se daljnje preventivne sigurnosne mjere poput vatrozida [6].

Revizija sigurnosti ili sigurnosna revizija je dio djelatnosti sigurnosne procjene. Ona uključuje ispitivanje računalnih sustava, mreža ili softvera za provjeru učinkovitosti sigurnosnih mjera. Reviziju sigurnosti obično provodi tvrtka za procjenu ranjivosti i testiranje prodora ili se procjena može provesti interno. Sigurnosna revizija kombinira automatizirano skeniranje ranjivosti i manualno testiranje penetracije kako bi se stvorilo izvješće koje prikazuje uobičajene ali i rijetke skrivene ranjivosti mrežne aplikacije. Time se dobiva detaljno izvješće koje se sastoji od informacija o ranjivostima, njihovoj ocjeni i mogućem utjecaju na poslovanje. Izvješće nudi i detaljne smjernice kako određene ranjivosti popraviti. Izvješće o reviziji sigurnosti treba pomoći u procjeni sigurnosne spremnosti organizacije i otkrivanju područja koja su ranjiva na bilo kakve sigurnosne prijetnje. Nakon što se problemi riješe, obično se ponovno obavlja skeniranje kako bi se potvrdila sanacija ranjivosti u mrežnoj aplikaciji. Nakon što revizija prođe, pružatelj revizije izdaje potvrdu [6].

Pregled izvornog koda predstavlja proces testiranja izvornog koda aplikacije na sigurnosne nedostatke povezane s logikom, implementacijom specifikacija, stilskim smjernicama i ostalim aktivnostima. Pregled koda može se obaviti automatizirano ili ručnim pregledom koda. Najčešće se u praksi preferira kombinirani pristup. Automatizirani pregled koda brzo otkriva niz nedostataka tijekom životnog ciklusa razvoja softvera. Programeri često koriste DAST alate za pronalaženje i popravljanje ranjivosti unutar izvornog koda prije same provjere koda. Ručni pregled, kao i što samo ime kaže, predstavlja ručni pregled cijele baze koda. Ovakav pristup može poslužiti za otkrivanje nedostataka poput pogrešaka poslovne logike koje bi automatski pregledi mogli propustiti [6].

Procjena sigurnosnog stanja mreže određuje zdravlje i otpornost u suočavanju s potencijalnim prijetnjama kibernetičke sigurnosti. Glavna je zadaća ove vrste sigurnosnog testiranja opisati koliko je dobro mrežna aplikacija ili sama mreža opremljena za samoobranu. Tu se kombiniraju sve različite metodologije testiranja sigurnosti kako bi se provela sveobuhvatna procjena sigurnosti aplikacije i mreže. Glavni koraci uključeni u procjenu sigurnosnog stanja su: identifikacija i procjena vrijednosti kritične imovine, utvrđivanje

sigurnosnih rizika i izloženosti podataka, procjena trenutnih sigurnosnih mjera te planiranje boljeg ulaganja u sigurnosne mjere [6].

2.2. Pristupi testiranju sigurnosti softvera

Testiranje sigurnosti softvera predstavlja zadatak testiranja softvera prilikom čega se softver analizira na potencijalne ranjivosti i slabosti. Testiranje sigurnosti softvera se provodi pomoću tri različita pristupa: testiranje crne kutije, testiranje bijele kutije i testiranje sive kutije.

Sigurnosno testiranje crne kutije predstavlja metodu testiranja sigurnosti softvera prilikom kojeg testni inženjer nije svjestan internog rada mrežne aplikacije ili sustava koji se testira. Inženjer koji se bavi ovim testiranjem mora se osloniti na postojeću dokumentaciju, stručnost i unose programera i korisnika kako bi moga dizajnirati testove. Testni inženjer nema nikakvog znanja o unutrašnjosti sustava ili aplikacije koja se testira. Testiranje crne kutije predstavlja najčešći pristup testiranju sigurnosti softvera prilikom kojeg ispitivač osmišljava testove kao neinformirani napadač. Tester bi trebao predvidjeti potencijalne rizike na sustav te prilagoditi pisanje testova tim rizicima [8]. Glavna prednost ovakvog načina testiranja je u tome što su testovi koji se koriste nezavisni od implementacije softvera, odnosno temelje se isključivo na specifikacijama sustava, te se isti testovi mogu iskoristiti i kad dođe do promjena u implementaciji softvera. Prednosti su i u tome što se testiranje može odvijati paralelno s razvojem softvera, te što se testovi izvode s korisničke točke gledišta. Mana ovakvog testiranja je u tome što neke implementirane dijelove nije moguće pokriti testovima, te problem predstavlja i nejasna specifikacija softvera, u slučaju kada korisnik nije dobro definirao svoje potrebe, čime se sam dizajn testova dosta otežava.

Sigurnosno testiranje bijele kutije je suprotno testiranju crne kutije. Za razliku od testiranja crne kutije, prilikom kojeg tester ne zna ništa o sustavu kojeg testira, kod testiranja bijele kutije testeru je poznato unutarnje stanje sustava i struktura koda [7]. Model crne kutije fokusira se na to što softver radi, dok se model bijele kutije fokusira na to kako određeni softver radi. Sigurnosno testiranje bijele kutije se najčešće obavlja nakon metode crne kutije, a glavni cilj takvog testiranja je provjeriti kod, a ne specifikacije. Sigurnosno testiranje bijele kutije koriste i sami developeri, jer se svakako očekuje da je sve što je implementirano i testirano prije nego što se postavi na server. Testiranje bijele kutije naziva se još i testiranje staklene kutije jer tester vidi kroz „zidove“ koda.

Sigurnosno testiranje sive kutije predstavlja oblik sigurnosnog testiranja koje zahtijeva da se kombiniraju određena znanja i vještine. Predstavlja hibridni oblik testiranja u kojem se kombiniraju metoda crne i metoda bijele kutije. Kod sigurnosnog testiranja bijele kutije, testeru je poznato unutrašnje stanje sustava, dok kod sigurnosnog testiranja crne kutije nema pristup unutrašnjosti sustava. Sigurnosno testiranje sive kutije predstavlja kombinaciju tih metoda, tester ima određeno znanje, ali ne i svo znanje. Također, tester kod

ovakvog oblika testiranja ima pristup i određene privilegije unutar sustava, kao i znanje o unutarnjem dijelu mreže, što potencijalno uključuje dokumentaciju o dizajnu i arhitekturi sustava. Sa testerom se dijele samo ograničene informacije, poput podataka za prijavu. Koristeći dostupnu dokumentaciju, testeri mogu usmjeriti svoje napore kako bi procijenili dijelove sustava s najvećim rizikom od samog početka. Sigurnosno testiranje sive kutije korisno je za razumijevanje razine pristupa koju bi neki povlašteni korisnik mogao dobiti, te štete koju bi mogao prouzročiti u sustavu. Glavne prednosti ovakvog testiranja su u tome što se testiranje i dalje provodi sa stajališta korisnika ili napadača, a ne programera, što može pomoći u otkrivanju nedostataka koje su programeri propustili, te pošto testeri bolje razumiju ciljani sustav, mogu otkriti značajnije ranjivosti uz puno manje truda i troškova [7].

2.3. Razine automatizacije

Postoji nekoliko razina automatizacije na kojima se sigurnosni testovi mogu izvoditi. Tu spadaju potpuno automatizirano testiranje, manualno ili ručno testiranje te kombinacija automatiziranog i manualnog testiranja.

Potpuno automatizirano testiranje izvodi se pomoću alata koji su dizajnirani za autonomno pokretanje nakon što im se preda IP adresa i URL-ovi za testiranje. Prije pokretanja samog automatiziranog testiranja, na testeru je da provjeri je li meta koju pokušava testirati vidljiva. Ako se IP adresa i URL-ovi koji su u opsegu testiranja nalaze iza vatrozida, potrebno je od sigurnosne osobe zatražiti siguran pristup. Kvalitetni alati za automatizirano testiranje trebaju imati pristup bazama podataka, trenutnim ranjivostima i prijetnjama kako bi mogli kvalitetno testirati sustav i izbjeći lažne rezultate. Glavne prednosti automatiziranih testova su: brzi rad i stopostotna pokrivenost aplikacije, točan broj prijavljenih opasnosti, isplativost i dobivanje pravovremenih aktivnih informacija [5].

Manualno ili ručno testiranje predstavlja razinu testiranja koja ako je provedena od strane stručnjaka za sigurnost mrežnih aplikacija pruža najveću moguću dubinu i kvalitetu testiranja. Manualno testiranje je oblik testiranja u kojem tester traži ranjivosti korak po korak, te kad pronađe problem, istražuje dalje tu ranjivost kako bi spoznao veličinu problema i rizika kojeg ta ranjivost u sustavu predstavlja. Manualni testeri također koriste razne alate kako bi proveli većinu testiranja, no kako svaki alat ima ograničenja, česta je praksa da koriste barem dva alata kako bi smanjili mogućnost propuštanja ranjivosti. Manualno testiranje je podložno vremenskom i troškovnom ograničenju, te se zbog toga provodi samo na dijelovima mrežne aplikacije. Glavna prednost manualnog testiranja je u tome što ono može biti detaljnije od automatiziranog testiranja. Pošto manualno testiranje provode ljudski stručnjaci, oni mogu bolje razumjeti ranjivosti od automatiziranih alata za testiranje. Naravno postoje i neki nedostaci manualnog testiranja poput visoke cijene provođenja, opseg

testiranja je ograničen te broj instanci pojedine ranjivosti nije prijavljen. Prijavljene su samo potencijalne ranjivosti, dok je na testeru da pronađe sve instance te ranjivosti [5].

Kombinacija automatiziranog i manualnog testiranja je metoda koja možda i najbolje može dati procjenu sigurnosnih rizika mrežne aplikacije. Automatizirano testiranje može se obaviti na mjesečnoj razini kako bi se dobile određene informacije o sustavu po relativno niskoj cijeni. Manualno testiranje se može provoditi na tromjesečnoj ili godišnjoj razini kako bi se pronašle ranjivosti koje automatizirano testiranje nije bilo u stanju detektirati. Ovakvom kombinacijom automatiziranog i manualnog testiranja dobiva se najbolje iz oba svijeta. Mrežna aplikacija testirana je u punom opsegu te su dobivena redovita i pravovremena izvješća o ranjivostima [5].

2.4. Metode za siguran razvoj mrežnih aplikacija

Jedan od ključnih koraka kod razvoja mrežnih aplikacija kojeg je potrebno izvršiti u samom početku razvoja aplikacije je integracija sigurnosti. Time se osim vremena štede i troškovi razvoja, jer određene loše prakse izbjegavamo odmah od početka razvoja. Ovo potpoglavlje rezervirano je za opis dviju metoda koje se koriste za siguran razvoj mrežnih aplikacija. Prvo će biti opisan *Security Development Lifecycle* tvrtke *Microsoft*, a zatim *Comprehensive Lightweight Application Security Process* organizacije OWASP.

Microsoft SDL (engl. *Microsoft Security Development Lifecycle)* predstavlja proces razvoja softvera koji je temeljen na spiralnom modelu razvoja, kojeg je *Microsoft* predložio kao pomoć programerima kod stvaranja mrežnih aplikacija uz smanjenje sigurnosnih problema, rješavanje sigurnosnih ranjivosti pa čak i smanjenja troškova razvoja i održavanja. Sam proces podijeljen je u sedam faza među koje spadaju: obuka, zahtjevi, dizajn, implementacija, verifikacija, realizacija i odgovor. Faza obuke predstavlja fazu koja uključuje siguran dizajn, modeliranje prijetnji, sigurno kodiranje, sigurnosno testiranje i prakse vezane uz privatnost. Faza zahtjeva uključuje uspostavljanje sigurnosti i privatnosti koje krajnji korisnici zahtijevaju. U fazi dizajna razmatra se pitanje sigurnosti i privatnosti, potrebno je utvrditi zahtjeve na dizajn, analizirati moguću površinu napada te koristiti modeliranje mogućih prijetnji. Faza implementacije trebala bi koristiti odobrene alate i pružiti analizu performansi kako bi se provjerila funkcionalna ograničenja aplikacije. Verifikacija ima za cilj provesti dinamičku analizu i testiranje te provesti ponovnu analizu moguće površine napada. Faza izdavanja uključuje pregled svih sigurnosnih aktivnosti te izradu plana odgovora na incidente. Faza odgovora služi za provedbu plana odgovora na incidente koji je pripremljen tijekom faze izdavanja [9].

CLASP (engl. *OWASP Comprehensive Lightweight Application Security Process)* predstavlja još jednu od metoda sigurnog razvoja softvera. CLASP metoda se sastoji od aktivnosti koje bi razvojni inženjeri trebali provesti kako bi mogli razviti siguran softver. Glavna razlika CLASP metode u odnosu na Microsoft SDL metodu je u tome što se kod CLASP metode aktivnosti kategoriziraju po ulogama. Svatko tko radi na

razvoju softvera pomoću CLASP metode ima ulogu koja nosi određene aktivnosti koje ta osoba treba izvršiti. Uloge i aktivnosti možemo prikazati na sljedeći način:

- Projektni menadžer: Glavna zadaća mu je da uspostavi svijest o sigurnosti programa, treba pratiti sigurnost
- Specifikator zahtjeva: Treba identificirati politiku sigurnosti, opisati neke slučajeve nepravilnog korištenja mehanizama
- Arhitekt: Treba u suradnji sa specifikatorom zahtjeva identificirati resurse i granice povjerenja, identificirati uloge korisnika, navesti operativno okruženje, dokumentirati zahtjeve vezane za sigurnost
- Dizajner: Treba istražiti pojedina tehnološka rješenja, identificirati moguća područja napada, uključiti sigurnosne principe u dizajn, riješiti prijavljene sigurnosne probleme
- Implementator: Zadaće su mu da implementira sučelje ugovora, razradi politiku resursa i sigurnosnih tehnologija, da u suradnji sa dizajnerom i arhitektom radi na izradi sigurnosnog vodiča
- Tester: Treba identificirati i implementirati testove sigurnosti te verificirati sigurnosne atribute resursa
- Revizor sigurnosti: Modelira moguće prijetnje te se uz dizajnera i implementatora bavi kontrolom razine sigurnosti izvornog koda

CLASP metoda je rezultat dugogodišnjeg rada i istraživanja s ciljem stvaranja velikog skupa sigurnosnih zahtjeva na razvoj softvera [10].

2.5. Dobre prakse za razvoj sigurnih mrežnih aplikacija

Danas su mrežne aplikacije ključni aspekti u poslovanju i svakodnevnom životu. Korištenjem mrežnih aplikacija tvrtke i pojedinci mogu puno jednostavnije obaviti više stvari korištenjem manje količine resursa uz brže postizanje velikog broja ciljeva. Budući da koristimo mrežne aplikacije za velik broj stvari i putem njih prosljeđujemo puno osjetljivih informacija, prisiljeni smo od samog početka razvoja mrežne aplikacije voditi brigu o njezinoj sigurnosti. U ovom potpoglavlju biti će opisane dobre prakse koje je potrebno slijediti kako bi na kraju razvili sigurnu mrežnu aplikaciju.

- Održavanje sigurnosti tijekom razvoja: Prije nego što se unajmi strani tim koji se bavi sigurnosnim aspektima, treba znati da se sigurnost mrežnih aplikacija može održavati i tijekom razvoja same aplikacije
- Paranoja je dobra: Dobro pravilo je da se svaki unos podataka u aplikaciju smatra neprijateljskim dok se ne dokaže suprotno. Provjera valjanosti ulaznih podataka provodi se tako da samo ispravno oblikovani podaci prolaze kroz tijek rada u aplikaciji. Time se sprječava obrada loših ili potencijalno oštećenih podataka. Osnovna stvar kod provjere valjanosti unosa je u tome da se unos mora provjeriti i sintaktičkim i semantičkim pristupom.

- Enkripcija podataka: Enkripcija predstavlja osnovni postupak kodiranja informacija kako bi se zaštitile od neovlaštenog pristupa. Sama enkripcija ne sprječava smetnje u prijenosu podataka, ali prikriva razumljiv prikaz sadržaja onima koji nemaju ovlašteni pristup.
- Upravljanje iznimkama: Upravljanje iznimkama predstavlja još jednu sigurnosnu mjeru usmjerenu na sam razvoj softvera. U slučaju bilo kakvog kvara ništa više osim generičke poruke o pogrešci ne bi smjelo biti prikazano. Uključivanje stvarnih sistemskih poruka o pogreškama korisniku neće značiti ništa, dok napadaču može poslužiti kao vrijedan trag za potencijalne upade u sustav. Prilikom razvoja treba imati na umu da postoje samo tri moguća ishoda za bilo koju operaciju: dopusti operaciju, odbij operaciju ili obradi iznimku. U slučaju iznimke aplikacija se treba vratiti na „odbijanje operacije“ i spriječiti bilo kakvo daljnje izvođenje koje se ne bi trebalo dogoditi.
- Provjera autentičnosti, uloge i kontrole pristupa: Neke dobre prakse kod upravljanja korisničkim računima su: zahtijevanje snažne lozinku, mehanizma za oporavak lozinke te neki oblik dvofaktorske autentikacije. Prilikom dizajniranja mrežne aplikacije cilj bi trebao biti da se svakom korisniku daju samo one ovlasti na koje on treba imati pravo. Korisnik nikad ne smije biti u mogućnosti da ovlasti koje ima koristi za dobivanje drugih ovlasti u sustavu.
- Mjere usmjerene na hosting: Bitno je osim sigurnosnih mehanizama koji su usmjereni na razvoj voditi brigu i o upravljanju konfiguracijom na razini usluge kako bi mrežna aplikacija bila sigurna.
- Izbjegavanje pogrešnih konfiguracija: S obzirom na jako veliki broj opcija koje nudi suvremeni softver za upravljanje mrežnim poslužiteljem, to također znači da postoji i jako puno načina da se stvari pokvare: dopuštanje isteka digitalnih certifikata, korištenje zastarjelih zaštitnih protokola, nepotrebno otvoreni portovi na poslužitelju, korištenje zastarjelih softverskih biblioteka itd. Potrebno je osim dobro dokumentiranih postupaka za postavljanje mrežnih aplikacija imati i dobro dokumentirane postupke za pisanje kvalitetnog softvera te za postavljanje mrežnih poslužitelja koji se koriste za posluživanje tih mrežnih aplikacija.
- Implementiranje HTTPS-a: Enkripcija na razini usluge također je korisna mjera koja se može iskoristiti kako bi se informacije zaštitile. U tu svrhu koristi se HTTPS (SSL ili *Secure Sockets Layer*). SSL označava tehnologiju koja se koristi za šifriranje veze između poslužitelja i preglednika, odnosno svi podaci koji se šalju između preglednika i poslužitelja ostaju privatni.
- Revizija i bilježenje: Ne samo da su zapisnici često jedina evidencija na sumnjive aktivnosti, već pružaju i mogućnost praćenja radnji korisnika. Bilježenje aktivnosti ili revizija obično ne zahtjeva puno podešavanja jer je uglavnom ugrađeno u softver mrežnog poslužitelja. Može poslužiti za praćenje sumnjivih aktivnosti, praćenja radnji korisnika i za pregled pogrešaka u aplikaciji koje nisu uhvaćene na razini koda. U rijetkim slučajevima dnevnici mogu biti potrebni u pravnim postupcima.

- Osiguranje kvalitete i testiranje: Ukoliko situacija dopušta, dobro je koristiti uslugu neke treće strane koja je specijalizirana za provođenje testiranja. Mnoge od tih usluga su vrlo pristupačne. Svakako je bolje biti pretjerano oprezan i ne oslanjati se samo na interni postupak osiguranja kvalitete. Dodatni vanjski napori u testiranju mogu poslužiti za otkrivanje problema koji se nisu uspjeli odmah otkriti. Kako bi se testiranje odvijalo glatko i bez napora, potrebno je definirati dobar proces provođenja testiranja koji se lako da replicirati i primijeniti.
- Budite proaktivni: Nove prijetnje i napadi na sigurnost mrežnih aplikacija se stalno razvijaju i potrebno je biti u toku s novim vrstama prijetnji i paralelno raditi na tome da se razvijaju još sigurniji mehanizmi zaštite. Kod zaštite mrežnih aplikacija, prioritet uvijek treba dati visokorizičnim aplikacijama. Kako se sigurnosne prijetnje razvijaju, tako se trebaju razvijati i novi pristupi i planovi za njihovo rješavanje. Ne realno je očekivati da je moguće spriječiti sve napade na mrežne aplikacije, ali treba djelovati proaktivno, odnosno ne čekati da neka nova vrsta napada ugrozi našu aplikaciju pa tek onda djelovati, već se novi mehanizmi zaštite trebaju razvijati paralelno kako se i pojavljuju nove prijetnje. Situacija je takva da se okruženje mrežne sigurnosti neprestano mijenja, te se tome i mi trebamo prilagoditi [11].

3. PRIMJENA SIGURNOSNIH MEHANIZAMA U RAZVOJU MREŽNIH APLIKACIJA

Kako bi se najbolje razumjelo na koje načine je moguće zaštititi mrežne aplikacije, potrebno je poznavati najčešće oblike i vrste napada na mrežne aplikacije kao i hakerski način razmišljanja. U ovom poglavlju biti će stavljen naglasak na najčešće prijetnje na sigurnost mrežnih aplikacija, kao i na način prevencije istih. Također biti će opisan način na koji hakeri razmišljaju prije nego što ugroze određenu mrežnu aplikaciju, te će biti dan uvid u sigurnosne mehanizme koje pruža *Django*, kao i načine na koje te mehanizme možemo iskoristiti u izradi mrežnih aplikacija.

3.1. Hakerski način razmišljanja

Da bi netko danas bio uspješan haker, potrebno je da osim širokog spektra znanja i vještina posjeduje i vrlo poseban način razmišljanja. Softverski inženjeri produktivnost mjere u dodanoj vrijednosti kroz neke nove značajke koje su ugradili ili u obliku poboljšanja koja su napravili na postojećem kodu. Softverski inženjer može reći: „Dodao sam značajke a i b, te je ovo bio uspješan dan za mene.“ Ili može reći nešto kao: „Poboljšao sam značajke a i b za 15%.“ Iako je rad inženjera softvera teško mjerljiv u odnosu na neka tradicionalna zanimanja, ipak ga se može mjeriti. Hakeri s druge strane produktivnost mjere na načine koji su puno kompleksniji i teže mjerljivi. Razlog tome leži u činjenici da većina hakerskog posla označava prikupljanje i analizu podataka. Hakeri se većinom ne fokusiraju na to da unište ili modificiraju softver, već umjesto toga analiziraju softver u potrazi za ulazima u sam sustav. Svaki kod ima mane koje se potencijalno mogu iskoristiti za napad. Dobar haker je stalno u potrazi za novim tragovima koji ga mogu odvesti do otkrića novih ranjivosti. Naravno, čest je slučaj da čak i dobar haker može posvetiti jako puno vremena otkrivanju ranjivosti aplikacije bez naročitog uspjeha. Sasvim je moguće provesti tjedne, pa i mjesece analizirajući mrežne aplikacije prije nego što se pronađe ulazna točka koja hakeru može biti od koristi. Većina hakera vodi detaljne bilješke i evidencije svih pokušaja i lekcija naučenih iz njih. Hakerski posao je takav da se stalno moraju stjecati nova znanja i skupovi vještina kako bi se održao korak sa sve naprednijim mehanizmima zaštite mrežnih aplikacija. Na hakere možemo gledati kao na detektive. Dobar haker je detektiv koji je besprijekorno organiziran, a izvrstan haker je dobar haker koji još posjeduje i širok spektar tehničkih znanja i vještina. Treba imati na umu da se sigurnosni mehanizmi zaštite mrežnih aplikacija konstantno razvijaju i unapređuju, te da te promjene prate i oni koji pokušavaju na bilo koji način ugroziti rad mrežnih aplikacija. Stoga je jako bitno da korak sa novim mehanizmima i tehnologijama drže i oni koji žele svoje mrežne aplikacije sačuvati od vanjskog neželjenog utjecaja [12].

3.2. Najčešće prijetnje na sigurnost mrežnih aplikacija i načini prevencije

U ovom potpoglavlju biti će prikazane najčešće prijetnje na sigurnost mrežnih aplikacija i načini na koje je moguće spriječiti te napade. Neki od najzastupljenijih napada i prijetnji na mrežne aplikacije su: SQL ubrizgavanje, XSS napadi, CSRF napadi, *clickjacking* ili krađa klikova, ranjivost kontrole pristupa, neispravna autentikacija, pogrešne sigurnosne konfiguracije i izlaganje osjetljivih podataka. Većina nabrojanih prijetnji uvrštene su na OWASP-ovu listu najčešćih prijetnji na mrežne aplikacije.

3.2.1. SQL ubrizgavanje

SQL ubrizgavanje predstavlja jedan od najpoznatijih napada na mrežne aplikacije. Predstavlja oblik napada koji je usmjeren na SQL baze podataka i koji dopušta napadaču da pruži vlastite parametre postojećem SQL upitu ili da izbjegne postojeći SQL upit i pruži vlastiti upit. Posljedica ovoga je ugrožena baza podataka [12]. SQL ubrizgavanje je posebna vrsta napada iz razloga što napadaču omogućava pregled podataka koje inače ne bi mogao dohvatiti. To najčešće uključuju pregled podataka koji pripadaju drugim korisnicima ili pregled bilo kojih drugih podataka kojima aplikacija može pristupiti. Glavna odlika ove vrste napada leži u tome da napadač može modificirati ili brisati podatke koje dohvati, što uzrokuje stalnu promjenu sadržaja i ponašanje aplikacije. Uspješan napada rezultira dohvaćanjem osjetljivih podataka, kao što su lozinke, podaci o kreditnoj kartici ili osobni podaci određenog korisnika. Postoji širok izbor SQL napada i tehnika koje napadač može iskoristiti za ubrizgavanje. Neki od primjera SQL ubrizgavanja su: dohvaćanje skrivenih podataka, narušavanje logike, UNION napadi, ispitivanje baze podataka i slijepo ubrizgavanje [13].

- Dohvaćanje skrivenih podataka označava tehniku napada prilikom koje se SQL upit može modificirati kako bi vratio određene podatke. Primjerice, neka postoji mrežna aplikacija za kupnju koja prikazuje proizvode u različitim kategorijama. Kada korisnik odabere kategoriju „Darovi“, njegov preglednik kreira sljedeći URL zapis:

  https://web-stranica-za-kupnju.com/proizvodi?kategorija=Darovi

  To znači da mrežna aplikacija pravi SQL upit prema bazi kako bi mogla dohvatiti detalje o proizvodima iz željene kategorije:

  SELECT * FROM proizvodi WHERE kategorija = 'Darovi' AND released = 1

  Ovaj SQL upit traži od baze da vrati sve detalje iz tablice proizvoda gdje je kategorija označena sa Darovi a da su proizvodi pušteni u prodaju. Pošto aplikacija ne implementira nikakvu obranu od SQL ubrizgavanja, napadač može kreirati napad na sljedeći način:

  https://web-stranica-za-kupnju.com/proizvodi?kategorija=Darovi'--

To će rezultirati sljedećim SQL upitom:

SELECT * FROM proizvodi WHERE kategorija = ' Darovi '--' AND released = 1

Ključna stvar ovdje je niz dvostrukih crtica --, što označava komentar u SQL-u, te će se ostatak upita ovdje tretirati kao komentar. Time se ostatak upita učinkovito uklanja i više neće uključivati AND released = 1 što znači da će se prikazati svi proizvodi, pa i oni koji nisu još objavljeni. Također napadač može doći do proizvoda iz bilo koje kategorije, uključujući i one kategorije za koje ne zna:

https://web-stranica-za-kupnju.com/proizvodi?kategorija=Darovi'+OR+1=1--

Ovo kreira SQL upit:

SELECT * FROM proizvodi WHERE kategorija = 'Darovi' OR 1=1--' AND released = 1

Ovaj modificirani upit vratit će sve stavke gdje je kategorija označena sa Darovi ili gdje je 1=1. Pošto je 1=1 uvijek istina, SQL upit će vratiti sve stavke [13].

- Narušavanje logike aplikacije označava tehniku kojom možemo promijeniti upit u svrhu ometanja logike aplikacije. Uzmimo za primjer aplikaciju koja omogućuje prijavu korisnika s korisničkim imenom i lozinkom. Ako korisnik pošalje recimo korisničko ime „josip“ i lozinku „zivkovic“ aplikacija će provjeriti unesene podatke izvođenjem sljedećeg upita:

  SELECT * FROM users WHERE username = 'josip' AND password = 'zivkovic'

  Ukoliko upit vrati podatke o korisniku, prijava je uspješna, a u suprotnom se prijava odbija. Napadač ovo može iskoristiti kako bi se prijavio u aplikaciji korištenjem niza SQL komentara. On to može ostvariti slanjem sljedećeg upita:

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

  Ovaj upit vratiti će korisnika čije je korisničko ime administrator i uspješno prijaviti napadača kao tog korisnika [13].

- UNION napadi predstavljaju tehniku napada koju napadač može iskoristiti da dohvati podatke iz različitih tablica baze podataka. U slučajevima kada se rezultati SQL upita vraćaju kao odgovori aplikaciji, napadač može iskoristiti to kako bi dohvatio podatke iz drugih tablica baze podataka. To se obavlja pomoću UNION ključne riječi, koja omogućuje izvršavanje dodatnog *SELECT* upita i slanje rezultata kroz originalni upit. Na primjer ako aplikacija izvrši sljedeći upit:

  SELECT ime, opis FROM proizvodi WHERE kategorija = 'Darovi'

tada napadač može poslati:

' UNION SELECT username, password FROM users--

Na ovaj način aplikacija će vratiti sva korisnička imena i njihove lozinke, kao i nazive i opise željenih proizvoda [13].

- Ispitivanje baze podataka koristan je oblik napada jer napadaču može pružiti uvid u informacije o verziji i strukturi baze podataka. Dobivanje ovakvih informacija uvijek je korisno jer one mogu pružiti osnovu za smišljanje drugih oblika napada na sustav. Kako bi se dobila informacija o verziji baze podataka u *Oracle-u* treba izvršiti naredbu:

  SELECT * FROM v$version

  Način na koji se dolazi do ove informacije najčešće ovisi o tipu baze podataka koja se koristi. Kako bi se dobio popis tablica i sama struktura baze podataka, u većini baza podataka dovoljno je izvršiti sljedeću naredbu:

  SELECT * FROM information_schema.tables

  Na ovaj način lako se može dobiti uvid u strukturu baze, kao i popis svih stupaca koje sadrže [13].

- Slijepo ubrizgavanje označava tehniku u kojoj se rezultati upita ne vraćaju u odgovorima aplikacije. Većina instanci SQL ubrizgavanja su slijepe ranjivosti. Aplikacija ne vraća rezultate SQL upita ili pojedinosti o greškama baze podataka unutar svojih odgovora. Slijepe ranjivosti mogu se iskoristiti za dohvaćanje neovlaštenih podataka, ali koriste tehnike koje su puno kompliciranije i teže za izvođenje. Postoje razne tehnike koje se mogu koristiti kod slijepog SQL ubrizgavanja:
  - Moguće je koristiti logiku upita kako bismo uočili razliku u odgovoru aplikacije ovisno o istinitosti jednog uvjeta.
  - Također je moguće uvjetno pokrenuti kašnjenje u obradi upita kako bi na temelju vremena potrebnog aplikaciji za odgovor mogli doći do istinitog uvjeta.
  - Pomoću OAST tehnike moguće je pokrenuti mrežnu interakciju izvan pojasa. Ova tehnika je moćna i djeluje u situacijama u kojima druge tehnike ne rade. Često je moguće neovlašteno prenijeti podatke putem izvan pojasnog kanala, primjerice postavljanjem podataka u DNS za domenu koja se kontrolira [13].

Većinu instanci SQL ubrizgavanja moguće je spriječiti korištenjem parametriziranih upita. To je učinkovit način sprječavanja SQL ubrizgavanja zbog toga što neki parametri nisu specificirani prilikom kreiranja naredbe, već se dodaju tijekom procesa izvođenja. Zbog toga napadači ne mogu modificirati upite čak i ako je

naredba bila njihova. Sljedeći kod je vrlo podložan SQL ubrizgavanju jer je korisnički unos spojen izravno u upit:

```
String query =  "SELECT * FROM proizvodi WHERE kategorija = ' "+ input + " ' " ;
Statement statement = connection.createStatement () ;
ResultSet resultSet = statement.executeQuery(query) ;
```

Ovaj kod može se napisati na način da ne ometa strukturu upita:

```
PreparedStatement statement = connection.prepareStatement( " SELECT * FROM proizvodi
WHERE kategorija = ? ");
Statement.setString (1, input);
ResultSet resultSet = statement.executeQuery();
```

Parametrizirani upiti mogu se koristiti u bilo kojoj situaciji u kojoj se pojavljuje nepouzdani unos. Da bi ovakav upit bio učinkovit u sprječavanju SQL ubrizgavanja, niz koji se koristi u upitu uvijek mora biti hardkodirana konstanta i ne smije sadržavati varijable iz bilo kojeg izvora [13]. Dodatne mjere koje je potrebno poduzeti kako bi se spriječilo SQL ubrizgavanje su provjera valjanosti ulaznih podataka i ograničenje korisničkih prava. Validacija unosa sprječava unos nepravilno oblikovanih podataka u sustav. Stoga kada želimo spriječiti SQL ubrizgavanje ključno je potvrditi sve unose. Također, ograničenje korisničkih prava igra veliku ulogu u sprječavanju ubrizgavanja. Korisnici baze podataka trebali bi imati samo najosnovnija prava. Treba im omogućiti pristup samo određenoj bazi podataka bez mogućnosti kreiranja ili mijenjanja podataka u tablicama [14].

3.2.2. XSS napadi

XSS napadi *(*engl*. Cross-Site Scripting)* predstavljaju jedan od najčešćih oblika napada i ranjivosti diljem interneta, a pojavili su se kao direktan odgovor na sve veću količinu korisničke interakcije u današnjim mrežnim aplikacijama. XSS napadi funkcioniraju na način da iskorištavaju činjenicu da mrežne aplikacije izvršavaju razne skripte u korisničkim preglednicima. Bilo koja dinamički stvorena skripta koja je pokrenuta dovodi mrežnu aplikaciju u opasnost ako skripta koja je pokrenuta može na bilo koji način biti kontaminirana ili modificirana, posebno od strane krajnjeg korisnika. XSS napadi mogu se kategorizirati u tri velike grupe:

- Pohranjeni XSS: zlonamjerna skripta dolazi iz baze podataka mrežne aplikacije
- Reflektirani XSS: zlonamjerna skripta dolazi iz trenutnog HTTP zahtjeva
- DOM XSS: kod je pohranjen i pokrenut u pregledniku

Postoje još razne varijacije na koje je moguće podijeliti XSS napade, ali ove tri kategorije označavaju vrste napada na koje većina modernih mrežnih aplikacija mora redovito paziti. Također, OWASP je ove tri vrste XSS napada označio kao najčešće oblike napada na mrežne aplikacije.

Pohranjeni XSS napadi su vjerojatno najčešći oblici XSS napada. Pohranjeni XSS napadi su vrlo zanimljivi jer predstavljaju oblik XSS napada kojeg je najlakše otkriti, ali često su i najopasniji oblik napada jer mogu utjecati na jako velik broj korisnika. Pohranjeni objekti u bazi podataka često su dostupni većini korisnika. U nekim slučajevima svi korisnici mogu biti izloženi pohranjenom XSS napadu ako je globalni objekt zaražen. Primjerice, ako je na početnoj stranici mrežne aplikacije postavljen video, pohranjeni XSS u naslovu videa potencijalno bi mogao utjecati na svakog posjetitelja te stranice u vremenskom okviru trajanja videa. Zbog toga pohranjeni XSS napadi mogu bit jako opasni za većinu organizacija [12]. Također, uzmimo za primjer aplikaciju koja omogućuje korisnicima slanje poruka koje se prikazuju drugim korisnicima:

** Pozdrav, ovo je moja poruka! **

Aplikacija ne vrši nikakvu obradu i provjeru podataka, tako da napadač može poslati poruku koja napada druge korisnike:

** <script>/* Ovdje ide zlonamjerni kod…*/</script> **

Korisnik prilikom otvaranja poruke pokreće skriptu koja onda radi ono što je napadač zamislio [15.] No s druge strane, sama priroda pohranjenih XSS napada olakšava njihovo otkrivanje. Iako se sama skripta izvršava u pregledniku, ona je također pohranjena u bazi podataka, odnosno na serverskoj strani. Budući da su skripte pohranjene na strani servera, redovitim skeniranjem ulaza baze podataka mogu se pronaći tragovi pohranjenih skripti, što predstavlja jeftin i efikasan način očuvanja mrežnih aplikacija koje pohranjuju jako puno vrsti podataka pruženih od strane krajnjih korisnika. Ovo predstavlja jedan od načina na koje je moguće ublažiti rizik od XSS napada. No napredniji oblici XSS napada ne moraju biti pohranjeni u tekstualnom obliku, već i nekom drugom poput base64 ili binarnom zapisu, te mogu biti pohranjeni na više mjesta i opasni samo kada dođu u doticaj sa određenom uslugom za korištenje [12].

Reflektirani XSS napadi s druge strane rade slično kao i pohranjeni XSS napadi, ali nisu pohranjeni u bazi podataka. Reflektirani XSS napad izravno utječe na kod klijenta u pregledniku bez oslanjanja na server koji će prikazati poruku koja će onda pokrenuti napadačevu skriptu. Pošto nije pohranjen na serveru, reflektirani XSS napad je malo kompliciraniji u usporedbi sa pohranjenim XSS napadom. Reflektirani XSS napad nastaje kada aplikacija primi podatke u HTTP zahtjevu i uzme te podatke unutar trenutnog odgovora bez ikakve provjere [12]. Primjer jednostavne reflektirane XSS ranjivosti slijedi u nastavku:

https://nesigurna-stranica.com/status?message=Sve+je+dobro.

** Status: Sve je dobro. **

Pošto aplikacija ne vrši nikakvu provjeru podataka, napadač lako može kreirati sljedeći napad:

https://nesigurna-stranica.com/status?message=<script>/*+Zlonamjerni+kod+ovdje…+*/</script>

** Status: <script>/* Zlonamjerni kod ovdje… */</script> **

Ukoliko korisnik posjeti URL kojeg je konstruirao napadač, tada se njegova skripta izvršava u korisnikovom pregledniku. U tom trenutku skripta može izvršiti bilo koju radnju i dohvatiti sve podatke kojima korisnik ima pristup [15]. Reflektirane XSS napade puno je teže otkriti pošto direktno napadaju korisnika i nikada nisu pohranjeni u bazi podataka. Ovakvi napadi oslanjaju se na URL-ove koje napadač može lako distribuirati svojim žrtvama. Reflektirani XSS napadi su puno teži za otkrivanje, ali s druge strane puno ih je teže i distribuirati širokom broju korisnika.

DOM XSS napadi predstavljaju posljednju kategoriju napada. DOM (engl. *Document Object Model)* predstavlja prikaz podataka o objektima koji čine strukturu i sadržaj dokumenata na webu. DOM je programsko sučelje za web dokumente. Predstavlja stranicu tako da programi mogu promijeniti strukturu dokumenta, stil i sam sadržaj. DOM predstavlja dokument kao čvorove i objekte i na taj način programski jezici mogu komunicirati sa stranicom. Web stranica je dokument koji se može prikazati u prozoru preglednika ili kao HTML izvor. U oba slučaja radi se o istom dokumentu ali DOM je taj koji omogućuje manipulaciju tim dokumentom [16]. DOM XSS napad može biti pohranjen XSS ili reflektiran XSS napad. Zbog razlika u DOM implementacijama preglednika, neki preglednici mogu biti ranjivi dok drugi nisu. Ovakvi XSS napadi zahtijevaju puno više znanja o DOM-u samog preglednika, kao i dobro poznavanje *JavaScript-a*. Glavna razlika između DOM XSS napada i ostalih oblika XSS napada je u tome što DOM XSS napadi nikad ne zahtijevaju bilo kakvu interakciju sa serverom. Budući da DOM XSS napad ne zahtjeva server kako bi funkcionirao, i „izvor“ i „slivnik“ moraju biti prisutni u DOM-u preglednika. Izvor je DOM objekt koji je sposoban pohraniti neki tekst, a slivnik je DOM API koji je sposoban izvršiti skriptu pohranjenu u obliku teksta. Budući da ovakav oblik XSS napada nikada nije u dodiru sa serverom, gotovo ga je nemoguće detektirati sa statičkim alatom za analizu. DOM XSS napad je također napad s kojim se danas jako teško nositi zbog velikog broja različitih preglednika koji su u upotrebi. Vrlo je moguće da greška u DOM implementaciji koja je isporučena od strane jednog preglednika ne bi bila prisutna u DOM implementaciji isporučenoj od strane nekog drugog preglednika. Isto se može reći i za verzije preglednika. Neke starije verzije preglednika bi mogle biti ranjive, dok moderni preglednici možda nisu [12]. XSS temeljen na DOM-u nastaje kada aplikacija sadrži *JavaScript* na strani klijenta koji obrađuje podatke iz nepouzdanog izvora i na nepouzdan način, obično

pisanjem podataka natrag u DOM. Primjerice, ako imamo aplikaciju koja koristi *JavaScript* za čitanje vrijednosti iz polja za unos i upisivanje te vrijednosti u element unutar HTML-a:

```
var search = document.getElementById('search').value;
var results = document.getElementById('results');
results.innerHTML = ' Tražili ste: ' + search;
```

Ako napadač može kontrolirati vrijednosti polja za unos, on onda može lako konstruirati zlonamjernu vrijednost koja uzrokuje izvršavanje skripte:

```
Tražili ste: <img src=1 onerror='/* Zlonamjerni kod ovdje…*/'>
```

Polje za unos bilo bi popunjeno iz dijela HTTP zahtjeva, dopuštajući pritom napadaču da izvede napad koristeći zlonamjerni URL, na isti način kao i reflektirani XSS. Napadač koji uspije ugroziti mrežnu aplikaciju nekim oblikom XSS napada obično može: izvršiti lažno predstavljanje, izvršiti bilo koju radnju koju korisnik može izvršiti, pročitati sve podatke kojima korisnik ima pristup, pohraniti korisničke podatke za prijavu, oštetiti mrežnu aplikaciju te ubaciti neki oblik virusa, primjerice trojanca u web stranicu. Sam način sprječavanja XSS napada može biti trivijalan u nekim slučajevima, no u nekima može biti jako složen. Sve ovisi o tome koliko je mrežna aplikacija složena i na koji način ona rukuje sa podacima koje kontrolira korisnik. Učinkovito sprječavanje XSS napada uglavnom će uključivati kombinaciju sljedećih mjera:

- Filtriranje ulaznih podataka po dolasku: Na mjestu gdje se prima neki korisnički unos, treba izvršiti filtriranje ulaznih podataka na temelju očekivanog ili važećeg unosa
- Kodiranje podataka na izlazu: Na mjestu gdje se podaci koje korisnik može kontrolirati šalju unutar HTTP odgovora, potrebno je kodirati izlazne podatke kako bismo spriječili njihovo lako interpretiranje
- Korištenje odgovarajućih zaglavlja odgovora: Kako bi se spriječili XSS napadi u HTTP odgovorima koji ne bi trebali sadržavati HTML ili *JavaScript*, treba koristiti zaglavlja *Content-Type* i *X-Content-Type-Options* kako bi se osiguralo da preglednici tumače odgovore na način koji smo i htjeli
- Primjena politike sigurnosti sadržaja (CSP): Kao zadnju liniju obrane, treba koristiti *Content Security Policy* kako bismo ublažili bilo koju XSS ranjivost koja i dalje postoji [15].

3.2.3. CSRF napadi

CSRF napadi ili krivotvorenje zahtjeva na različitim mjestima, predstavlja oblik napada u kojem napadač pokušava prevariti autenticiranog korisnika na način da ga navede da pritisne određeni link primjerice kako bi ukrao sesiju prijavljenog korisnika. Pošto onda napadač posjeduje sesiju prijavljenog korisnika, može direktno utjecati na sustav i stanje aplikacije, umjesto da se zamara načinima da ukrade podatke od korisnika

[17]. CSRF napadi iskorištavaju odnos povjerenja između mrežne aplikacije i samog preglednika, a rade na principu da napadač navede korisnika da izvrši radnje koje inače nije namjeravao izvesti. Također, ovi napadi omogućuju napadaču da zaobiđe određene politike porijekla, koje su i osmišljene kako bi spriječile da različite web stranice ometaju jedna drugu. Često će CSRF napadi proći ne zamijećeno od strane korisnika koji je napadnut, zbog toga što se zahtjevi u pregledniku rješavaju u pozadini. To znači da ovakvi napadi mogu biti korišteni za izvođenje operacija protiv servera, a da korisnik to uopće ne zna. CSRF napad je jedan od najskrivenijih napada koji mogu ugroziti sigurnost mrežne aplikacije [12]. Uspješan CSRF napad navesti će žrtvu da nenamjerno izvede određenu radnju. To može biti promjena e-pošte, promjena lozinke ili prijenos novčanih sredstava. Ovisno o prirodi same radnje, napadač može steći potpunu kontrolu nad korisničkim računom. Također ako žrtva ima neku povlaštenu ulogu unutar mrežne aplikacije, napadač bi mogao preuzeti potpunu kontrolu nad podacima i funkcionalnošću same aplikacije. Da bi CSRF napad bio moguć, moraju biti ispunjena tri ključna uvjeta:

- Relevantna akcija: Postoji radnja unutar aplikacije koju napadač želi pokrenuti. To može biti neka privilegirana radnja (izmjena dopuštenja za druge korisnike) ili radnja specifična za korisnika (promjena lozinke)
- Rukovanje sesijom: Izvođenje te radnje uključuje izdavanje jednog ili više HTTP zahtjeva, a aplikacija se oslanja isključivo na kolačiće sesije za identifikaciju korisnika, te ne postoji nikakav drugi mehanizam za praćenje sesije ili provjeru valjanosti korisničkog zahtjeva
- Nema nepredvidivih parametara zahtjeva: Sami zahtjevi ne sadrže nikakve parametre koje napadač ne može odrediti ili pogoditi

Primjerice, neka imamo aplikaciju koja omogućuje korisniku promjenu adrese e-pošte. Kada korisnik izvrši tu radnju, kreira se sljedeći HTTP zahtjev:

POST /email/change HTTP/1.1
HOST: nesigurna-stranica.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session = jhdsdjkskdjskdQPkdkQ5Tv1lslmk
email = josip@normal-user.com

Ovo ispunjava zahtjeve potrebne za CSRF napad: aplikacija koristi kolačiće sesije kako bi identificirala korisnika, napadač može lako odrediti vrijednosti parametara potrebnih za izvođenje akcije te radnja promjene e-pošte je od interesa za napadača. Nakon ove radnje, napadač će moći pokrenuti radnju ponovnog postavljanja lozinke i potpuno preuzeti kontrolu nad korisničkim računom. Uz te uvjete, napadač može kreirati web stranicu koja sadrži sljedeći HTML:

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

Ako žrtva posjeti napadačevu stranicu, dogodit će se sljedeće: napadačeva stranica pokrenuti će HTTP zahtjev, ako je korisnik prijavljen njegov će preglednik automatski uključiti kolačić sesije u zahtjev te će se zahtjev obraditi na uobičajen način, tretirati će se kao da ga je podnijela žrtva i promijeniti će se adresa e-pošte. Mehanizam isporuke ovakve vrste napada u biti je isti kao i za reflektirani XSS napad. Napadač će postaviti zlonamjerni HTML na web stranicu koju kontrolira, a zatim će pokušati navesti žrtvu da posjeti tu web stranicu. Također treba imati na umu da neki CSRF napadi koriste GET metodu i mogu biti potpuno samostalni u izvršavanju neke radnje. U tim situacijama napadač neće morati koristiti vanjsku stranicu, već će moći žrtvama izravno poslati zlonamjerni URL. U slučaju prethodnog primjera i GET metode, samostalni napad bio bi kreiran na sljedeći način:

```
<img src="https://nesigurna-stranica.com/email/change?email=user@evil-user.net">
```

CSRF napadi iskorištavaju povjerenje između web preglednika, korisnika i web poslužitelja. Prema zadanim postavkama, preglednik vjeruje da su radnje izvršene na korisnikovom uređaju, izvršene u ime tog korisnika. To je djelomično točno jer je korisnik taj koji pokreće samu akciju napada, ali ne razumije šta se događa u pozadini. Kada korisnik klikne na URL, preglednik pokreće HTTP GET zahtjev u njegovo ime bez obzira na to odakle je URL došao. Budući da je URL pouzdan, podaci koje zahtjeva napadač su poslani kroz GET zahtjev. Najbolji način obrane od CSRF napada je uključivanje CSRF tokena unutar zahtjeva. U tom slučaju aplikacija se više neće oslanjati isključivo na kolačiće prilikom kreiranja zahtjeva, već će sada zahtjev sadržavati i parametar čiju vrijednost napadač ne može odrediti. No i dalje treba biti jako oprezan jer postoje načini na koje se obrana može razbiti. Najzanimljivije CSRF ranjivosti nastaju zbog pogrešaka u validaciji CSRF tokena. Stoga je jako bitno da se CSRF tokeni strogo validiraju u svakom slučaju prije izvršavanja određene radnje, da budu nepredvidljivi te da budu vezani za sesiju korisnika [18].

3.2.4. Krađa klikova

Krađa klikova (engl. *Clickjacking*), je zlonamjerna tehnika koja vizualno vara korisnika i navodi ga da klikne na nešto što nije dobro percipirano. Napadač koristi nevidljivi iframe preko autentične web stranice i mami korisnika da radi na toj stranici. Korisnici misle da klikaju po autentičnoj stranici, a zapravo klikaju po

skrivenoj web stranici i izvršavaju radnje koje nikada nisu namjeravali izvršiti [19]. Primjerice, žrtva pristupi stranici čiji je link dobila preko e-pošte i klikne na gumb kako bi osvojila nagradu. Nesvjesno, napadač je naveo žrtvu da pritisne skriveni gumb i to rezultira plaćanjem računa na drugom web mjestu. Ovo je jednostavan primjer clickjackinga. Ova vrsta napada razlikuje se od CSRF napada po tome što korisnik mora izvršiti radnju poput klika na gumb, dok CSRF napad ovisi o lažiranju cijelog zahtjeva bez korisnikova znanja ili unosa. *Clickjacking* napad koristi CSS za stvaranje i manipuliranje slojevima web stranice. Napadač postavlja ciljnu web stranicu kao nevidljivi iframe preko stranice s kojom mami žrtvu kao što je u nastavku prikazano:

```
<head>
        <style>
                #target_website {
                        position:relative;
                        width:128px;
                        height:128px;
                        opacity:0.00001;
                        z-index:2;
                        }
                #decoy_website {
                        position:absolute;
                        width:300px;
                        height:400px;
                        z-index:1;
                        }
        </style>
</head>
...
<body>
        <div id="decoy_website">
        ...decoy web content here...
        </div>
        <iframe id="target_website" src="https://vulnerable-website.com">
        </iframe>
</body>
```

Iframe je pozicioniran unutar preglednika tako da postoji precizno preklapanje između ciljnih radnji koje će žrtva obavljati i web stranice koja služi kao mamac. Apsolutne i relativne vrijednosti položaja koriste se kako bi se osiguralo točno preklapanje ciljane web stranice i stranice koja se koristi kao mamac bez obzira na veličinu zaslona, vrstu preglednika i platformu. *Z-indeks* određuje redoslijed slaganja slojeva. Vrijednost neprozirnosti definirana je kao 0,0 ili jako blizu nule, tako da je sadržaj iframe-a transparentan za korisnika. Neki preglednici mogu otkriti pokušaj napada krađom klikova na temelju transparentnosti *iframe-a*. Zato često napadači odabiru vrijednosti neprozirnosti tako da postignu željeni učinak ali bez pokretanja zaštitnih mehanizama samog preglednika. Skripte za razbijanje okvira predstavljaju jedan od zaštitnih mehanizama u borbi protiv krađe klikova. No ovakvi mehanizmi su često specifični za preglednik i samu platformu, a zbog fleksibilnosti HTML-

a napadačima nije problem takve mehanizme zaobići. Zbog toga su osmišljeni protokoli koji ograničavaju korištenje *iframe-a* unutar preglednika i ublažavaju krađu klikova. Zaštita od strane servera protiv krađe klikova pruža se na način da se definiraju ograničenja nad upotrebom određenih komponenti kao što su iframe-ovi, a sama implementacija zaštite ovisi o usklađenosti preglednika i provedbi zadanih ograničenja. Dva mehanizma za zaštitu od krađe klikova su: *X-Frame-Options* i *Content Security Policy*. *X-Frame-Options* pruža vlasniku web stranice kontrolu nad upotrebom iframe-ova na način da se uključivanje web stranica koje se nalaze unutar okvira može zabraniti. *X-Frame-Options* nije dosljedno implementiran u svim preglednicima, međutim ako se pravilno primjeni u kombinaciji sa CSP-om može pružiti učinkovitu zaštitu od krađe klikova [20].

3.2.5. Ranjivost kontrole pristupa

Ranjivost kontrole pristupa predstavlja sigurnosni propust u kojem korisnici mogu djelovati izvan predviđenih dopuštenja. Kvarovi obično dovode do neovlaštenog otkrivanja informacija, izmjene ili uništenja svih podataka ili obavljanja poslovne funkcije izvan korisničkog ograničenja. Sam pojam kontrole pristupa označava sigurnosnu mjeru koja određuje i regulira koji korisnici mogu vidjeti ili koristiti resurse u određenom okruženju. Loša kontrola pristupa događa se kada postoji nedostatak ili nedovoljna središnja kontrola pristupa. Čak i u onim situacijama u kojima su mrežne aplikacije zaštićene od neautenticiranih korisnika, napadači se potencijalno mogu maskirati u korisnika kojem sustav vjeruje [21]. Uobičajene ranjivosti kontrole pristupa uključuju:

- Kršenje načela najmanje privilegije, odnosno pristup bi trebao biti odobren prema određenim mogućnostima, ulogama ili korisnicima, ali je dostupan svima
- Zaobilaženje provjere kontrole pristupa mijenjanjem URL-a (mijenjanje parametara ili prisilno pregledavanje)
- Dopuštanje pregledavanja ili uređivanja tuđeg računa
- Povećanje privilegija. Mogućnost djelovanja unutar aplikacije bez prijave u sustav ili djelovanja kao da su dodijeljene veće privilegije nego što stvarno jesu
- Prisilno pregledavanje autenticiranih stranica kao neautorizirani korisnik ili privilegiranih stranica kao standardni korisnik [22]

Ranjivosti kontrole pristupa moguće je riješiti implementacijom načela najmanjih privilegija i kontrole pristupa temeljene na raznim ulogama, koja ograničavaju korisnička prava pristupa na minimum koji je neophodan da bi se određena funkcija obavila [21].

3.2.6. Neispravna autentikacija

Neispravna autentikacija ili oštećena autentikacija, podrazumijeva napade koji za cilj imaju preuzimanje jednog ili više korisničkih računa dajući pritom napadaču iste privilegije kao što ih ima i napadnuti korisnik. Autentikacija je "pokvarena" u slučaju kada napadači mogu mijenjati lozinke, podatke o korisničkom računu ili sesije kako bi preuzeli identitet korisnika. Općenito se neispravna autentikacija odnosi na slabosti u dva područja: upravljanje sesijom i upravljanje podacima za prijavu korisnika. Oba slučaja smatraju se neispravnom provjerom autentičnosti jer napadači mogu iskoristiti oba načina da se maskiraju u korisnika sustava: oteti ID sesije ili ukrasti podatke za prijavu. Upravljanje sesijom dio je neispravne provjere autentičnosti. Sesija predstavlja niz mrežnih transakcija povezanih s istim korisnikom unutar određenog vremenskog razdoblja. Mrežne aplikacije izdaju svakom korisniku jedinstveni ID sesije za svaki posjet, što aplikaciji omogućuje komunikaciju s korisnikom dok se kreće kroz stranicu. ID sesije obično ima oblik kolačića i URL parametra. Bez odgovarajućih zaštitnih mjera, mrežne aplikacije su ranjive na otmicu sesije, u kojoj napadači koriste ukradene ID-ove za lažno predstavljanje. Najjednostavniji primjer otmice sesije je korisnik koji se zaboravi odjaviti iz aplikacije i zatim se udalji od svog uređaja. Napadač tada može nastaviti svoju sesiju. Drugi uobičajeni način za otmicu sesije je "prepisivanje URL-a“. U ovom scenariju ID sesije pojedinca se pojavljuje unutar URL-a web stranice. Svatko tko može vidjeti taj URL, tipa putem nezaštićene Wi-Fi veze, može se uključiti u sesiju. Fiksacija sesije predstavlja još jedan oblik otmice sesije, a glavna ideja iza ovog napada leži u tome da napadač unaprijed odredi ID sesije koji će žrtva koristiti. Napadač tada može žrtvi poslati vezu koja sadrži unaprijed određeni ID sesije. Napadač može upotrijebiti taj unaprijed određeni ID sesije da oponaša žrtvu nakon što se žrtva prijavi. Ukoliko napadač pokaže taj ID sesije serveru, server će utvrditi da ID sesije odgovara autenticiranoj sesiji i odobriti pristup zaštićenim podacima i resursima. Jedan od načina rješavanja ovog problema je rotiranje ID-ova sesije nakon što se korisnik prijavi, umjesto da se korisniku da isti ID prije i nakon autentikacije. Mrežna aplikacija bi nakon što se korisnik prijavi trebala izdati novi ID sesije tom korisniku i na taj način bi unaprijed određeni ID sesije postao beskoristan. Krađa vjerodajnica, ili podataka za prijavu predstavlja još jedan problem neispravne autentikacije. Napadači koriste razne metode kako bi pogodili ili naveli korisnike da otkriju svoje tajne podatke. Neki od načina krađe povjerljivih podataka poput korisničkih lozinki su:

- Punjenje vjerodajnica: Napadači koji pristupe bazi podataka s nekriptiranom e-poštom i lozinkama, često popis prodaju ili daju drugim napadačima. Ti napadači zatim koriste napad grubom silom kako bi testirali podatke ukradene s jedne stranica na različitim aplikacijama . Ova taktika se pokazala uspješnom jer ljudi često koriste istu lozinku u više aplikacija.
- Prskanje lozinki: Slično prethodnom napadu, ali koristi se skup slabih ili uobičajenih lozinki za provalu u korisnički račun. Istraživanja su pokazala da jako velik broj korisnika koristi lozinke

poput "123456" ili "lozinka" ili nekih psovki ili poznatih sportskih imena. Slično napadu grubom silom, ali često izbjegne automatsko zaključavanje nakon previše neuspjelih pokušaja prijave i to zato što pokušava istu lozinku na različitim korisnicima umjesto da pokušava različite lozinke na jednom korisniku.

- Phishing napadi: Vrsta napada u kojem napadači pošalju žrtvama e-poruku koja se pretvara da je iz pouzdanog izvora, te navede korisnika da podijeli svoje tajne podatke.

Napadi s oštećenom autentikacijom su česti i opasni, ali moguće ih je spriječiti. Neke od zaštitnih mjera su:

- Upravljanje duljinom sesije: Svaka mrežna aplikacija treba imati mogućnost automatskog prekida sesije, bilo nakon odjave, razdoblja bez aktivnosti ili određenog vremena. Duljinu sesije treba prilagoditi vrsti korisnika i aplikacije koju koriste.
- Rotacija ID-ova sesije: Kao što je već spomenuto, najbolji način za sprječavanje fiksiranja sesije je da se korisniku nakon prijave ID sesije odmah promijeni.
- Višefaktorska autentikacija: Implementacija višefaktorske autentikacije je broj jedan savjet za sprječavanje automatiziranih napada, napada gomilanjem vjerodajnica te napada grubom silom. Višefaktorska autentikacija pruža dodatnu razinu sigurnosti korisničkim računima.
- Izbjegavanje slabih lozinki: Prilikom dizajniranja stranice za prijavu treba voditi računa o duljini i složenosti samih lozinki. Savjetuje se i automatsko odbijanje najčešćih lozinki na webu.
- Implementacija zaštite od napada grubom silom: Napadi koji uključuju neispravnu autentikaciju mogu ne samo ugroziti podatke, već i srušiti mrežnu aplikaciju. Tijekom različitih napada na mrežnu aplikaciju količina prometa i broj upita može se povećati i 200 puta, stoga je zaštita od napada grubom silom nužna za stabilnost mrežne aplikacije [23].

3.2.7. Pogrešne sigurnosne konfiguracije

Pogrešne sigurnosne konfiguracije predstavljaju oblik prijetnje na sigurnost mrežnih aplikacija do kojih dolazi kada sigurnosne postavke nisu definirane na pravi način u procesu konfiguracije ili se koriste i primjenjuju unaprijed zadane postavke. To naravno može utjecati na bilo koji sloj aplikacije ili sustava. Do pogrešne konfiguracije može doći iz raznih razloga. Današnje mrežne infrastrukture su zamršene i stalno se mijenjaju te je jako bitno da organizacije provode redovite sigurnosne kontrole kako bi na vrijeme uočile neke promjene u konfiguraciji i sustavu. Pogrešne sigurnosne konfiguracije mogu biti rezultat vrlo jednostavnih propusta, ali svejedno mogu izložiti aplikaciju napadu. Ponekad, pogrešna konfiguracija može ostaviti informacije izloženima, tako da napadač neće ni morati izvršiti aktivni napad. Općenito vrijedi da što je više podataka i koda izloženo korisnicima, to je veći rizik za sigurnost mrežne aplikacije. Neke od uobičajenih pojava koje mogu dovesti do pogrešne konfiguracije su:

- Omogućeni su zadani računi i lozinke: Korištenje zadanih postavki za korisničke račune i lozinke uobičajena je sigurnosna pogrešna konfiguracija i može napadačima olakšati upad u sustav
- Princip sigurne lozinke nije implementiran: Napadači mogu dobiti neovlašteni pristup sustavu korištenjem metode grube sile sve dok ne budu uspješno autenticirani
- Softver je zastario, a nedostaci nisu zakrpani: Izbjegavanje ažuriranja softvera može napadačima olakšati i omogućiti jednostavniji upad u sustav iskorištavanjem propusta u softveru koji nisu zakrpani
- Datoteke i direktoriji su nezaštićeni: Ukoliko se datoteke i direktoriji ostave nezaštićenima, napadači mogu koristiti tehnike prisilnog pregledavanja kako bi dobili pristup zaštićenim datotekama i direktorijima
- Neiskorištene značajke nisu uklonjene: Ostavljanje neiskorištenih i nepotrebnih značajki može ugroziti aplikaciju i napadačima omogućiti ubacivanje zlonamjernog koda kojeg aplikacija zatim izvršava
- Sigurnosne značajke nisu ispravno konfigurirane: Neispravna konfiguracija sigurnosnih značajki i zanemarivanje održavanja čini aplikaciju ranjivom
- Loše prakse u procesu stvaranja aplikacije: Loše prakse prilikom kodiranja mogu dovesti do raznih napada na aplikaciju. Primjerice, nedostatak valjane provjere ulazno izlaznih podataka može dovesti do napada ubacivanjem koda kojeg će aplikacija potom izvršavati

Početni korak u prevenciji pogrešnih sigurnosnih konfiguracija je valjano razumijevanje značajki sustava i svih njegovih dijelova. Također, rizik od pogrešne konfiguracije može se ublažiti sljedećim mjerama: redovito provoditi zakrpe i ažuriranje softvera, provođenje revizija i skeniranje sustava, potrebno je imati dobro održavan i strukturiran razvojni ciklus, educirati zaposlene o važnosti sigurnosnih konfiguracija, primijeniti kontrole pristupa datotekama i direktorijima, koristiti minimalnu platformu bez suvišnih značajki i komponenti, šifriranje podataka u mirovanju, izmjena bilo kakvih izvornih postavki u aplikaciji te korištenje arhitekture koja nudi učinkovito i sigurno odvajanje elemenata [24].

3.2.8. Izlaganje osjetljivih podataka

Izlaganje osjetljivih podataka predstavlja sigurnosnu ranjivost koja se odnosi na zaštitu svih podataka i resursa koje mrežna aplikacija koristi. Ova ranjivost nastupa kada dođe od nesvjesnog izlaganja osjetljivih podataka ili kada sigurnosni incident proizvede slučajno ili nezakonito uništenje, gubitak, izmjenu ili neovlašteno otkrivanje i pristup osjetljivim podacima. Današnje mrežne aplikacije rade s velikom količinom podataka, spremaju ih u baze podataka, mijenjaju te podatke i po potrebi im pristupaju. Zbog toga je jako bitno da su osjetljivi podaci šifrirani u svakom trenutku, bilo da se radi o mirovanju ili prijenosu tih podataka [25]. Izlaganje osjetljivih podataka može se podijeliti u tri grupe:

- Povreda povjerljivosti: Postoji neovlašteno ili slučajno otkrivanje te pristup osjetljivim podacima
- Povreda integriteta: Postoji neovlaštena ili slučajna izmjena osjetljivih podataka
- Povreda dostupnosti: Postoji neovlašteni ili slučajni gubitak pristupa osjetljivim podacima ili njihovo uništavanje [26]

Kako bi se ovakva sigurnosna ranjivost izbjegla potrebno je primijeniti odgovarajuće mjere zaštite osjetljivih podataka koje mrežna aplikacija koristi. Kako bi se korisnici mrežne aplikacije zaštitili, bitno je redovito pratiti sve podatke pohranjene unutar sustava i obavljati redovite revizije. Također je bitno da se razumije rizik koji dostupni osjetljivi podaci predstavljaju za sustav koji ih koristi. Korištenje odgovarajućih sigurnosnih mehanizama u šifriranju je od velike važnosti. Sigurnosni standard za mrežne aplikacije preporučuje AES (256 bita i više) te RSA (2048 bita i više). Kod prijenosa podataka treba koristiti HTTPS s odgovarajućim certifikatom te treba onemogućiti pristup mrežnoj aplikaciji onim vezama koje nisu HTTPS. Skladištenje podataka također igra veliku ulogu u sprječavanju izlaganja osjetljivih podataka. Osjetljive podatke koje nije potrebno čuvati i ne treba čuvati u bazi podataka. Podaci koji nisu dostupni ne mogu biti ukradeni [25].

3.3. Implementacija sigurnosnih mehanizama korištenjem Django radnog okvira

Django je *Python-ov* mrežni radni okvir koji omogućava brz i jednostavan razvoj mrežnih aplikacija. Razvijen je od strane iskusnih programera te rješava većinu stvari koje mogu predstavljati problem prilikom razvoja mrežnih aplikacija, pa se programeri mogu posvetiti razvoju aplikacije bez potrebe za izmišljanjem "tople vode". Osmišljen je kako bi pomogao programerima da što brže moguće svoju aplikaciju dovedu iz faze koncepta do finalne faze razvoja. *Django* je izuzetno skalabilan, te neke od najprometnijih mrežnih aplikacija iskorištavaju tu *Djangovu* sposobnost brzog i fleksibilnog skaliranja. Kao i svi moderni alati za razvoj mrežnih aplikacija *Django* ozbiljno shvaća sigurnost i pomaže programerima da izbjegnu uobičajene sigurnosne pogreške. U ovom potpoglavlju biti će stavljen naglasak na sigurnosne mehanizme koje pruža *Django*. Također će biti opisani neki od najkorištenijih sigurnosnih mehanizama, njihov način implementacije te zašto ih je dobro koristiti u razvoju mrežnih aplikacija.

3.3.1. Izrada funkcija za prijavu i odjavu korisnika

Danas svaka mrežna aplikacija mora imati kvalitetne mehanizme prijave i odjave korisnika kako bi se izbjegle neke od sigurnosnih prijetnji koje su ranije spomenute. Kreiranje tih funkcija u *Djangu* je vrlo jednostavno. *Django* pruža neke gotove mehanizme kako bi provjerio podatke za prijavu te obavio proces prijave i odjave korisnika. *Djangov* bekend za autentikaciju, *django.contrib.auth*, pruža *authenticate()* i *login()* funkcije čiji je posao provjera autentičnosti i prijava korisnika. Funkcija *authenticate()* prima dva argumenta ključnih riječi *username* i *password* te vraća objekt tipa *User* ako su predani *username* i *password* točni. U

protivnom se vraća None. Funkcija *authenticate()* samo provjerava jesu li predani podaci za prijavu važeći ili ne, ona ne prijavljuje korisnika. Za prijavu korisnika koristi se *login()* funkcija. Ona također prima dva argumenta, *request* objekt (*HttpRequest*) i stvoreni *User* objekt. Za prijavu korisnika sprema ID korisnika u sesiju, koristeći *Djangov* radni okvir za sesije. Nakon što je korisnika prijavljen, trebao bi se moći odjaviti, a to je zadaća *logout()* funkcije. Ona prihvaća *HttpRequest* objekt i nakon odjave vraća *None*. Funkcija *logout()* potpuno briše podatke o sesiji povezane s prijavljenim korisnikom. Funkcija *logout()* također uklanja kolačić iz preglednika. Korištenjem ovih mogućnosti, vrlo je lako kreirati vlastiti sustav prijave u *Djangu*. Primjer funkcije za prijavu dan je u nastavku.

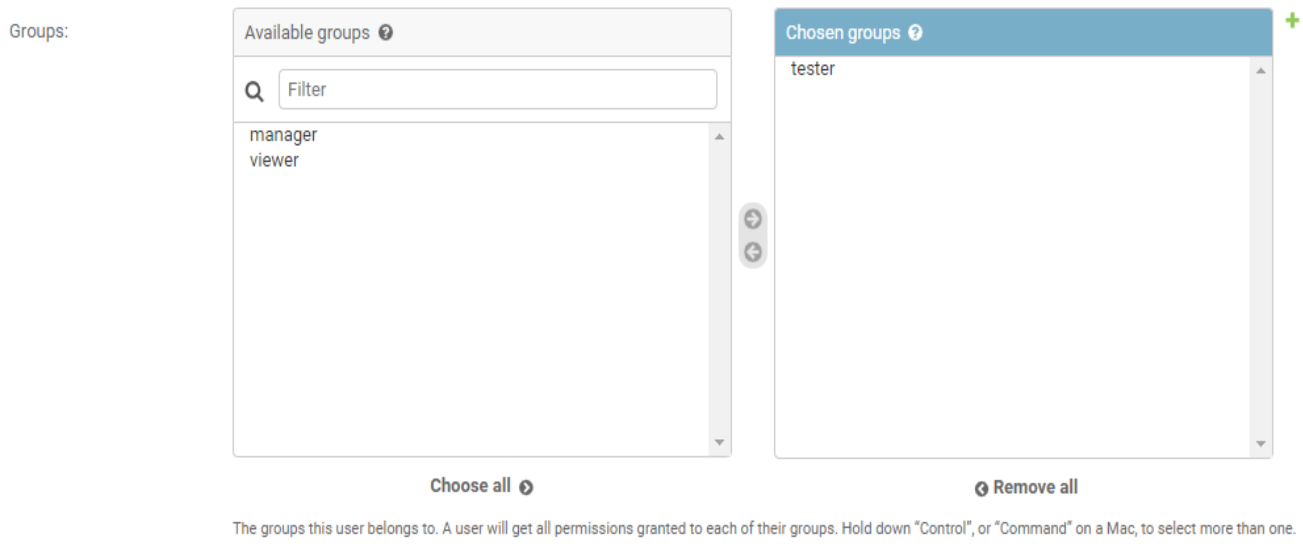
```
def login(request):
    username = password = ''
    language = request.LANGUAGE_CODE
    context["alert"] = language
    if request.POST:
        username = request.POST['username']
        password = request.POST['password']
        context["username"] = username
        user = authenticate(username=username, password=password)
        if user is not None:
           if user.is_active:
              login(request, user)
           else:
              print ("Cannot connect!")
              context["loginMessage"] = "Invalid login details!"
              return render(request, 'WBTapp/testerTemplates/index.html',context)
           if not check_role(request):
                return HttpResponse("Your user is in the creation process!")
           elif 'manager' in check_role(request):
              return redirect('infoScreen/')
           elif 'tester' in check_role(request):
              return redirect('testerScreen')
           elif 'viewer' in check_role(request):
              return redirect('infoScreen/')
           else:
                return HttpResponse("You don't have any permissions.")
           else:
               return HttpResponse("Your account is disabled.")
         else:
         print ("Invalid login details!")
         context["loginMessage"] = "Invalid login details!"
         return render(request, 'WBTapp/testerTemplates/index.html',context)
    context["loginMessage"] = "Refresh!"
    return render(request, 'WBTapp/testerTemplates/index.html',context)
```

U nastavku je prikazana funkcija za odjavu napisana u *Djangu*.

```
@login_required(login_url=login)
def logout_user(request):
logout(request)
return redirect('index')
```

3.3.2. Definiranje grupa korisnika i njihovih prava unutar aplikacije

Django pruža cijelo sučelje unutar kojeg se mogu kreirati grupe korisnika za lakše upravljanje korisnicima i njihovim pravima unutar sustava. Upravo je taj mehanizam iskorišten kako bi se korisnicima prava mogla dodjeljivati na temelju grupe kojoj pripadaju. Korisnici na taj način dobivaju sva prava koja su dodijeljena pojedinoj grupi. Za potrebe aplikacije kreirane su tri grupe korisnika: manager, tester i viewer. Za svaku grupu korisnika napravljen je ekran kojem pristupaju nakon uspješne prijave. Svaki korisnik može biti smješten unutar više grupa i na taj način dobiva prava svake grupe unutar koje se nalazi. Na slici 3.1. prikazan je dio admin sučelja unutar kojeg su prikazane grupe kojima pojedini korisnik može pripadati.



Slika 3.1. *Prikaz grupa unutar Django admin sučelja*

3.3.3. Postavljanje LDAP-a

LDAP (engl. *Lightweight Directory Access Protocol*) predstavlja protokol koji aplikacijama omogućuje brzo postavljanje upita o korisničkim informacijama. Tvrtke pohranjuju korisnička imena, lozinke, adrese e-pošte i druge statične podatke unutar imenika. LDAP je otvoreni, aplikacijski protokol za pristup i

održavanje tih podataka. LDAP se kao protokol može koristiti s nizom različitih imeničkih programa. Uobičajeno, imenik sadrži podatke koji su:

- Opisni: Točke poput naziva i lokacije zajedno definiraju neko sredstvo u imeniku
- Statički: Informacije se ne mijenjaju mnogo, a kada se i mijenjaju, razlike su minimalne
- Vrijedni: Podaci koji su pohranjeni unutar imenika su ključni za osnove poslovne funkcije

Prosječni se zaposlenik povezuje s LDAP-om na desetke puta dnevno. LDAP upit obično sadržava:

- Vezu sesije: Korisnik se spaja na poslužitelj preko LDAP porta
- Zahtjev: Korisnik poslužitelju šalje neki oblik upita
- Odgovor: LDAP protokol postavlja upit direktoriju, pronalazi informacije i isporučuje ih korisniku
- Završetak: Korisnik prekida vezu s LDAP portom

Uobičajena upotreba LDAP-a je pružanje središnjeg mjesta za provjeru autentičnosti, odnosno pristup imeniku koji pohranjuje korisnička imena i lozinke. LDAP se tada može koristiti u različitim aplikacijama ili uslugama za provjeru valjanosti korisnika. LDAP se često koristi sa *Microsoft Active Direktorijem*, ali se može koristiti i sa drugim alatima kao što su *OpenLDAP*, *Red Hat Direktorij* i ostali. *Django* podržava rad s LDAP-om putem modula koji se zove *django-auth-ldap*. Instalacija i postavljanje modula unutar aplikacije je dosta jednostavno. Sam paket instalira se pomoću sljedeće naredbe:

pip install django-auth-ldap

Ovaj modul zahtjeva *python-ldap >=3.0* i *OpenLDAP* biblioteke i zaglavlja koje trebaju biti dostupne u sustavu. Kako bi se autentikacija korisnika izvodila pomoću LDAP imenika, unutar *settings.py* datoteke potrebno je postaviti sljedeću liniju koda:

AUTHENTICATION_BACKENDS = ["django_auth_ldap.backend.LDAPBackend"]

Time se osigurava da se kao bekend za prijavu i provjeru korisnika koristi LDAP imenik. Za potrebe testiranja aplikacije korišteni su *OpenLDAP* i *phpLDAPadmin*. *OpenLDAP* predstavlja najpopularniji LDAP poslužitelj otvorenog koda na tržištu. On omogućuje izradu i upravljanje LDAP imenikom. No zbog vrlo skromnog korisničkog sučelja zahtjeva dobro poznavanje LDAP protokola i strukture direktorija. Iz tog se razloga *OpenLDAP* često nadopunjava vanjskim aplikacijama, poput *phpLDAPadmin* mrežne aplikacije koja omogućuje interakciju s *OpenLDAP-om* putem osnovnog korisničkog sučelja. *phpLDAPadmin* je mrežna aplikacija koja se koristi za administriranje LDAP imenika. U nastavku je prikazan *docker-compose* YAML dokument koji se koristio za postavljanje *OpenLDAP* poslužitelja kao i *phpLDAPadmin* sučelja za vođenje LDAP imenika:

```
version: '3.7'
services:
    ldap_server:
        image: osixia/openldap:latest
        environment:
            LDAP_DOMAIN: example.com
            LDAP_BASE_DN: dc=example,dc=com
            LDAP_ADMIN_PASSWORD: Passw0rd
        ports:
            - 389:389
        volumes:
            - ./ldap_data:/var/lib/ldap
            - ./ldap_config:/etc/ldap/slapd.d
    ldap_server_admin:
        image: osixia/phpldapadmin:0.7.2
        ports:
            - 8092:80
        environment:
            PHPLDAPADMIN_LDAP_HOSTS: ldap_server
            PHPLDAPADMIN_HTTPS: 'false'
```

U nastavku je prikazan primjer konfiguracije postavki unutar *settings.py* datoteke:

```
AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
AUTH_LDAP_USER_DN_TEMPLATE = 'uid=%(user)s,ou=users,dc=example,dc=com'
AUTH_LDAP_USER_ATTR_MAP = {
        "first_name": "givenName",
        "last_name": "sn",
        "email": "mail",
}
AUTH_LDAP_ALWAYS_UPDATE_USER = True
AUTH_LDAP_CACHE_TIMEOUT = 3600
AUTHENTICATION_BACKENDS = ( "django_auth_ldap.backend.LDAPBackend",)
```

3.3.4. Upravljanje sesijom

Upravljanje istekom sesije predstavlja sigurnosni mehanizam koji je vrlo koristan i može poslužiti da se određene sigurnosne prijetnje izbjegnu. Primjerice, uzmimo u obzir situaciju u kojoj je korisnik prijavljen u sustav i obavlja određene zadatke i poslove u aplikaciji. Problem ovdje može nastati ukoliko se korisnik nakon odrađenog posla zaboravi odjaviti iz sustava ili je tijekom obavljanja posla bio primoran fizički se udaljiti od računala na neko vrijeme. Neki ovdje potencijalni problemi su:

- njegovu sesiju može nastaviti bilo tko (obavljati posao u njegovo ime)

- ukoliko se radi o privilegiranom korisniku koji ima uvid u dijelove sustava kojima ostali korisnici ne mogu pristupiti, sada i niže rangirani korisnici sustava mogu "baciti pogled" na njegov zaslon računala
- korisnik može biti prisutan i primjerice čitati nešto u aplikaciji i biti "neaktivan", njemu sesija u pozadini istječe, a on o tome ne zna ništa
- napadačima je olakšan upad u sustav

Iz navedenih razloga vidljivo je da je mehanizam koji će primijetiti neaktivnost korisnika i istoga odjaviti iz sustava od velike važnosti. *Django* nudi rješenje i za ovaj problem u obliku *django-session-security* mehanizma. Ovaj mehanizam treba automatski odjaviti neaktivni preglednik i na taj način zaštititi osjetljive podatke koji mogu ostati prikazani na zaslonu. Sama implementacija mehanizma je jednostavna i sastoji se od sljedećih koraka:

- Potrebno je instalirati paket naredbom *pip install django-session-security*
- Unutar *settings.py* datoteke pod *INSTALLED_APPS* dodati *'session_security'*
- Unutar *settings.py* datoteke pod *MIDDLEWARE_CLASSES* nakon *AuthenticationMiddleware-a* dodati *'session_security.middleware.SessionSecurityMiddleware',*
- Potrebno se uvjeriti da *TEMPLATE_CONTEXT_PROCESSORS* unutar *settings.py* datoteke sadrži *'django.core.context_processors.request'*
- Unutar *urls.py* datoteke pod *URL-ove* treba dodati:
  url(r'session_security/', include('session_security.urls')),
- Unutar osnovnog predloška treba postaviti *{% include 'session_security/all.html' %}*

Ovime je implementacija spomenutog mehanizma gotova te ostaje još samo unutar *settings.py* datoteke postaviti varijable obavijesti i isteka vremena:

SESSION_SECURITY_WARN_AFTER: x sekundi (korisnik će nakon x sekundi biti obaviješten o isteku sesije)

SESSION_SECURITY_EXPIRE_AFTER: y sekundi (ukoliko korisnik nije aktivan, biti će odjavljen nakon y sekundi)

3.3.5. Primjena dekoratora

Dekorator u *Pythonu* predstavlja funkciju koja uzima neku drugu funkciju kao svoj argument, a vraća još jednu funkciju. Dekoratori mogu biti iznimno korisni jer dopuštaju proširenje postojeće funkcije, bez ikakvih promjena izvornog koda funkcije. Možemo reći da su dekoratori funkcije primijenjene na funkcije ili klase, što dovodi do obogaćene funkcionalnosti. Dekoratori u *Djangu* predstavljaju način ograničavanja pristupa određenim prikazima. Mogu biti iznimno korisni kada primjerice treba odvojiti prijavljene korisnike

od neautoriziranih korisnika ili kada treba stvoriti administratorsku stranicu kojoj mogu pristupati samo privilegirani korisnici. Dekoratori se u *Django* aplikacijama koriste jer predstavljaju jednostavan način za čišćenje koda i odvajanje određenih funkcionalnosti iz samih prikaza. *Django* dolazi sa nekoliko korisnih ugrađenih dekoratora kao što su: *@login_required* i *@permission_required* za korisnička dopuštenja te *@require_http_methods* za ograničavanje metoda zahtjeva (GET i POST). Od navedenih ugrađenih dekoratora posebno treba naglasiti *@login_required* dekorator koji se i najviše koristio kod ograničavanja prikaza prilikom izrade mrežne aplikacije. Navedeni dekorator radi na principu da provjerava je li korisnik koji pokušava otići na neku stranicu prijavljen u sustav ili ne. Ukoliko je korisnik prijavljen omogućiti će mu se putanja do određene web stranice, a ukoliko se nije prijavio vratiti će ga se na stranicu za prijavu, te će mu prikaz određene stranice biti omogućen nakon što obavi prijavu. Ovo se pokazalo jako korisnim jer se izbjegava potreba za pisanjem koda za provjeru prijave korisnika unutar svakog prikaza i funkcije, te je kao posljedica toga naš kod kraći i pregledniji. Primjer implementacije i upotrebe *@login_required* dekoratora dan je u nastavku:

```
@login_required(login_url=login)
def primjer(request):
return render(request, 'PRIMJERapp/primjerTemplates/primjer.html', context)
```

3.3.6. Dvofaktorska autentikacija

Dvofaktorska autentikacija predstavlja dodatni sloj sigurnosti koji se koristi kako bi se osiguralo da su ljudi koji pokušavaju dobiti pristup mrežnoj aplikaciji ili općenito mrežnom računu oni za koje se predstavljaju da jesu. Korisnik obično prvo unosi svoje korisničko ime i lozinku. Zatim, umjesto da odmah dobije pristup sustavu, sustav od korisnika traži još jednu informaciju. Taj drugi faktor kojeg sustav zahtjeva najčešće potječe iz jedne od sljedećih kategorija:

- Nešto što znate: Ovo može biti neki PIN, lozinka ili odgovor na "tajno pitanje"
- Nešto što imate: Ovdje spada nešto što bi korisnik trebao imati u svojem posjedu, poput kreditne kartice, pametnog telefona ili hardverskog tokena
- Nešto što jeste: Ovo je malo naprednija kategorija i može uključivati uzorak otiska prsta, skeniranje šarenice ili otisak glasa

Dvofaktorska autentikacija predstavlja dobru sigurnosnu mjeru jer čak i ako je naša lozinka ukradena ili telefon izgubljen, šanse da će netko imati naše podatke drugog faktora jako su male. No, nisu ni svi mehanizmi drugog faktora isti. Danas se koristi nekoliko tipova dvofaktorske autentikacije od kojih su neke jače i složenije od drugih, ali sve nude bolju zaštitu od samih lozinki. Najčešći oblici dvofaktorske autentikacije su:

- Hardverski tokeni: Najstariji oblik 2FA-a, hardverski tokeni su mali i proizvode nove numeričke kodove svakih 30 sekundi. Kada korisnik pokuša pristupiti računu, baci pogled na uređaj i unese prikazani kod u aplikaciju.
- SMS poruke i glasovni 2FA: Oblik dvofaktorske autentikacije u kojem se jednokratna šifra (OTP) šalje direktno putem tekstualne poruke korisniku. Na sličnom principu radi i glasovni 2FA koji automatski poziva korisnike i usmeno isporučuje 2FA kod.
- Softverski tokeni za 2FA: Vjerojatno najpopularniji oblik dvofaktorske autentikacije koji koristi softverski generiranu, vremenski ograničenu jednokratnu lozinku (TOTP). Korisnik mora preuzeti i instalirati 2FA aplikaciju na pametni telefon ili svoju radnu površinu. Korisnik prilikom prijave prvo unosi korisničko ime i lozinku, a nakon toga unosi kod prikazan u aplikaciji.
- Push obavijesti za 2FA: Umjesto oslanjanja na primitak i unos 2FA koda, web stranice i aplikacije šalju korisniku obavijest da je u tijeku pokušaj autentikacije. Vlasnik uređaja pregledava pojedinosti obavijesti te može odobriti ili odbiti pristup jednim klikom.
- Biometrijska 2FA: Predstavlja oblik dvofaktorske autentikacije u kojem se korisnik tretira kao token. Ovo uključuje provjeru identiteta putem otisak prsta, prepoznavanja lica i sličnog [27].

Django nudi implementaciju dvofaktorskog mehanizma izgrađenog na temelju jednokratnih lozinki (*django-otp*) i *Djangova* ugrađenog radnog okvira za provjeru autentičnosti (*django.contrib.auth*) za pružanje lake integracije u većini *Django* projekata. Koraci instalacije i postavljanje dvofaktorske autentikacije u *Djangu* dano je u nastavku:

- Potrebno je instalirati paket pomoću *pip install django-two-factor-auth* naredbe
- Unutar *settings.py* datoteke pod *INSTALLED_APPS* potrebno je dodati:

```
'django_otp',
'django_otp.plugins.otp_static',
'django_otp.plugins.otp_totp',
'two_factor',
```

- Unutar *settings.py* datoteke pod MIDDLEWARE ispod *AuthenticationMiddleware-a* treba dodati *'django_otp.middleware.OTPMiddleware'*
- U *settings.py* datoteci treba postaviti:

```
LOGIN_URL = 'two_factor:login'
LOGIN_REDIRECT_URL = 'two_factor:profile'
```

- Unutar *urls.py* datoteke pod URL-ove treba dodati *path('', include(tf_urls))*

Ovime je najjednostavnije postavljanje dvofaktorskog mehanizma u *Django* projektu završeno.

3.3.7. Ostale sigurnosne mjere

U ovome potpoglavlju biti će opisane dodatne sigurnosne mjere koje pruža *Django* radni okvir. Biti će prikazan način na koji *Django* pohranjuje i štiti lozinke, koje je dodatne sigurnosne postavke potrebno uključiti unutar *settings.py* datoteke u projektu te koje je sigurnosne korake potrebno napraviti prije nego što se projekt napravljen u *Djangu* pusti u produkciju.

Upravljanje lozinkama je nešto što se ne bi trebalo nepotrebno prepravljati i shvaćati olako. *Django* pruža siguran i fleksibilan skup alata za upravljanje korisničkim lozinkama te prema zadanim postavkama koristi *PBKDF2* (funkcija derivacije ključa temeljena na lozinki) koja smanjuje rizike od napada grubom silom. Atribut lozinke se unutar User objekta u *Djangu* predstavlja sljedećom formom:

<algorithm>$<iterations>$<salt>$<hash>

To su komponente koje se koriste za pohranu korisničkih lozinki. *Django* prema zadanim postavkama koristi PBKDF2 u kombinaciji sa SHA256 hash funkcijom koju preporučuje NIST. Ovakav način upravljanja bi trebao zadovoljiti većinu korisnika jer je prilično siguran i potrebno je jako puno vremena uložiti da bi se ovakav način upravljanja lozinkama ugrozio. *Django* podržava upravljanje lozinkama i pomoću drugih funkcija kao što su: ARGON2, BCRYPT, SCRYPT te PBKDF2SHA1, kao i pisanje vlastitih funkcija koje odgovaraju specifičnim zahtjevima. Kod konfiguriranja ovoga unutar projekta treba biti oprezan te ukoliko ne postoji opravdan razlog za mijenjanje ovih postavki ili pisanja vlastitih funkcija, to se i ne bi trebalo onda raditi [28].

Nadalje biti će opisani dodatni koraci koje je potrebno napraviti prije nego što se projekt izrađen pomoću *Django* radnog okvira pusti u produkciju. Pod te korake spadaju:

- Sigurnosni propusti: *Django* pruža uvid u sigurnosne propuste koje naša mrežna aplikacija sadrži. Pokretanjem naredbe ***manage.py check -- deploy*** u terminalnu, dobiva se popis potencijalnih sigurnosnih propusta u aplikaciji na koje onda možemo obratiti pažnju.
- *The Mozilla Observatory*: Ova stranica također može pomoći u otkrivanju sigurnosnih propusta. Radi na način da skenira našu mrežnu aplikaciju i napravi izvještaj o sigurnosti naše stranice.
- CSRF zaštita: Nakon što se postavi HTTPS, potrebno je sljedeće linije koda dodati u *settings.py*:

  CSRF_COOKIE_SECURE = True
  SESSION_COOKIE_SECURE = True

- XSS zaštita: Za zaštitu protiv XSS napada treba dodati sljedeće linije koda u *settings.py* datoteku:

  SECURE_BROWSER_XSS_FILTER = True
  SECURE_CONTENT_TYPE_NOSNIFF = True

- Zaštita admin sučelja: Admin sučelje je jako korisna stvar za rad sa grupama, rolama i pravima korisnika unutar sustava. No putanja admin sučelja (URL) je nešto što je unaprijed poznato i postavljeno od strane Djanga. Dobra praksa bila bi promijeniti tu zadanu putanju u nešto što je samo korisniku koji pristupa admin sučelju poznato.
- SSL preusmjeravanje: Kako bismo osigurali da *Django* sve zahtjeve koji nisu HTTPS preusmjeri na HTTPS treba dodati sljedeću liniju koda u *settings.py*:

  SECURE_SSL_REDIRECT = True

- HTTP sigurnost prijenosa: Ukoliko ne dolazi do zadovoljavajućeg posluživanja HTTPS zahtjeva ili dođe do isteka certifikata stranice, preglednici će odbijati spajanje na web stranicu u određenom vremenskom periodu. Postavke unutar *settings.py* datoteke prikazane su u nastavku:

  SECURE_HSTS_SECONDS = 86400
  SECURE_HSTS_INCLUDE_SUBDOMAINS = True
  SECURE_HSTS_PRELOAD = True

Korištenjem spomenutih mehanizama napravljen je veliki korak naprijed u prevenciji najčešćih napada i poboljšanju sigurnosti mrežnih aplikacija izrađenih pomoću *Django* radnog okvira [29].

4. TESTIRANJE I OPIS IMPLEMENTIRANIH SIGURNOSNIH MEHANIZAMA

U ovome poglavlju biti će prikazano testiranje implementiranih sigurnosnih mehanizama u mrežnoj aplikaciji napravljenoj pomoću *Django* radnog okvira. U potpoglavlju 4.1. opisano je okruženje na kojem je testiranje mrežne aplikacije provedeno. U potpoglavlju 4.2. testirana je sama funkcionalnost sigurnosnih mehanizama i način rada mrežne aplikacije. U potpoglavlju 4.3. prikazano je sigurnosno testiranje mrežne aplikacije pomoću ZAP alata. I na kraju u potpoglavlju 4.4. uspoređeni su neki od najpopularnijih mrežnih radnih okvira po pitanju sigurnosnih mehanizama koje pružaju.

4.1. Okruženje za testiranje

Prilikom provođenja samog testiranja mrežne aplikacije korišteno je stolno računalo. Sva testiranja koja su provedena izvršena su lokalno na stolnom računalu čije su specifikacije prikazane u tablici 4.1.

Tablica 4.1. *Prikaz specifikacija stolnog računala za provođenje testiranja*

Procesor	Intel Core i7-6700
RAM	16 GB
SSD	ADATA SX8200PNP
Grafička kartica	Intel HD Graphics 530
Matična ploča	Gigabyte B150M-HD3-CF
Operacijski sustav	Windows 10 Pro

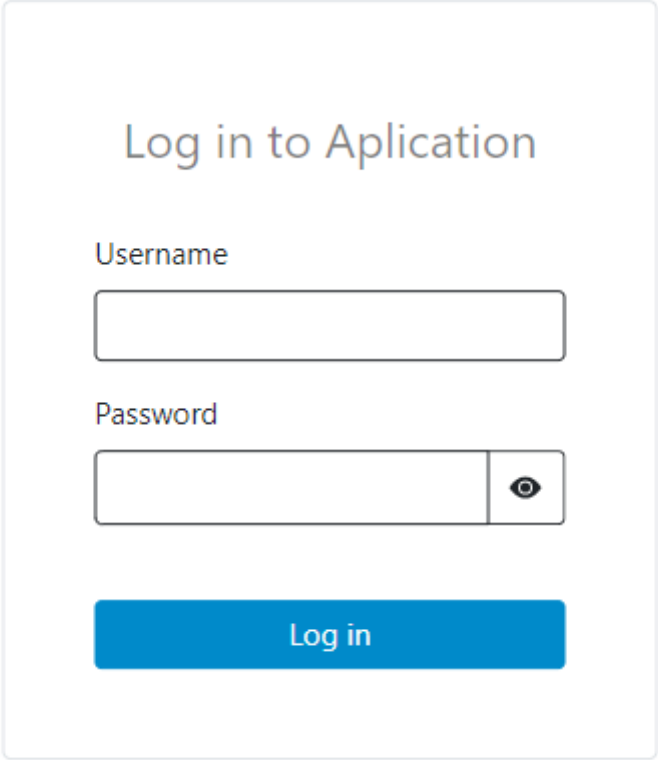
4.2. Testiranje funkcionalnosti implementiranih mehanizama

Unutar ovog potpoglavlja prikazan je način rada implementiranih mehanizama. Prikazan je mehanizam prijave, mehanizam preusmjeravanja korisnika na temelju uloge koju ima unutar sustava, prikazana su i ograničenja koja su postavljena pojedinoj grupi te na kraju mehanizam odjave iz aplikacije. Za potrebe testiranja kreirana su tri tipa korisnika od kojih svaki ima dodijeljene različite uloge unutar sustava.

Kreirani korisnici su:

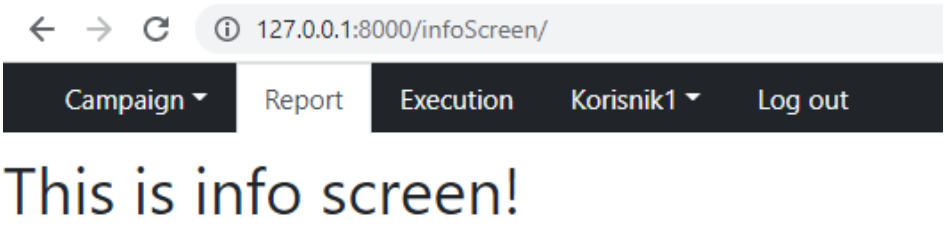
- Korisnik1 – ima ulogu menager-a, viewer-a i testera
- Korisnik2 – ima ulogu testera
- Korisnik3 – ima ulogu viewer-a

Na slici 4.1. prikazan je izgled sučelja za prijavu korisnika u aplikaciju.

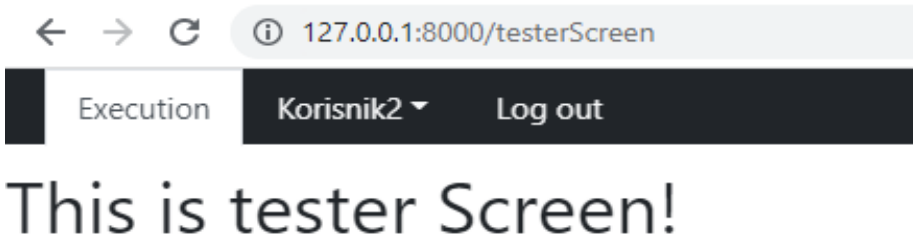


Slika 4.1. *Prikaz sučelja za prijavu korisnika*

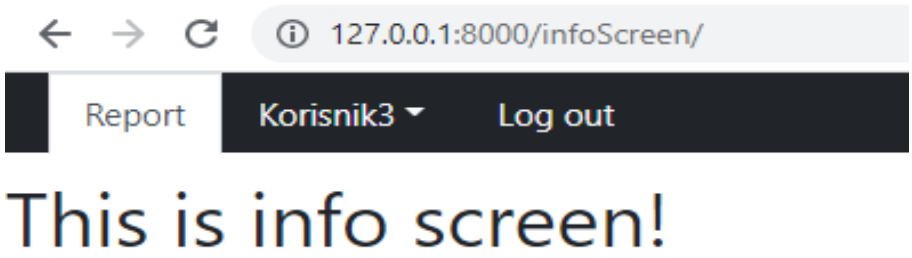
Nakon što se svaki od navedenih korisnika prijavi biti će preusmjeren ovisno o ulozi koja mu je dodijeljena. Ukoliko korisnik ima više dodijeljenih uloga, biva preusmjeren ovisno o važnosti same uloge. Primjerice, Korisnik1 je menager, viewer i tester, te će prvo biti preusmjeren na menager zaslon jer je uloga managera viša od uloge testera i viewera, no Korisnik1 će također moći pristupiti tester zaslonu. Također korisnici koji imaju ulogu viewer-a biti će preusmjereni na isti zaslon kao i manageri, ali neće imati dostupne sve funkcionalnosti koje imaju manageri u navigacijskoj traci. U nastavku su prikazani zasloni na koje svaki od korisnika bude preusmjeren nakon što obavi prijavu. Na slici 4.2. prikazan je izgled zaslona na koji će Korisnik1 biti preusmjeren nakon prijave. Na slici 4.3. prikazan je izgled zaslona kojemu će nakon obavljene prijave moći pristupiti Korisnik2. Na slici 4.4. prikazan je zaslon na koji će Korisnik3 biti preusmjeren nakon što obavi prijavu.



Slika 4.2. *Prikaz zaslona Korisnika1*

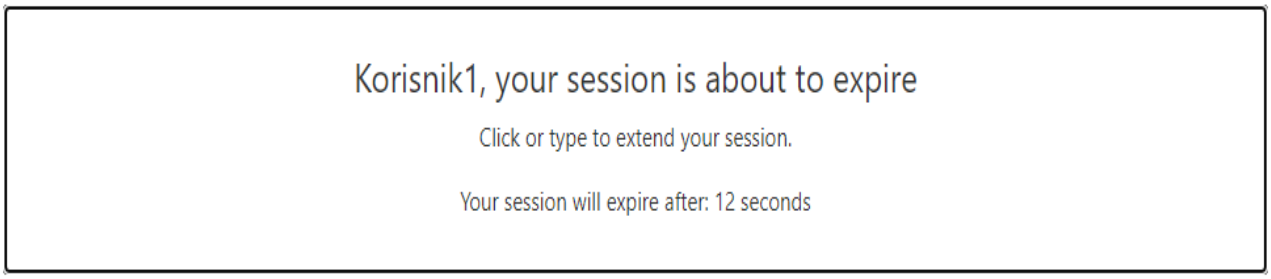


Slika 4.3. *Prikaz zaslona Korisnika2*



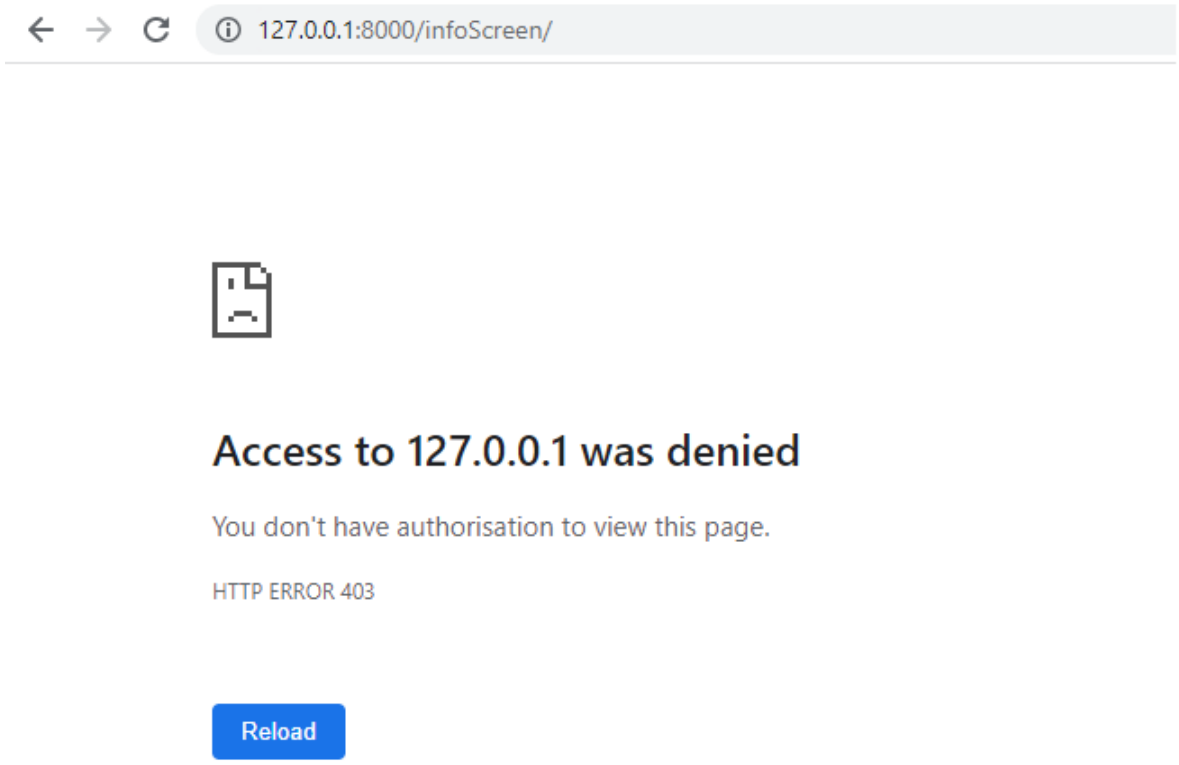
Slika 4.4. *Prikaz zaslona Korisnika3*

Nakon što prijavljeni korisnici provedu određeno vrijeme neaktivni, na zaslonu se pojavljuje poruka o isteku sesije kao što je prikazano na slici 4.5.



Slika 4.5. *Prikaz poruke o isteku sesije*

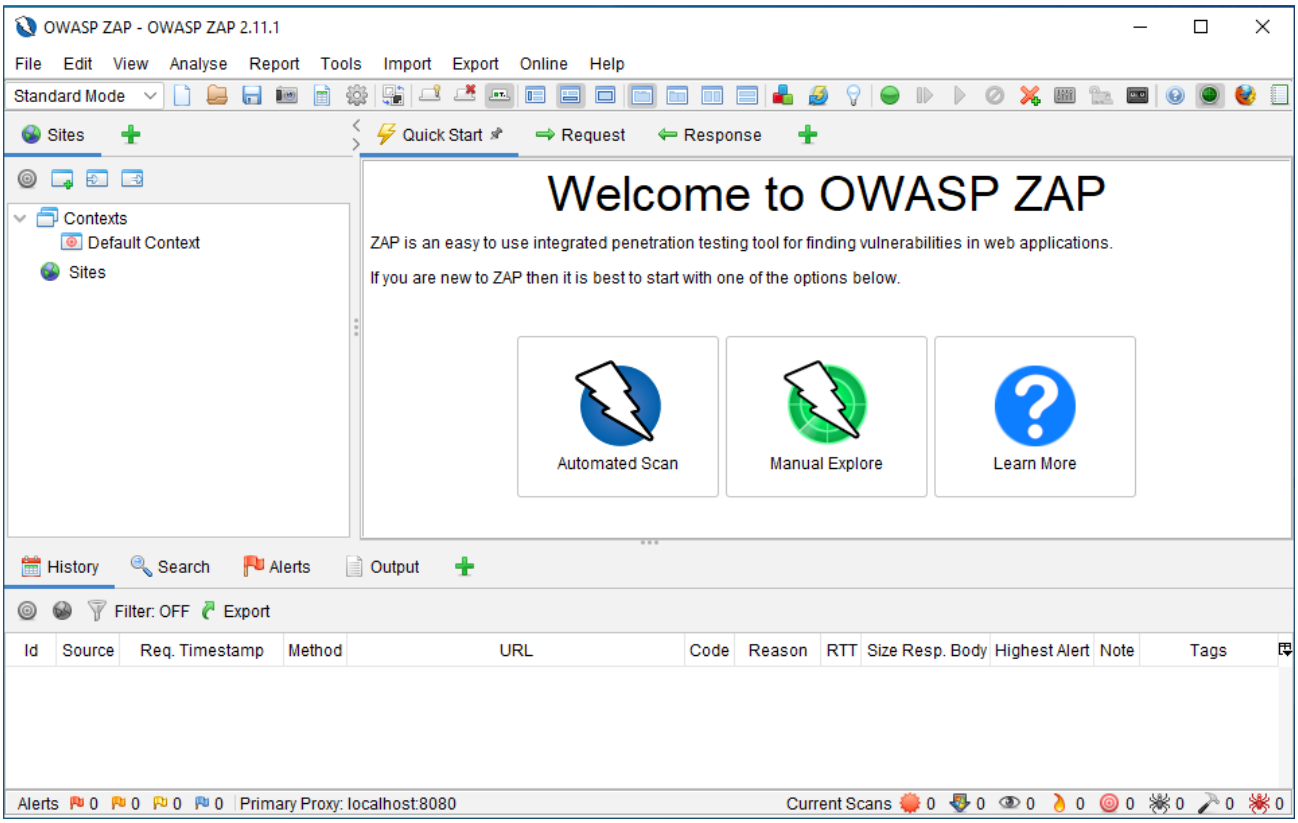
Nakon što prikazano vrijeme istekne, prijavljeni korisnik biva odjavljen iz aplikacije i preusmjeren na zaslon za prijavu. Ukoliko neki od korisnika, primjerice Korisnik2 koji je samo tester, sazna URL stranice kojoj nema pristup te proba pristupiti toj stranici, pristup će mu biti zabranjen te će o tome dobiti i adekvatnu poruku kao što je prikazano na slici 4.6.



Slika 4.6. *Prikaz poruke o zabrani pristupa određenoj stranici*

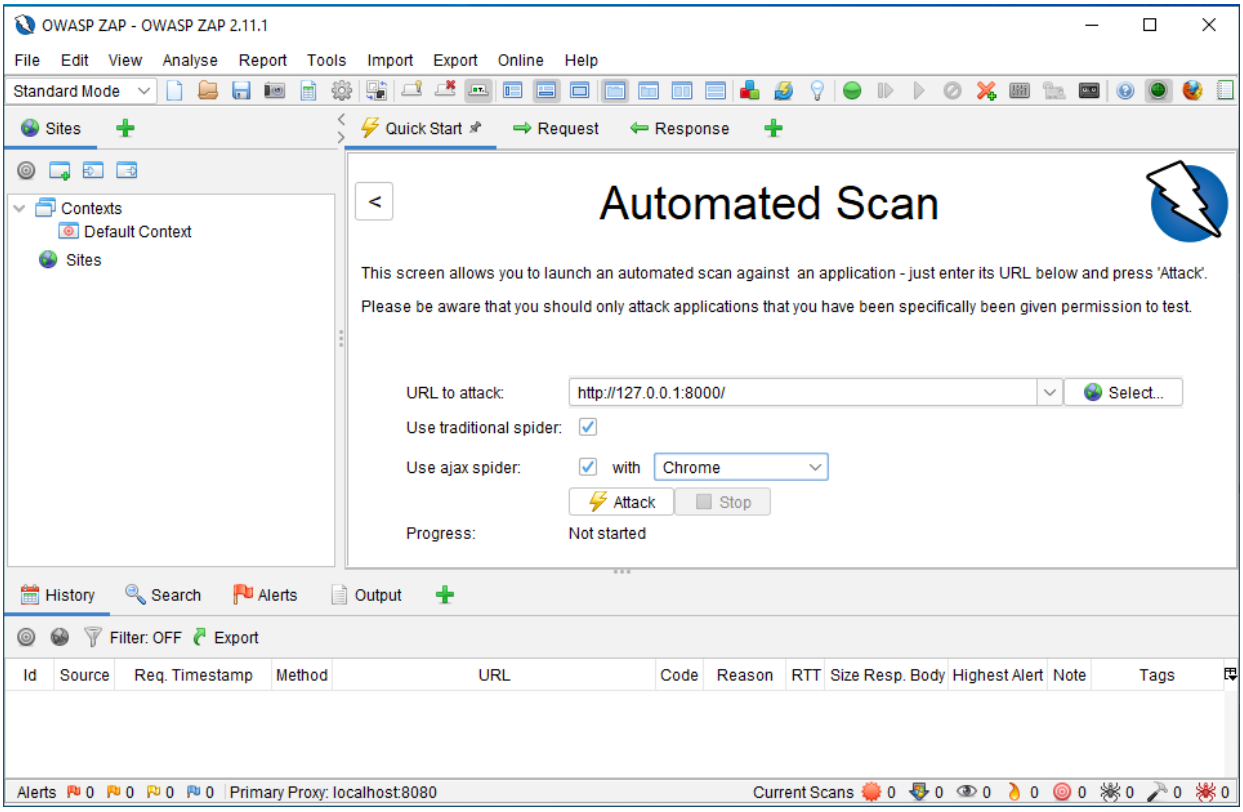
4.3. Testiranje mrežne aplikacije pomoću alata za ispitivanje sigurnosti

U ovome potpoglavlju prikazano je sigurnosno testiranje mrežne aplikacije pomoću ZAP-a. ZAP (engl. *Zed Attack Proxy*) predstavlja alat za testiranje mrežne aplikacije kojeg je razvio OWASP. Radi na svim operacijskim sustavima koji podržavaju *Javu 8*. Predstavlja jedan od najpopularnijih besplatnih alata koji se koristi za pronalaženje brojnih sigurnosnih propusta u mrežnoj aplikaciji. Glavne prednosti ZAP-a su to što je jednostavan za korištenje, jednostavno ga je postaviti, besplatan je te podržava više platformi. Omogućuje automatsko testiranje, prisilno pregledavanje, presretanje *proxya*, skenira mrežnu aplikaciju u potrazi za ranjivostima koje se tiču XSS napada te SQL ubrizgavanja i slično [30] Na slici 4.7. prikazan je izgled ZAP sučelja.



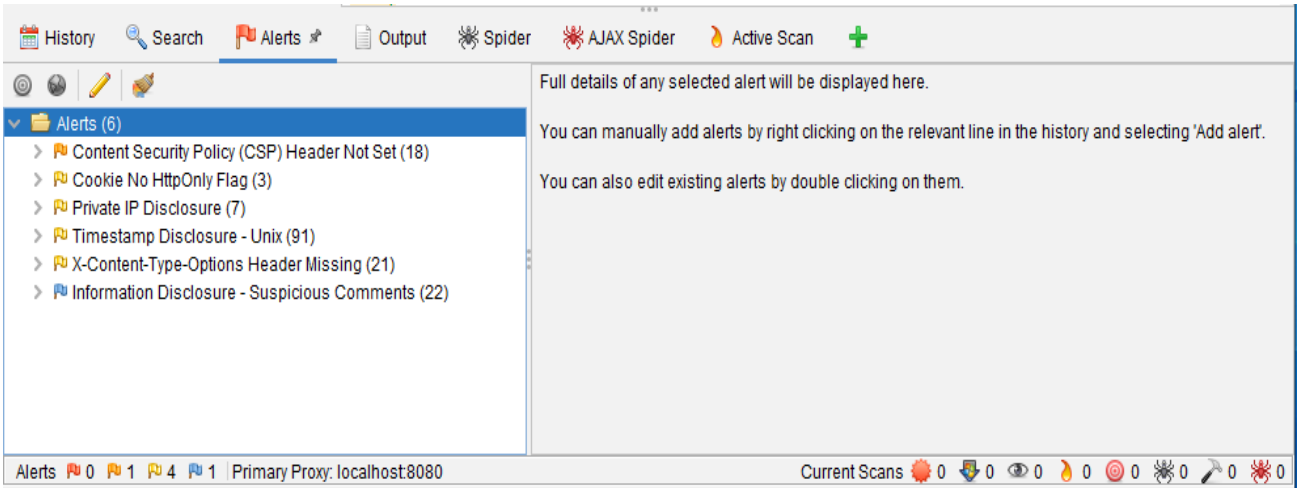
Slika 4.7. *Izgled ZAP sučelja*

Najjednostavniji način za korištenje ZAP-a je putem kartice *Quick Start*. *Quick Start* je dodatak koji omogućuje brzo i lagano skeniranje mrežne aplikacije putem automatskog skeniranja. Prikaz *Quick Start* kartice sa postavljenim parametrima za skeniranje prikazano je na slici 4.8.



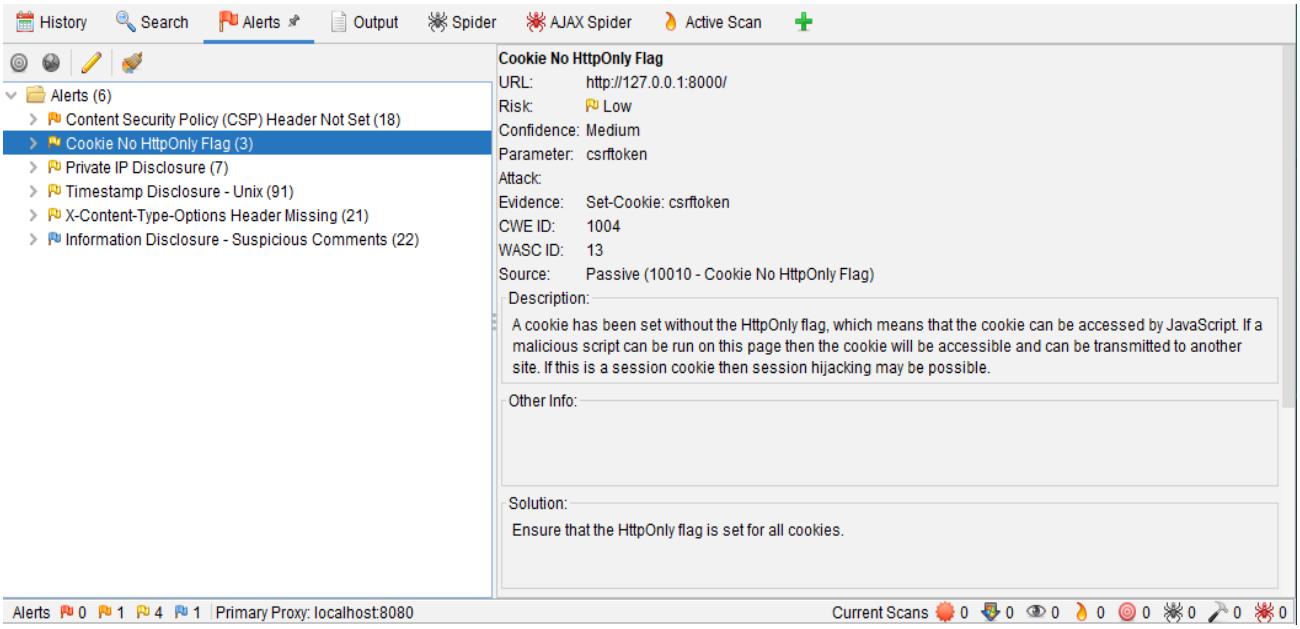
Slika 4.8. *Quick Start kartica sa postavljenim parametrima za testiranje*

Pritiskom na *Attack* tipku pokreće se automatsko testiranje navedene mrežne aplikacije. Nakon što se skeniranje dovrši dobiva se pregled potencijalnih rizika sa navedenim opisom koje je moguće pregledati pod karticom *Alerts* kao što je prikazano na slici 4.9.



Slika 4.9. *Prikaz upozorenja nakon provedenog skeniranja*

Pritiskom na bilo koje upozorenje otvara se prozor sa pojašnjenjem o kakvoj vrsti rizika se radi kao i rješenjem za navedenu rizičnu situaciju. Na slici 4.10. može se vidjeti detaljniji opis druge po redu prijetnje pod nazivom *Cookie No HttpOnly Flag*.



Slika 4.10. *Detaljan prikaz prijetnje*

Također je vidljivo, po boji zastavica, da ni jedna od pronađenih prijetnji ne spada u kategoriju visokog rizika. Na slici 4.11., na temelju boje zastavice, može se vidjeti kako ZAP definira i grupira određene rizike.

Slika 4.11. *Kategorije rizika u ZAP-u* [30]

U nastavku će biti prikazano kako je nedostatke otkrivene skeniranjem mrežne aplikacije pomoću ZAP-a moguće ispraviti. Pronađeni nedostaci su:

- *Content Security Policy Header Not Set*: *Content Security Policy Header* predstavlja HTTP zaglavlje koje moderni preglednici koriste za poboljšanje sigurnosti mrežne stranice. Zaglavlje omogućuje da se ograniči način na koji preglednik učitava resurse kao što su *JavaScript*, CSS ili bilo šta slično. CSP je prvotno osmišljen kako bi smanjio mogućnost XSS napada, a novije verzije štite i od drugih oblika napada kao što je krađa klikova. Kako bi se ovaj nedostatak riješio potrebno je instalirati *django-csp* pomoću *pip install django-csp* naredbe. Nakon toga unutar *MIDDLEWARE-a* treba dodati *'csp.middleware.CSPMiddleware'*. Unutar settings.py datoteke treba dodati i postaviti sljedeće linije koda:

  CSP_DEFAULT_SRC = ("'none'",)
  CSP_STYLE_SRC = ("'self'",)
  CSP_SCRIPT_SRC = ("'self'",)
  CSP_FONT_SRC = ("'self'",)
  CSP_IMG_SRC = ("'self'",)

  Na slici 4.12. može se vidjeti dio koda kojeg je ZAP označio kao potencijalnu prijetnju jer se nigdje ne vodi računa o učitavanju resursa kao što je u ovom slučaju CSS.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Page not found at /static</title>
  <meta name="robots" content="NONE,NOARCHIVE">
  <style type="text/css">
    html * { padding:0; margin:0; }
    body * { padding:10px 20px; }
    body * * { padding:0; }
    body { font:small sans-serif; background:#eee; color:#000; }
    body>div { border-bottom:1px solid #ddd; }
    h1 { font-weight:normal; margin-bottom:.4em; }
    h1 span { font-size:60%; color:#666; font-weight:normal; }
    table { border:none; border-collapse: collapse; width:100%; }
    td, th { vertical-align:top; padding:2px 3px; }
    th { width:12em; text-align:right; color:#666; padding-right:.5em; }
    #info { background:#f6f6f6; }
    #info ol { margin: 0.5em 4em; }
    #info ol li { font-family: monospace; }
    #summary { background: #ffc; }
    #explanation { background:#eee; border-bottom: 0px none; }
    pre.exception_value { font-family: sans-serif; color: #575757; font-size: 1.5em; margin: 10px 0 10px 0; }
  </style>
</head>
```

Slika 4.12. *Primjer koda kojeg je ZAP označio jer se ne poštuje politika o CSP zaglavljima*

Ukoliko se primjene preporučene mjere i ograniči se učitavanje vanjskih resursa kao što je prikazano kodom u nastavku, moguće je riješiti problem CSP zaglavlja kojeg je ZAP pronašao.

```
CSP_DEFAULT_SRC = ["'none'"]
CSP_SCRIPT_SRC = [
        "https://stackpath.bootstrapcdn.com",
        "https://cdn.jsdelivr.net",
        "https://code.jquery.com"
]
CSP_STYLE_SRC = ["https://stackpath.bootstrapcdn.com"]
CSP_IMG_SRC = ["'self'"]
CSP_FRAME_SRC = ["https://docs.google.com"]
```

Navedeni kod je samo primjer. Konfiguracija naravno ovisi o tome koji se vanjski resursi koriste. Primjenom ovoga, sve ugrađene skripte i stilovi više neće biti dopušteni, već moraju biti učitani iz izvora. Jako je bitno očistiti kod od svih ugrađenih stilova i skripti te prije implementacije treba proučiti dokumentaciju.

- *Cookie No HttpOnly Flag:* Ukoliko je kolačić postavljen bez oznake *HttpOnly*, to znači da mu se može pristupiti *JavaScript-om*. Ukoliko se zlonamjerna skripta može pokrenuti na stranici, kolačić je moguće prenijeti i na drugu stranicu. Ukoliko se radi o kolačiću sesije, moguće je i otimanje sesije. Kako bi kolačić bio postavljen kao *HttpOnly* potrebno je osigurati da klijent i poslužitelj budu na istoj domeni, inače *HttpOnly* neće biti postavljen. Na slici 4.13. vidljivo je upozorenje o tome da pojedini kolačići nisu postavljeni kao *HttpOnly*.

```
HTTP/1.1 200 OK
Date: Thu, 01 Sep 2022 11:28:56 GMT
Server: WSGIServer/0.2 CPython/3.9.0
Content-Type: text/html; charset=utf-8
X-Frame-Options: DENY
Vary: Cookie, Accept-Language
Content-Length: 5643
Content-Language: en
Strict-Transport-Security: max-age=30; includeSubDomains; preload
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Referrer-Policy: same-origin
Content-Security-Policy: default-src 'self'
Set-Cookie: csrftoken=t3nMLmYkOyoEtxRykvZ97AxrB0KV40fyXZEY0HeS9agHuyo4UpyDQl47HNBta2L2; expires=Thu, 31 Aug 2023
11:28:56 GMT; Max-Age=31449600; Path=/; SameSite=Lax; Secure
```

Slika 4.13. *Primjer kolačića koji nije označen kao HttpOnly*

Kako bi se ovaj nedostatak uklonio treba unutar *settings.py* datoteke dodati sljedeće linije koda:

SESSION_COOKIE_HTTPONLY = True
CSRF_COOKIE_HTTPONLY = True

- *Private IP Disclosure:* Privatne IP adrese poput (10.xxx, 172.xxx, 192.168.xx) pronađene su u tijelu HTTP odgovora. Ove informacije mogu biti korisne za daljnje napade usmjerene na unutarnje sustave. Kako bi se ovaj nedostatak riješio treba ukloniti privatne IP adrese iz tijela HTTP odgovora. Za komentare je poželjno koristiti JSP/ASP/PHP komentare umjesto HTML/JavaScript komentara koje klijentski preglednici mogu vidjeti. Na slici 4.14. može se vidjeti dio koda u kojem je ostala neobrisana IP adresa.

```
console.log(typeof reload);
if (reload=="true") {
    location.href="managerScreen?reload=false"; //primjer = 'http://10.11.12.13:2000';
}
```

Slika 4.14. *Primjer komentara u kojem je ostala neobrisana IP adresa*

- *TimeStamp Disclosure-Unix:* Ovaj nedostatak ukazuje na to da je aplikacija ili mrežni poslužitelj otkrio neki oblik vremenske oznake unutar koda koji potencijalno može ugroziti sigurnost mrežne aplikacije. Kako bi se ovaj nedostatak uklonio treba ručno provjeriti da podaci sa vremenskom oznakom nisu osjetljivi i da se ti podaci ne mogu iskoristiti za dohvaćanje osjetljivijih podataka i ozbiljnijeg ugrožavanja mrežne aplikacije. Na slici 4.15. može se vidjeti dio koda kojeg je ZAP označio jer sadrži vremensku oznaku koja potencijalno može biti problematična.

```
    padding: 12px 20px;
    color: #ffffff;
    display: inline-block;
    box-shadow: 0 3px 6px -1px rgba(0, 0, 0, 0.12), 0 10px 36px -4px rgba(77, 96, 232, 0.3);
    background: -webkit-linear-gradient(315deg, #73a5ff, #5477f5);
    background: linear-gradient(135deg, #73a5ff, #5477f5);
    position: fixed;
    opacity: 0;
    transition: all 0.4s cubic-bezier(0.215, 0.61, 0.355, 1);
    border-radius: 2px;
    cursor: pointer;
    text-decoration: none;
    max-width: calc(50% - 20px);
    z-index: 2147483647;
}

.toastify.on {
```

Slika 4.15. *Primjer koda kojeg je ZAP označio kao vremensku oznaku*

Na slici 4.16. može se vidjeti kako ZAP interpretira označeni dio koda.

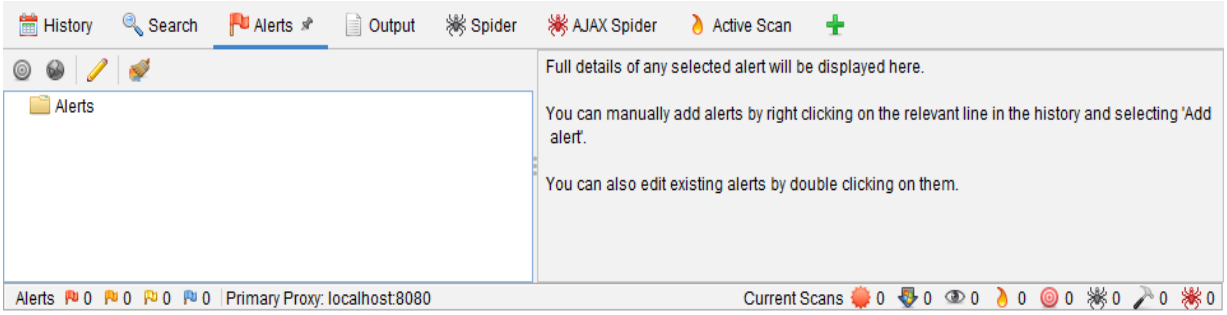
Slika 4.16. *Prikaz opisa pronađene prijetnje*

Sva upozorenja o otkrivanju vremenske oznake treba ručno pregledati kako bi se potvrdilo da otkriveni podaci nisu osjetljivi te da se ne koriste ni u kojem obliku za generiranje osjetljivih informacija na strani poslužitelja.

- *X-Content-Type-Options Header Missing: X-Content-Type-Options* nije postavljen na *'nosniff'*. To omogućuje starijim verzijama preglednika da izvrše MIME njuškanje na tijelu odgovora, potencijalno uzrokujući da se tijelo odgovora tumači i prikazuje kao vrsta sadržaja različita od deklarirane vrste sadržaja. Kada preglednik dobije odgovor od poslužitelja, on sam pokušava otkriti koja je vrsta sadržaja u pitanju i kako s njime postupati. U određenim okolnostima to može dovesti do ozbiljnih sigurnosnih problema poput XSS napada. Treba provjeriti postavlja li aplikacija ili poslužitelj odgovarajuće zaglavlje i postavlja li zaglavlje *X-Content-Type-Options* na *'nosniff'* za sve web stranice. Također treba osigurati da krajnji korisnik koristi moderan preglednik usklađen sa svim standardima. Unutar *settings.py* datoteke treba osigurati sljedeće:

  SECURE_BROWSER_XSS_FILTER = True
  SECURE_CONTENT_TYPE_NOSNIFF = True

Poštivanjem određenih pravila i korištenjem preporučenih mjera sigurnosti moguće je izbjeći dobar dio sigurnosnih propusta kao što je i prikazano na slici 4.17.



Slika 4.17. *Prikaz izvještaja u ZAP-u bez upozorenja*

4.4. Usporedba sigurnosti najpopularnijih mrežnih radnih okvira

Postoji jako velik broj programskih jezika i radnih okvira koji se koriste za izradu mrežnih aplikacija. Programeri trebaju u obzir uzeti jako puno čimbenika prije odabira radnog okvira u kojem će razvijati svoju mrežnu aplikaciju. Sigurnost je naravno jedan od tih čimbenika, posebno kada se razvija mrežna aplikacija koja će biti nužna za poslovanje određene organizacije. U ovome potpoglavlju biti će prikazano kako se razni radni okviri nose s nekim od najučestalijih napada i prijetnji na mrežne aplikacije. Odabranih pet mrežnih radnih okvira su: *Pythonov Django*, *Node.js*, *Ruby on Rails*, *ASP.NET Core* i *Java Spring*.

Tablica 4.2. *Prikaz svojstava i mogućnosti mrežnih radnih okvira*

Mrežni radni okvir	Svojstva	Mogućnosti
Django	Brzina, skalabilnost i sigurnost	Dobri zaštitni mehanizmi, dobar sustav za upravljanje korisnicima i njihovim pravima, dobar sustav za upravljanje lozinkama
Node.js	Skaliranje te brzo i jednostavno postavljanje aplikacije	Većina funkcionalnosti dolazi u obliku modula, mogućnost pokretanja aplikacije pomoću HTTPS-a, snažna biblioteka za kriptiranje podataka, jaka zajednica korisnika
Ruby on Rails	Snažna zajednica korisnika, brz i jednostavan razvoj mrežnih aplikacija	Dobra zaštita protiv SQL ubrizgavanja i XSS napada, koristi "dragulje" za pružanje funkcionalnosti, koristi sesije temeljene na kolačićima
ASP.NET Core	Najmlađi radni okvir, mnoštvo otvorenog koda za razvoj mrežnih aplikacija	Koristi Razor za prikaz stranice unutar preglednika, ugrađene mogućnosti autentikacije i autorizacije, robustan sustav provjere autentičnosti, jedan od jačih izbora po pitanju sigurnosti
Java Spring	Popularan izbor, nudi robusne alate za izradu mrežnih aplikacija, prilagodljiv	Dobar sustav autentikacije i autorizacije, popularan izbor, snažna pohrana i upravljanje "tajnama", nudi velik broj obrambenih mehanizama

Prvo će biti opisan *Django* kao radni okvir koji se koristio prilikom izrade mrežne aplikacije. *Django* je radni okvir namijenjen većini korisnika *Pythona*. Kao neke svoje glavne karakteristike *Django* navodi brzinu, skalabilnost i sigurnost. *Django* sigurnost shvaća jako ozbiljno. O SQL ubrizgavanju brine *Djangov* Queryset API koji vodi brigu o sprječavanju napada ubrizgavanjem. XSS je još jedan od uobičajenih napada o kojima *Django* vodi računa ukoliko se koriste *Djangovi* predlošci za izradu korisničkog sučelja. Također kada su u pitanju XSS napadi uključivanje ranije spomenutih postavki u *settings.py* datoteci može biti od velike pomoći. *Django* radi i odličan posao kada su u pitanju autentikacija i autorizacija korisnika. *Django* programerima nudi sredstva za upravljanje korisnicima sustava, kao i definiranje dopuštenja za svakog korisnika. Pruža mogućnost stvaranja grupa korisnika i primjene dopuštenja za pojedine grupe radi lakše administracije. *Django* pruža i dobar sustav za upravljanje lozinkama. Po zadanim postavkama koristi PBKDF2 u kombinaciji sa SHA256 hash funkcijom. *Django* prema zadanim postavkama pruža i zaštitu protiv CSRF napada. Sve što je potrebno je dodati "*csrf_token*" unutar elemenata obrasca bilo koje stranice. Iz svega navedenog može se zaključiti da je *Django* sigurnost stavio u fokus svoje implementacije, vodi brigu o uobičajenim sigurnosnim problemima i programeri bi se trebali osjećati sigurno koristeći ga, no i dalje postoji nekoliko manjih problema. Iako radi dobar posao u sprječavanju SQL ubrizgavanja, i dalje postoje funkcije koje omogućuju stvaranje sirovih SQL upita. Također XSS zaštita koju pružaju predlošci nije 100% sigurna. Problem koji se još javlja je upravljanje sesijom poddomena. Poddomene unutar stranice mogu postaviti kolačiće za cijelu domenu, što može dovesti do napada fiksiranja sesije. Zadane postavke unutar *Djanga* mogu biti zbunjujuće u načinu na koji rade, te je bitno sve jako dobro proučiti.

Sljedeći na redu je *Node.js* koji je vrlo popularan radni okvir za *JavaScript* aplikacije. Pruža izvrsne mogućnosti skaliranja te omogućuje brzo postavljanje i pokretanje aplikacije. Pošto većina funkcionalnosti u *Node-u* dolazi u obliku modula koji se dodaju sustavu prema potrebi, takav model razvoja ima i zanimljive sigurnosne posljedice. *Node* dolazi s mogućnošću pokretanja mrežne aplikacije pomoću HTTPS-a. Također nudi i snažnu biblioteku za kriptiranje podataka, tako da se podaci mogu zaštiti dok su u mirovanju. Posljedica minimalističkog modela je ovisnost o zajednici suradnika za pružanje dobre sigurnosne funkcionalnosti. Također potrebne su brojne vanjske biblioteke kako bi se spriječilo nekoliko najčešćih rizika kod razvoja, poput izlaganja osjetljivih podataka, neispravne autentikacije i SQL ubrizgavanja. Zajednica *Node* korisnika je jaka, te postoji velik broj dobro napisanih modula za sigurnosni razvoj aplikacije koje bi programeri trebali koristiti. *Node* ima mnogo sjajnih značajki, no ovisnost o zajednici programera može dovesti do izravnih napada ili korištenja nesigurnog koda. Svakako treba biti oprezan prije nego što se tuđi kod iskoristi u razvoju aplikacije.

Ruby on Rails je mrežni radni okvir po izboru *Ruby* programera. Ima snažnu zajednicu korisnika i cilj mu je učiniti razvoj mrežnih aplikacija što bržim i jednostavnijim. Pruža čvrstu zaštitu protiv SQL ubrizgavanja kao i protiv XSS napada koji su onemogućeni prema zadanim postavkama. Koristi i odgovarajuća sigurnosna

zaglavlja koja štite aplikacije od napada poput krađe klikova. *Ruby* koristi "dragulje" za pružanje dodatnih funkcionalnosti. Jedan od njih je i *Devise* koji pruža robusnu funkciju provjere autentičnosti. Još jedan dragulj, *Pundit*, ima za zadaću definirati kontrolu pristupa na razini resursa. Ova ovisnost o draguljima može se smatrati nedostatkom, no ova dva navedena su provjerena od strane OWASP-a. Prema zadanim postavkama *Rails* koristi sesije koje se temelje na kolačićima. Sesije neće isteći na poslužitelju, što aplikaciju ostavlja otvorenom za razne napade. Naravno ovo je moguće popraviti mijenjanjem konfiguracije. Rails ima neke nedostatke, ali još uvijek je solidan radni okvir kada je riječ o sigurnosnim postavkama.

ASP.NET Core je najmlađi radni okvir od svih ovdje spomenutih. Objavljen je od strane Microsofta te nastoji pružiti i omogućiti mnoštvo otvorenog koda visokih performansi za izradu mrežnih aplikacija. Koristi moćni Razor za prikaz stranice unutar preglednika. Razor kodira HTML prema zadanim postavkama, čime se sprječavaju XSS napadi. ORM rukuje sa SQL ubrizgavanjem te ga sprječava. Autentikacija i autorizacija su ugrađeni u *ASP.NET Core*. Pruža se robustan sustav provjere autentičnosti koji može koristiti korisničko ime i lozinku s bazom podataka ili nekim vanjskim pružateljem usluga kao što su Facebook ili Google. Ugrađene biblioteke pružaju mogućnost ograničavanja pristupa na temelju uloga korisnika. Još jedna korisna značajka *ASP.NET-a* je *API* za zaštitu podataka koji pruža programerima izvrsnu apstrakciju za šifriranje podataka u mirovanju. *ASP.NET* je od početka rađen imajući na umu sigurnost, te se ovdje ne može pronaći jako puno mana i nedostataka. Koristeći zadane postavke u kombinaciji s korisnim alatima, *ASP.NET* je jedan od jačih izbora po pitanju sigurnosti.

Java Spring nudi robusne alate za izradu mrežnih aplikacija te je popularan izbor od strane velikog broja organizacija. *Spring Security* je dodatak Spring aplikacijama koji se koristi za autentikaciju i autorizaciju. Vrlo je prilagodljiv i korišten od strane velikog broja poslovnih aplikacija. *Spring Vault* je još jedan dodatak koji pruža snažnu pohranu i upravljanje raznim "tajnama". Tajne mogu predstavljati ključeve za šifriranje ili korisnička imena i lozinke koje se koriste za pristup raznim uslugama. XSS napadi riješeni su od strane Spring-ove izvorne metode *HtmlUtils.htmlEscape*. Ova metoda nastoji izbjeći nizove koji se vraćaju u korisničko sučelje, te ju se svakako treba koristiti. Spring također ima nekoliko nedostataka kada su u pitanju uobičajeni sigurnosni problemi. SQL ubrizgavanje predstavlja najveći problem kada je u pitanju *Spring*, budući da se upiti moraju parametrizirati pomoću prilagođenog koda, ili se treba koristiti zaseban ORM, kao što je *Hibernate*. *Hibernate* štiti većim dijelom, ali još uvijek može biti ranjiv na ubrizgavanje putem svog HQL jezika. *Java Spring* dolazi s mnogo ugrađenih obrambenih mehanizama, te predstavlja dobar odabir za Java projekte [31].

Svi navedeni mrežni radni okviri nude dobre mehanizme zaštite od najčešćih prijetnji na mrežne aplikacije. Programeri često mogu biti vođeni pogrešnim mentalitetom poput "radni okvir će me zaštititi". To posebno dovodi do problema kada programeri, bilo slučajno ili namjerno, zaobilaze zadane i predložene sigurnosne značajke koje im radni okviri pružaju. U takvim situacijama, programeri će biti odgovorni za bilo

kakve sigurnosne propuste jer su radili na svoju ruku, umjesto da su koristili predložene sigurnosne značajke. Programeri bi svakako trebali biti pravilno obučeni u radu sa pojedinim radnim okvirom, te prilikom razvoja koristiti sigurnosne dokumentacije za pojedini radni okvir koje su lako dostupne.

5. ZAKLJUČAK

Kada se u obzir uzme činjenica da veliku većinu OWASP-ove liste najčešćih napada čine napadi i prijetnje koji su na listi bili prije pet i više godina, može se zaključiti da ti napadi predstavljaju ozbiljne prijetnje na sigurnost gotovo svih mrežnih aplikacija. Mehanizmi zaštite mrežnih aplikacija razvijaju se iz dana u dan, ali kako se razvijaju mehanizmi zaštite, pojavljuju se i nove prijetnje o kojima treba voditi računa. Stoga je bitno da se sigurnosno testiranje provodi i nakon što mrežna aplikacija bude puštena u rad, a ne samo tijekom njenog razvoja i implementacije. U ovom diplomskom radu, opisane su najčešće prijetnje na sigurnost mrežnih aplikacija te su dane smjernice kako te prijetnje ublažiti i ukloniti. S obzirom na činjenicu da softver pišu ljudi, nerealno je očekivati da kreirane mrežne aplikacije budu bez nedostataka. Samim provođenjem sigurnosnog testiranja može se otkriti jako velik broj sigurnosnih nedostataka i mana koje je potrebno ispraviti kako bi mrežna aplikacija bila sigurna za korištenje. *Django* je jedan od alata koji nudi jako puno po pitanju sigurnosnih mehanizama, te se upravo on koristio za izradu mrežne aplikacije. Primjena i opis sigurnosnih mehanizama koje pruža *Django* također je bio jedan od zadataka ovog diplomskog rada. *Django* pruža jako širok raspon sigurnosnih mehanizama od kojih su najvažniji bili korišteni prilikom izrade mrežne aplikacije. Izrada mrežne aplikacije uključivala je izradu funkcija za prijavu i odjavu korisnika, kreiranje uloga korisnika i njihovih prava unutar sustava, postavljanje LDAP direktorija, rad sa sesijom neaktivnih korisnika, korištenje dekoratora i konfiguraciju raznih sigurnosnih postavki unutar aplikacije. Zatim je provedeno sigurnosno testiranje pomoću ZAP alata i dobiven je popis sigurnosnih nedostataka unutar aplikacije. Također je opisano i na koji način je pronađene prijetnje moguće ukloniti. Većina pronađenih prijetnji bila je niže "kategorije", odnosno opasnosti, dok je jedna prijetnja bila srednje opasnosti. Iz toga je jasno vidljivo da u obzir uvijek treba uzeti što širi raspon sigurnosnih mehanizama te ih dobro proučiti i implementirati jer i oni najmanji i na prvi pogled nebitni sigurnosni propusti napadačima mogu poslužiti kao ulaznica u sustav koju će iskoristiti kako bi opasno naštetili mrežnoj aplikaciji.

LITERATURA

[1] What is Software Testing?, https://www.guru99.com/software-testing-introduction-importance.html#5 [4.kolovoz 2022.]

[2] What is Functional Testing? Types & Examples, https://www.guru99.com/functional-testing.html [4. kolovoz 2022.]

[3] What is Non Functional Testing? Types with Example, https://www.guru99.com/non-functional-testing.html [4. kolovoz 2022.]

[4] What is Maintenance Testing?, http://tryqa.com/what-is-maintenance-testing/ [4. kolovoz 2022.]

[5] R. Lepofsky, The Manager's Guide to Web Application Security, Apress, CA, 2014.

[6] 6 Security Testing Methodologies Explained: Definitions, Processes, Checklist, https://www.getastra.com/blog/security-audit/security-testing-methodologies-explained/ [6. kolovoz 2022.]

[7] Software Security Testing: Definition, Types & Tools, https://www.getastra.com/blog/security-audit/software-security-testing/ [6. kolovoz 2022.]

[8] What is Black-box Security Testing?, https://www.acunetix.com/blog/articles/black-box-security-testing/ [6. kolovoz 2022.]

[9] Microsoft Security Development Lifecycle (Microsoft SDL), https://www.techopedia.com/definition/23582/microsoft-security-development-lifecycle-microsoft-sdl [6.kolovoz 2022.]

[10] CLASP, https://owasp.org/www-pdf-archive/Us_owasp-clasp-v12-for-print-lulu.pdf [7.kolovoz 2022.]

[11] 11 Best Practices for Developing Secure Web Applications, https://www.lrswebsolutions.com/Blog/Posts/32/Website-Security/11-Best-Practices-for-Developing-Secure-Web-Applications/blog-post/ [7. kolovoz 2022.]

[12] A. Hoffman, Web Application Security, O'Reilly Media, 2020.

[13] SQL injection, https://portswigger.net/web-security/sql-injection [9. kolovoz 2022.]

[14] 10 Common Web Application Vulnerabilities and How to Prevent Them, https://scand.com/company/blog/10-common-web-application-security-vulnerabilities-and-how-to-prevent-them/ [9.kolovoz 2022.]

[15] Cross-site scripting, https://portswigger.net/web-security/cross-site-scripting [9. kolovoz 2022.]

[16] Introduction to the DOM, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction [9. kolovoz 2022.]

[17] 10 Common Web Application Security Vulnerabilities and How to Prevent Them, https://relevant.software/blog/web-application-security-vulnerabilities/#Cross-Site_Request_Forgery [9. kolovoz 2022.]

[18] Cross-site request forgery (CSRF), https://portswigger.net/web-security/csrf [9. kolovoz 2022.]

[19] H. Wu, L. Zhao, Web Security A WhiteHat Perspective, Auerbach Publications, New York, 2015.

[20] Clickjacking, https://portswigger.net/web-security/clickjacking [10.kolovoz 2022.]

[21] 5 application security threats and how to prevent them, https://www.techtarget.com/searchsecurity/feature/How-to-mitigate-5-persistent-application-security-threats [10. kolovoz 2022]

[22] Broken Access Control, https://owasp.org/Top10/A01_2021-Broken_Access_Control/ [10. kolovoz 2022.]

[23] What Is Broken Authentication?, https://auth0.com/blog/what-is-broken-authentication/ [12.kolovoz 2022.]

[24] Security Misconfiguration: Impact, Examples, and Prevention, https://brightsec.com/blog/security-misconfiguration/ [12. kolovoz 2022.]

[25] 10 Common Web Security Vulnerabilities, https://www.toptal.com/security/10-most-common-web-security-vulnerabilities [12. kolovoz 2022.]

[26] What is Sensitive Data Exposure Vulnerability & How to avoid it?, https://securiti.ai/blog/sensitive-data-exposure/ [23. kolovoz 2022.]

[27] What Is Two-Factor Authentication (2FA)?, https://authy.com/what-is-2fa/ [23.kolovoz 2022.]

[28] Password management in Django, https://docs.djangoproject.com/en/4.0/topics/auth/passwords/ [25. kolovoz 2022.]

[29] Django web security checklist before deployment, https://dev.to/thepylot/django-web-security-checklist-before-deployment-secure-your-django-app-4jb8 [25.kolovoz 2022.]

[30] ZAP Getting Started, https://www.zaproxy.org/getting-started/ [25.kolovoz 2022.]

[31] How Secure Are Popular Web Frameworks? Here Is a Comparison, https://www.veracode.com/blog/secure-development/how-secure-are-popular-web-frameworks-here-comparison [25. kolovoz 2022.]

SAŽETAK

Jedna od najbitnijih stvari kod razvoja svake mrežne aplikacije je briga o sigurnosti mrežne aplikacije. Postoji jako velik broj oblika mrežnih napada i prijetnji koje mogu ugroziti mrežne aplikacije, stoga je primjena odgovarajućih sigurnosnih mehanizama od velike važnosti. Kroz ovaj diplomski rad opisana je važnost provođenja testiranja softvera. Poseban je naglasak stavljen na sigurnosno testiranje softvera. Također opisane su najčešće prijetnje na sigurnost mrežnih aplikacija kao i načini na koje ih je moguće spriječiti. Opisani su i implementirani sigurnosni mehanizmi koje pruža *Django* radni okvir, te je za potrebe testiranja kreirana mrežna aplikacija pomoću *Djanga*. Na kraju je provedeno testiranje implementiranih funkcionalnosti kao i sigurnosno testiranje mrežne aplikacije pomoću ZAP alata.

Ključne riječi: sigurnost mrežne aplikacije, testiranje softvera, sigurnosno testiranje softvera, Django radni okvir, sigurnosni mehanizmi, prijetnje mrežnim aplikacijama, ZAP

ABSTRACT

Application of security mechanisms in the development of web applications for the automotive industry

One of the most important things in the development of any web application is to take care of the security of the network application. There are a very large number of forms of network attacks and threats that can threaten network applications, therefore the application of appropriate security mechanisms is of great importance. This masterwork describes the importance of software testing. Special emphasis is placed on software security testing. The most common threats to the security of web applications are also described, as well as ways to prevent them. The security mechanisms provided by the Django framework were described and implemented, and a web application using Django was created for testing purposes. At the end, testing of the implemented functionalities was carried out, as well as security testing of the web application using the ZAP tool.

Keywords: web application security, software testing, software security testing, Django framework, security mechanisms, web applications threats, ZAP

ŽIVOTOPIS

Josip Živković rođen je 12.05.1998. godine u Virovitici. Odrastao je u Virovitici gdje je i pohađao osnovnu školu. Srednju školu, odnosno matematičku gimnaziju pohađao je u Virovitici. Kroz četverogodišnje školovanje u srednjoj školi je ostvarivao odličan uspjeh. Akademske 2017./2018. godine upisao je Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, gdje je nakon upisa u drugu akademsku godinu odabrao smjer Komunikacije i informatika. Godine 2020. stječe akademski naziv sveučilišni prvostupnik inženjer elektrotehnike i informacijske tehnologije. Nakon toga upisuje diplomski sveučilišni studij automobilskog računarstva i komunikacija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.

Potpis:

_____