Analiza razvoja web aplikacije uz primjenu sigurnosnih mehanizama i sigurnosnog testiranja

Žuvić, Darian

Professional thesis / Završni specijalistički

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:957471

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-06-30**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Darian Žuvić

ANALIZA RAZVOJA WEB APLIKACIJE UZ PRIMJENU SIGURNOSNIH MEHANIZAMA I SIGURNOSNOG TESTIRANJA

SPECIJALISTIČKI RAD

Zagreb, 2023.

UNIVERSITY OF ZAGREB

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Darian Žuvić

ANALYSIS OF WEB APPLICATION DEVELOPMENT WITH THE APPLICATION OF SECURITY MECHANISMS AND SECURITY TESTING

SPECIALIST THESIS

Zagreb, 2023.

Specijalistički rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva u sklopu poslijediplomskog specijalističkog studija Informacijska sigurnost.

Mentor: izv. prof. dr. sc. Marin Vuković

Specijalistički rad ima: 186 stranica

Specijalistički rad br.:_____

Povjerenstvo za ocjenu u sastavu:

1. izv. prof. dr. sc. Miljenko Mikuc – predsjednik

2. izv. prof. dr. sc. Marin Vuković – mentor

3. izv. prof. dr. sc. Toni Perković, Sveučilište u Splitu Fakultet elektrotehnike, strojarstva i brodogradnje - član

Povjerenstvo za obranu u sastavu:

1. izv. prof. dr. sc. Miljenko Mikuc – predsjednik

2. izv. prof. dr. sc. Marin Vuković – mentor

3. izv. prof. dr. sc. Toni Perković, Sveučilište u Splitu Fakultet elektrotehnike, strojarstva i brodogradnje - član

Datum obrane: 10. siječnja 2024.

Sažetak

Specijalistički rad obuhvaća područje sigurnosti web aplikacija te obuhvaća pregled tehnologija koje se koriste u razvoju web aplikacije i analizu primijenjenih arhitektonskih obrazaca. Posebna pažnja se pridaje analizi penetracijskog testiranja web aplikacija. Provodi se analiza sigurnosnih mehanizama i njihova primjena na konkretnom primjeru web aplikacije.

Primjer sigurne web aplikacije se razvija korištenjem „*Laravel okvira*“. Pritom je izložena detaljna analiza sustava za autentifikaciju korisnika, ugrađenog u navedeni okvir.

Zaključak je kako „*Laravel okvir*“ predstavlja kvalitetno rješenje koje je prikladno koristiti prilikom razvoja sigurne web aplikacije.

U radu je izvršena detaljna analiza dvaju najpopularnijih alata za penetracijsko testiranje web aplikacija: „*Burp Suite*“ i „*OWASP ZAP okvir*“.

Ključne riječi

SSDLC, Laravel, sustav za autentifikaciju korisnika, sigurnost web aplikacija, penetracijsko testiranje web aplikacija, Burp Suite, OWASP ZAP, metode napada na web aplikacije, ranjivosti web aplikacija, modeliranje prijetnji, sigurnosni mehanizmi web aplikacija, arhitektonski obrasci

Abstract

Specialist thesis covers the area of web application security. In doing so, special attention is paid to the analysis of web application penetration testing. An analysis of security mechanisms and their application was carried out on a concrete example of a web application. The specialist work also includes an overview of the technologies used in the development of the web application and the analysis of the applied architectural patterns.

An example of a secure web application is developed using the Laravel framework. In doing so, a detailed analysis of the system for user authentication, built into the aforementioned framework, is presented. It is concluded that the "Laravel framework" represents a quality solution that is appropriate to use when developing a secure web application. As part of the work, a detailed analysis of the two most popular tools for penetration testing of web applications was also performed. This refers to analysis of Burp Suite and OWASP ZAP Framework.

Keywords

SSDLC, Laravel, User Authentication System, Web Application Security, Web Application Penetration Testing, Burp Suite, OWASP ZAP, Web Application Attack Methods, Web Application Vulnerabilities, Threat Modeling, Web Application Security Mechanisms, Architectural Patterns

SADRŽAJ

1 UVOD

Trendovi ukazuju na kontinuirani rast vrijednosti informacijske i komunikacijske imovine organizacija[1] te paralelni razvoj sve sofisticiranijih oblika napada i sigurnosnih mehanizama[2]. Zaostajanje u prilagodbi novim sigurnosnim standardima može rezultirati značajnim troškovima za organizaciju. U skladu s tim se javlja potreba za ulaganjem u sigurnost imovine odnosno primjenu robusnih sigurnosnih mehanizama te njihovo konstantno unaprjeđenje.

U ovom Specijalističkom radu je opisan proces sigurnog razvoja web aplikacije odnosno proces razvoja koji podrazumijeva primjenu sigurnosnih mehanizama. Pri tome se dokumentiraju ključni koncepti vezani uz korištene tehnologije i arhitektonske obrasce. Također se analizira proces penetracijskog testiranja, koji se primjenjuje u kasnijim fazama životnog ciklusa razvoja softvera, a koji omogućuje dodatnu verifikaciju sigurnosti analiziranog sustava.

Struktura rada je koncipirana na način da omogući uvid u teoretske osnove i konkretnu primjenu sigurnosnih koncepata.

Rad započinje s pregledom sigurnog životnog ciklusa razvoja softvera („*SSDLC*“) i izdvojenog skupa napada. Prilikom navedene analize se upućuje na važnost ugradnje sigurnosnih procedura u sklopu svih faza životnog ciklusa razvoja softvera. Na ovaj način se omogućuje ostvarenje nižih dugoročnih troškova vezanih uz održavanje sigurnosti sustava. U sklopu analize pojedinog oblika napada se, uz opis tehnika vezanih uz potencijalnu eksploataciju web aplikacije, izdvaja i skup primjenjivih mehanizama obrane.

Slijedi detaljna analiza te usporedba funkcionalnosti dvaju najpopularnijih okvira za penetracijsko testiranje web aplikacija. U sklopu navedene analize su dokumentirani ključni koncepti vezani uz proces penetracijskog testiranja. Navedeni koncepti se prezentiraju na precizan i jasan način. Pri tome se nastoje ublažiti uočeni nedostaci pojedinih dijelova originalne dokumentacije vezane uz analizirane okvire.

[1] Temeljeno na sljedećem online sadržaju: https://nwncarousel.com/news-room/news/gartner-top-5-markets-in-4-6-trillion-tech-industry-for-2023-2024/

[2] Temeljeno na sljedećim online sadržajima: https://krontech.com/gartners-8-cybersecurity-predictions-for-2023-2025 i https://www.gartner.com/en/articles/top-strategic-cybersecurity-trends-for-2023

U radu su izložena tri poglavlja u sklopu kojih se opisuju ključne sigurnosne procedure pojedinih faza životnog ciklusa razvoja softvera iz praktične perspektive. Ovdje je riječ o demonstraciji konkretnog primjera procesa modeliranja prijetnji, sigurne implementacije i penetracijskog testiranja web aplikacije. Pri tom se koristi primjer sigurne web aplikacije razvijene korištenjem „*Laravel okvira*“.

2 SIGURAN ŽIVOTNI CIKLUS RAZVOJA SOFTVERA

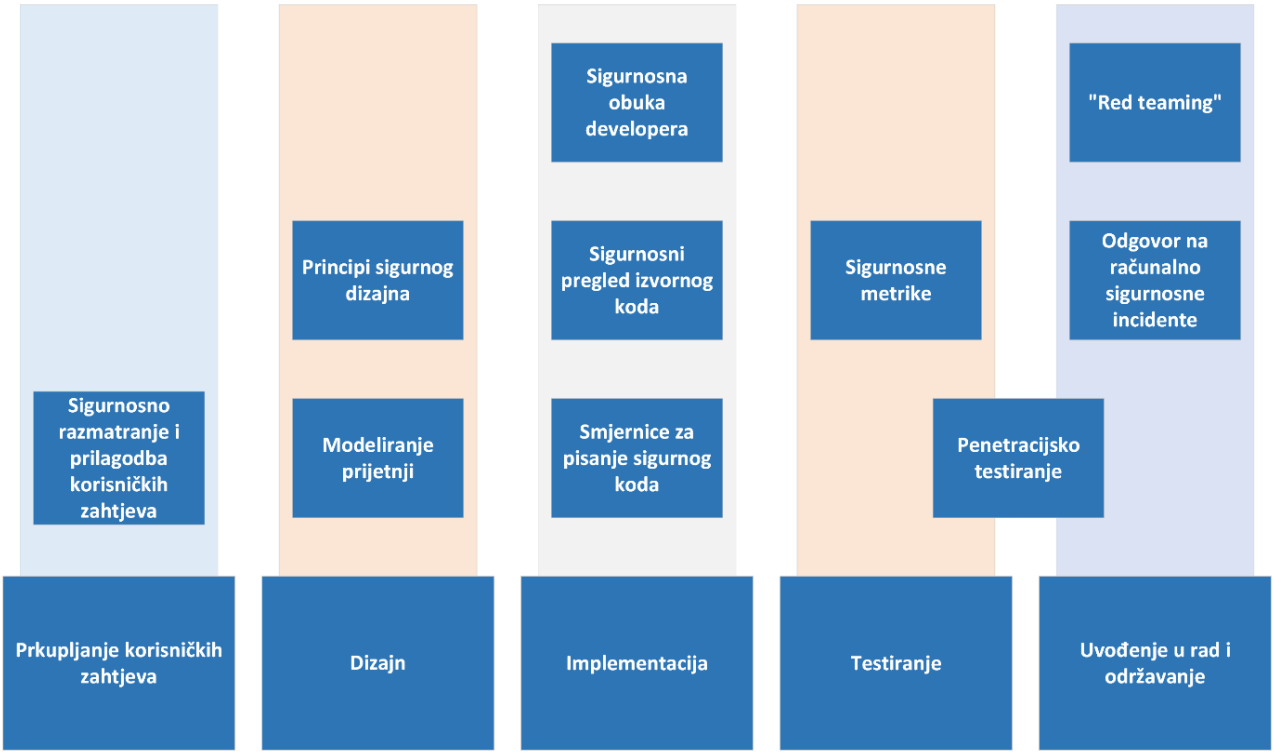
Životni ciklus razvoja softvera („*SDLC*“) predstavlja metodologiju razvoja softvera. Ona podrazumijeva definiciju ključnih faza razvoja softvera te očekivanih rezultata vezanih uz izvršenje svake od navedenih faza. „*SDLC*“ je struktura koju prati razvojni tim unutar softverske organizacije[3]. Primjena strukturiranog procesa razvoja softvera u pravilu omogućuje minimiziranje rizika, troškova te vremena potrebnog za izradu projekta. Svaka organizacija oblikuje „*SDLC*“ u ovisnosti o njihovim specifičnim zahtjevima.

Postoje razni obrasci bazirani na konceptu životnog ciklusa razvoja softvera. Ovdje je riječ o „*SDLC modelima*“. Navedeni modeli predstavljaju predložak za konkretno definiranje strukture procesa razvoja softvera odnosno definiciju odgovarajućih faza razvoja te njihovih međusobnih odnosa. Model vodopada, agilni model i spiralni model predstavljaju neke od primjera „*SDLC modela*“.

Tradicionalni oblici „*SDLC-a*“ su u pravilu podrazumijevali izvođenje odgovarajućih sigurnosnih provjera isključivo u sklopu faze testiranja i to pri kraju pojedine instance razvoja softvera. Važno je naglasiti da otkrivanje sigurnosnih propusta u kasnijim fazama razvoja softvera rezultira s većim troškovima njihove sanacije. U svrhu adresiranja ovog problema preporučuje se ugradnja odgovarajućih sigurnosnih procedura u sklopu svake pojedine faze „*SDLC-a*“. Ovdje je dakle riječ o definiranju sigurnog životnog ciklusa razvoja softvera („*SSDLC*“).

Općenite faze „*SDLC-a*“ su demonstrirane u sklopu priloženog dijagrama. Pritom su, kao preporučeni dio pojedinih faza razvoja softvera, označene relevantne sigurnosne procedure. Primjena sigurnosnih procedura u sklopu svih faza „*SDLC-a*“ podrazumijeva oblikovanje sigurnog životnog ciklusa razvoja softvera („*SSDLC*“).

[3] Temeljeno na sljedećim online izvorima: https://aws.amazon.com/what-is/sdlc/, https://www.kellton.com/kellton-tech-blog/what-is-software-development-life-cycle, https://snyk.io/learn/secure-sdlc/



Slika 2.1. Model SSDLC procesa. Dijagram je temeljen na sintezi informacija iz različitih izvora, pri čemu su se primarno koristile informacije sadržane u sklopu sljedećeg online izvora: https://owasp.org/.

U nastavku su izdvojene pojedine komponente vezane uz općeniti model „*SSDLC-a*“. Fokus je na opisu ključnih sigurnosnih procedura koje je preporučljivo ugraditi u sklopu navedenih faza životnog ciklusa razvoja softvera.

2.1 Dizajn

Faza dizajna, u kontekstu razvoja web aplikacije, podrazumijeva specifikaciju softverskih zahtjeva a na osnovu prethodno definiranih korisničkih zahtjeva. Ovdje je riječ o specifikaciji odnosno prijevodu zahtjeva definiranih iz perspektive korisnika u konkretne tehničke zahtjeve sustava. Tijekom ove faze se definiraju tehnički detalji proizvoda uz odgovarajuće konzultacije s klijentom. Dizajner prilikom konzultacija s klijentom pokušava što bolje razumjeti poslovni problem odnosno potrebe klijenta. Klijent je obavješten o tehničkim ograničenjima i okvirnim troškovima razvoja softvera te se u sklopu navedenih konzultacija nastoji oblikovati optimalno rješenje za klijenta. Proces dizajna rezultira s izradom modela konkretnog softverskog rješenja. To u pravilu podrazumijeva definiranje arhitekture sustava, eventualne detalje vezane uz pojedine procese te specifikaciju softverskih zahtjeva. Razina detalja, definiranih u sklopu izrađenog modela sustava odnosno plana izrade, ovisi o specifičnostima svakog pojedinog projekta.

Pronalaženje sigurnosnih propusta u ovoj fazi razvoja softvera rezultira s relativno malim troškovima vezanim uz njihovu sanaciju[4]. Kvalitetno definiranje sigurnosnih zahtjeva također u značajnoj mjeri doprinosi smanjenju sigurnosnih rizika. U svrhu ostvarenja navedenih ciljeva, preporučuje se primjena principa sigurnog dizajna te izvođenje procesa modeliranja prijetnji.

2.1.1 Principi sigurnog dizajna

U svrhu kvalitetnog oblikovanja sigurnosnih softverskih zahtjeva web aplikacije, preporučuje se primjena odgovarajućih principa sigurnog dizajna. Poštivanje navedenih principa omogućuje minimiziranje ranjivosti web aplikacije na razini dizajna odnosno minimiziranje

[4] Ovaj zaključak je baziran na rezultatima istraživanja „IBM System Science instituta“. Pokazuje se da sanacija pogrešaka, otkrivenih tijekom faze uvođenja u rad, rezultira sa 100 puta većim troškovima u odnosu na troškove njihove sanacije tijekom faze dizajna. Detalji vezani uz navedeno istraživanje se mogu preuzeti korištenjem sljedećeg linka: https://www.researchgate.net/figure/IBM-System-Science-Institute-Relative-Cost-of-Fixing-Defects_fig1_255965523

eventualnih negativnih efekata, koji mogu nastati uslijed eksploatacije pogrešaka vezanih uz dizajn sustava.

2.1.1.1 Princip najmanjih prava

Princip najmanjih prava („*Principle of Least Privilege*“) podrazumijeva dodjelu minimalnih ovlasti pojedinim entitetima sustava. Korisnicima sustava se dodjeljuju minimalne ovlasti potrebne za zadovoljenje prethodno definiranih korisničkih zahtjeva. Odgovarajuća granulacija funkcionalnosti sustava predstavlja preduvjet za kvalitetnu primjenu navedenog načela. Ograničavanjem pristupa podacima i funkcionalnostima web aplikacije minimizira se šteta od potencijalnog napada.

2.1.1.2 Razdvajanje zaduženja

Osnovna ideja principa razdvajanja zaduženja se temelji na nužnosti izbjegavanja dodjele prevelikih zaduženja, a time posljedično i prevelikih ovlasti pojedinom korisniku[5]. Dakle ovdje nije riječ o direktnom minimiziranju ovlasti u skladu s potrebama pojedinog korisnika već o oblikovanju pojedinih uloga na način da niti jedna ne zahtjeva korištenje previsoke razine ovlasti. Dodjela visoke razine ovlaštenja pojedinom korisniku podrazumijeva visoku razinu ovisnosti o pojedinom entitetu te posljedično dovodi do više razine ranjivosti sustava. Prevelika količina zaduženja dodijeljena pojedinom korisniku također može rezultirati sa relativno niskom kvalitetom izvršenja navedenih poslova. Kao rješenje navedenog problema preporučuje se preraspodjela zaduženja na više različitih uloga odnosno korisnika.

[5] Kao primjer prevelike dodjele zaduženja se može navesti scenarij vezan uz prodavača zaposlenog u sklopu određene online trgovine. Prodavač je zadužen za prodaju proizvoda te istovremeno i za oblikovanje cjenovne politike odnosno popusta vezanih uz pojedine proizvode. Navedena dodjela ovlasti rezultira s relativno visokim rizikom iz perspektive vlasnika online trgovine.

2.1.1.3 Princip obrane u dubinu

Ovaj princip podrazumijeva ugradnju više slojeva obrane vezane uz pojedini štićeni resurs. U slučaju ako napadač uspije zaobići ili probiti određeni sloj obrane, redundantni sigurnosni mehanizmi mogu spriječiti uspješnu realizaciju napada. Prilikom primjene principa obrane u dubinu potrebno je obratiti pozornost na očuvanje odgovarajućeg balansa između sigurnosti sustava i kompleksnosti njegovog korištenja iz perspektive krajnjih korisnika.

2.1.1.4 Sigurno ispadanje

Preporučuje se oblikovanje sigurnosnih mehanizama na način da eventualna pojava pogreške ne rezultira s kompromitiranjem sigurnog stanja sustava. U slučaju registriranja pogreške, koja može utjecati na funkcioniranje sigurnosnog mehanizma, preporučuje se automatsko blokiranje pristupa štićenim resursima. Dakle pristup štićenom dijelu sustava može biti omogućen samo u slučaju ako sigurnosni mehanizam funkcionira na predviđen način odnosno u slučaju ako nije detektiran određeni oblik relevantne pogreške.

2.1.1.5 Princip otvorenog dizajna

Sigurnost sustava se ne smije bazirati na tajnosti detalja vezanih uz njegovu implementaciju. Ranjivosti sustava je moguće otkriti neovisno o naporima uloženima u skrivanje njegove implementacije. Oslanjanje na tajnost implementacije može rezultirati sa značajnim problemima u slučaju ako potencijalni napadači eventualno ostvare pristup navedenim informacijama (primjerice izvornom kodu web aplikacije).

2.1.1.6 Ne vjerujte vanjskim uslugama

Vanjske usluge mogu biti eksploatirane od strane napadača. Podatke, zaprimljene na osnovu komunikacije s vanjskim uslugama, je potrebno provjeriti na isti način kao i podatke zaprimljene od strane krajnjih korisnika sustava.

2.1.1.7 Minimiziranje prostora za napad

Primjena ovog principa podrazumijeva izbjegavanje ugradnje nepotrebnih funkcionalnosti. Svaka dodatna funkcionalnost rezultira s potencijalnim uvođenjem novih ranjivost u sustav. Prilikom oblikovanja sustava potrebno je procijeniti je li određena funkcionalnost zaista potrebna odnosno opravdava li njezina implementacija dodatne troškove vezane uz osiguravanje web aplikacije.

2.1.2 Modeliranje prijetnji

Modeliranje prijetnji predstavlja strukturirani proces analize sigurnosnog aspekta dizajna sustava. Ova metodologija podrazumijeva registriranje imovine te osnovnih i izvedenih sigurnosnih zahtjeva vezanih uz pojedine štićene resurse. Također se provodi identifikacija potencijalnih prijetnji te povezanih ranjivosti analiziranog softvera.

Izvođenje navedenog procesa podrazumijeva generiranje sljedećih artefakata:

- Apstraktni prikaz analiziranog sustava (arhitektura sustava)
- Lista relevantnih ranjivosti
- Lista relevantnih prijetnji koja također sadrži ocjenu prioriteta vezanih uz pojedine prijetnje
- Lista preporučenih kontrola

Na osnovu rezultata procesa modeliranja prijetnji donose se odluke vezane uz prilagodbu dizajna web aplikacije odnosno ugradnju odgovarajućih sigurnosnih mehanizama.

Konkretni primjer izvođenja procesa modeliranja prijetnji je detaljno dokumentiran u sklopu poglavlja „Modeliranje prijetnji na konkretnom primjeru“.

2.2 Implementacija i testiranje

Faza implementacije podrazumijeva konkretno programiranje programskog proizvoda odnosno definiciju odgovarajućeg programskog koda.

Testiranje se u sklopu „*SDLC modela vodopada*“ definira kao zasebna faza koja se izvodi nakon faze implementacije softvera. U praksi se navedene faze često izvode paralelno. Na ovaj način se omogućuje ranije otkrivanje eventualnih pogrešaka a time i minimiziranje troškova vezanih uz njihovu sanaciju. Testiranje se provodi u svrhu provjere usklađenosti implementirane web aplikacije sa specifikacijom softverskih zahtjeva.

U nastavku su opisani ključni sigurnosni koncepti i procedure, koje je preporučljivo primijeniti tijekom faza implementacije i testiranja softvera.

2.2.1 Smjernice za pisanje sigurnog koda i sigurnosne metrike

Pojedine programerske tvrtke često dokumentiraju vlastite standarde odnosno smjernice za pisanje sigurnog koda i to u ovisnosti o specifičnostima njihovog proizvodnog procesa. Navedeni dokumenti se ipak u pravilu baziraju na određenom skupu univerzalnih, tehnološki agnostičkih smjernica. Ovdje je prvenstveno riječ o „*OWASP smjernicama za sigurno kodiranje*[6]“ i „*SEI CERT standardima za kodiranje*[7]“.

[6] Navedene smjernice je moguće preuzeti korištenjem sljedećeg linka: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

[7] Navedene smjernice je moguće preuzeti korištenjem sljedećeg linka: https://wiki.sei.cmu.edu/confluence/display/seccode

Primjena standarda i smjernica za pisanje sigurnog koda smanjuje vjerojatnost pojave pogreške odnosno ugradnje ranjivosti tijekom faze implementacije softverskog proizvoda. Preporučljivo je definirati smjernice na koncizan i razumljiv način odnosno učiniti ih primjenjivim od strane razvojnih programera koji nisu nužno specijalizirani u području informacijske sigurnosti.

U svrhu sigurnog razvoja web aplikacije odnosno softverskog proizvoda, također se preporučuje definicija sigurnosnih metrika. Navedene metrike se koriste prilikom izvođenja testova odnosno verifikacija vezanih uz procjenu zadovoljenja sigurnosnih softverskih zahtjeva. Ovdje je primjerice riječ o vremenu potrebnom za razrješenje ranjivosti i frekvenciji generiranja pogrešaka.

2.2.2 Sigurnosni pregled izvornog koda („*secure code review*“)

Sigurnosni pregled izvornog koda podrazumijeva analizu izvornog koda softverskog proizvoda s ciljem direktne identifikacije te eliminacije odgovarajućih ranjivosti. Izvođenjem navedenog procesa se provjerava prisutnost relevantnih sigurnosnih mehanizama te kvaliteta njihove implementacije u sklopu analiziranog sustava.

Proces pregleda izvornog koda u pravilu podrazumijeva primjenu automatiziranih alata u kombinaciji s manualnom provjerom prioritetnih područja u sklopu izvornog koda.

„*Automatizirani secure code review*“ podrazumijeva korištenje automatiziranih alata s ciljem ostvarenja brze i jeftine detekcije osnovnih oblika poznatih ranjivosti. Ovdje je prvenstveno riječ o „*SAST*[8]“ i „*SCA alatima*[9]“. Navedeni pristup sigurnosnom pregledu izvornog koda omogućuje detekciju potencijalnih problema u svakom pojedinom trenutku tijekom implementacije analiziranog proizvoda. Nedostatak ove metode jest veliki broj lažno

[8] „*SAST (Static Application Security Testing) alati*“ omogućuju automatizirano izvršavanje statičke analize izvornog koda analizirane aplikacije.

[9] „*SCA (Software Composition Analysis) alati*“ omogućuju automatsku detekciju korištenih paketa otvorenog koda te eventualnih, prethodno indeksiranih, ranjivosti vezanih uz korištene verzije navedenih komponenti. „SCA alati“ dakle upućuju razvojnog programera na korištenje sigurne verzije odnosno ažuriranje odgovarajuće komponente sustava. Temeljeno na informacijama preuzetima sa sljedećeg web mjesta: https://github.blog/2022-09-09-sca-vs-sast-what-are-they-and-which-one-is-right-for-you/

pozitivnih i lažno negativnih rezultata, nepotpuno razumijevanje konteksta aplikacije i vezanost uz određeni programski jezik.

Manualno izvođenje procesa sigurnog pregleda koda podrazumijeva detaljnu analizu koda od strane senior developera te samim time predstavlja relativno značajan trošak. Navedeni pristup omogućuje otkrivanje dodatnih oblika ranjivosti koje nije moguće detektirati isključivim korištenjem automatiziranih alata.

Predlaže se kombiniranje dvaju navedenih pristupa (manualni i automatizirani pregled koda) s ciljem pravovremene detekcije svih jednostavnijih oblika ranjivosti te također detekciju specifičnih oblika ranjivosti, koje je teško detektirati samo na osnovu primjene automatiziranih alata.

Proces sigurnog pregleda izvornog koda je korisno usporediti s procesom penetracijskog testiranja. Na ovaj način je moguće ostvariti višu razinu razumijevanja obaju navedenih pojmova[10].

Proces penetracijskog testiranje podrazumijeva oblik sigurnosne procjene koji problemu analize sigurnosti pristupa iz perspektive potencijalnog napadača. Penetracijsko testiranje se u užem smislu riječi („*black box penetracijsko testiranje*“) bazira na primjeni dinamičkog „*black box pristupa*[11]“ dok „*secure code review*“ podrazumijeva primjenu statičkog „*white box pristupa*[12]“. „*Secure code review pristup*“ je fokusiran na direktnu verifikaciju implementiranih sigurnosnih mehanizama dok je penetracijski test fokusiran na pronalaženje načina za zaobilazak sigurnosnih mehanizama[13].

Korištenje različitih perspektiva, odnosno primjena različitih oblika sigurnosnog testiranja, može rezultirati s pronalaskom različitih vrsta pogrešaka.

---

[10] Navedena usporedba je prvenstveno bazirana na literaturi dostupnoj preko sljedeće poveznice: https://owasp.org/www-project-web-security-testing-guide/

[11] *„Black box pristup“* podrazumijeva izvršenje analize bez pristupa detaljima implementacije odnosno izvornom kodu analizirane aplikacije.

[12] *„White box pristup“* podrazumijeva izvršenje analize uz pristup svim detaljima vezanim uz implementaciju analiziranog sustava.

[13] *„White box penetracijsko testiranje“* podrazumijeva kombiniranje procesa sigurnog pregleda izvornog koda i penetracijskog testiranja u užem smislu riječi. Tijekom izvođenja navedenog procesa se u pravilu izvršava **djelomična** analiza implementacijskih detalja analiziranog sustava.

Prednost penetracijskog testiranja u odnosu na proces sigurnog pregleda izvornog koda podrazumijeva niže troškove, brže izvođenje, nižu zahtijevanu razinu vještina izvođača odnosno primjenu univerzalnih znanja koja nisu usko vezana uz specifičnu korištenu tehnologiju.

Važno je naglasiti da je određene oblike problema prikladnije testirati analizom aplikacije prilikom njezinog izvođenja. Ovdje je prvenstveno riječ o detekciji pogrešaka koje mogu proizaći kao rezultat kompleksne interakcije različitih komponenti sustava (uključujući komponente trećih strana). Penetracijsko testiranje također omogućuje efikasno detektiranje ranjivosti u konfiguraciji konkretne instance finalizirane aplikacije. Kao primjer se može navesti korištenje zadanih vjerodajnica, slabih lozinki ili pogrešnog tipa okruženja (primjena razvojnog okruženja u produkciji). Prilikom izvršenja navedene analize, jednostavnije oblike ranjivosti je moguće testirati isključivim korištenjem „*DAST alata*“.

Nedostatak penetracijskog testiranja je to što, iako navedeni proces u pravilu rezultira s listom uspješno izvedenih eksploatacija, ne rezultira nužno s uputstvima vezanim uz sanaciju detektiranih ranjivosti. Proces sigurnog pregleda izvornog koda s druge strane predstavlja bolju priliku za unaprjeđenje znanja developera vezanih uz sigurnu implementaciju softverskog proizvoda. S obzirom da se navedeni proces temelji na detaljnom razumijevanju implementacije analiziranog sustava, izvršitelj je u mogućnosti definirati detaljna uputstva odnosno preporuke vezane uz sanaciju detektiranih ranjivosti. „*Secure code review pristup*“ također omogućuje preciznu analizu cjelokupnog sustava dok penetracijsko testiranje u pravilu omogućuje detektiranje relativno malog podskupa ranjivosti analiziranog sustava.

Za kraj je potrebno naglasiti da se u praksi preporučuje primjena obaju pristupa sigurnosnom testiranju web aplikacija. U optimalnom slučaju navedeni bi se procesi trebali provoditi od strane različitih izvršitelja.

2.3 Održavanje

Glavni fokus navedene faze životnog ciklusa razvoja softvera podrazumijeva unaprjeđenje aplikacije u skladu s novootkrivenim ili prethodno neadresiranim korisničkim zahtjevima i ranjivostima te općenitim pogreškama u radu sustava. Pri tome se može pristupiti poboljšanju

postojećih ili razvoju novih funkcionalnosti. Navedena faza slijedi nakon uvođenja u rad predmetne aplikacije[14].

Razvoj novih inačica softvera u načelu podrazumijeva ponovno izvođenje svih prethodno navedenih faza životnog ciklusa razvoja softverskog proizvoda. Svaki od navedenih dijelova glavnog ciklusa podrazumijeva provođenje analize dodatnih korisničkih zahtjeva, redizajn pojedinih dijelova odnosno dizajn unaprijeđene verzije aplikacije, implementaciju, testiranje, uvođenje u rad te ponovnu retrospekciju.

U kontekstu zadovoljavanja sigurnosnih zahtjeva sustava ova faza prvenstveno podrazumijeva primjenu i unaprjeđenje prethodno definiranog plana odgovora na računalno sigurnosne incidente, razvoj i primjenu sigurnosnih zakrpi te provođenje dodatnog sigurnosnog testiranja sustava. Dodatno sigurnosno testiranje sustava se pritom bazira na primjeni procesa „*red teaming-a*“ i penetracijskog testiranja.

2.3.1 Odgovor na računalne sigurnosne incidente

Odgovor na računalne sigurnosne incidente podrazumijeva proces detekcije i reakcije vezane uz napad koji ugrožava štićenu informacijsku imovinu. Postoje različiti pristupi izvođenju navedenog procesa. „*Front-loaded prevention pristup*“ podrazumijeva detekciju napada i prevenciju uspješne realizacije napada tijekom njegova izvođenja. „*Back-loaded recovery pristup*“ podrazumijeva detekciju i minimiziranje štetnih posljedica realiziranog napada te blokiranje nastavka njegovog izvršenja. „*Hybrid incident response pristup*“ podrazumijeva kombiniranu primjenu dvaju prethodno opisanih pristupa. Odgovor na računalne sigurnosne incidente se bazira na pravovremenom generiranju odgovarajućih upozorenja vezanih uz izvođenje napada i to na temelju indirektnih indikatora kompromitacije i/ili indikatora realizacije napada. Efikasno upravljanje računalnim sigurnosnim incidentima omogućuje blokiranje izvođenja napada prije samog nastanka štete i/ili minimiziranje direktne štete

[14] Bazirano na informacijama sa sljedećeg izvora: https://www.techwalla.com/articles/the-maintenance-phase-in-the-software-life-cycle

vezane uz eventualno kompromitirane podatke i funkcionalnosti, umanjenje troškova i vremena oporavka te kontrolu štete vezanu uz reputaciju organizacije[15].

U svrhu kvalitetnijeg izvođenja odgovora na računalne sigurnosne incidente preporučuje se prethodno definiranje odgovarajućeg plana. U sklopu navedenog plana se detaljno opisuju pojedine faze izvođenja procesa te se dodjeljuju odgovornosti vezane uz izvođenje i upravljanje navedenim procesom.

Proces upravljanja incidentima započinje s detekcijom izvođenja konkretnog napada. U svrhu ostvarenja brze odnosno rane detekcije napada, preporučuje se primjena „*SIEM sustava*[16]“. Rezultati prethodnog koraka se zatim analiziraju te se odbacuju eventualna lažno pozitivna upozorenja. Slijedi dodjela prioriteta pojedinim verificiranim incidentima te eventualno izvještavanje odgovarajućih dionika o nastaloj situaciji. Ključna faza procesa upravljanja incidentima podrazumijeva poduzimanje odgovarajućih koraka s ciljem obuzdavanja incidenta te prikupljanje forenzičkih dokaza vezanih uz detektirani napad. U konačnici slijedi faza oporavka napadnutog sustava te dokumentiranje uputa odnosno prijedloga vezanih uz razrješavanje detektiranih ranjivosti te unaprjeđenje samog procesa upravljanja incidentima a na osnovu prikupljenog iskustva.

U svrhu provjere i unaprjeđenja sposobnosti tima zaduženog za upravljanje računalnim sigurnosnim incidentima („*CSIRT tim*“), preporučuje se izvođenje „*red teaming procesa*“. Ovdje je riječ o vježbi koja podrazumijeva izvršenje kontroliranog napada na osnovu kojeg se izvršava testiranje procesa odgovora na računalne sigurnosne incidente te performansi sigurnosnog tima.

[15] Bazirano na informacijama sa sljedećeg izvora: https://www.ibm.com/topics/incident-response

[16] *„SIEM (Security information and event management) sustav“* omogućuje agregiranje sigurnosno relevantnih podataka iz različitih izvora te analizu navedenih podataka i njihove korelacije u svrhu pronalaska obrazaca vezanih uz potencijalno izvršenje napada. Prikupljeni podaci se pritom mogu odnositi na rezultate obrade vatrozida, antivirusnog programa, *„IPS/IDS sustava“*, *„proxy poslužitelja“* te operativnih i sistemskih zapisa. Sigurnosni timovi koriste rezultate *„SIEM sustava“* u svrhu brže i jednostavnije detekcije napada u tijeku. Moderni oblici *„SIEM sustava“* podrazumijevaju primjenu umjetne inteligencije u svrhu automatizacije detekcije napada te generiranja automatskog odgovora vezanog uz navedene incidente. Bazirano prvenstveno na informacijama iz sljedećeg izvora: https://netacea.com/glossary/security-information-and-event-management-siem/

2.3.2 Metodologije penetracijskog testiranja

Prilikom primjene procesa penetracijskog testiranja, preporučljivo je koristiti neku od navedenih metodologija:

- *Penetration Testing Execution Standard*
- *PCI Penetration Testing Guide*
- *Penetration Testing Framework*
- *Technical Guide to Information Security Testing and Assessment*
- *Open Source Security Testing Methodology Manual*
- *OWASP Web Security Testing Guide*

Pritom je također moguće kombinirati navedene metodologije u ovisnosti o specifičnim potrebama penetracijskog testera. „*OWASP Web Security Testing Guide*“ predstavlja izuzetno kvalitetan materijal koji je preporučljivo koristiti prilikom izvođenja penetracijskog testiranja. Navedeni vodič podrazumijeva primjenu „*black box pristupa penetracijskog testiranja*“ te sadrži detaljnu dokumentaciju širokog raspona tehnika za izvođenje kontroliranog napada na web aplikacije[17].

U nastavku Specijalističkog rada su sažeto opisane pojedine faze „*PTES metodologije*[18]“. Navedeni standard sadrži upute vezane uz izvođenje penetracijskog testiranja u širem smislu riječi. „*PTES*“ osim smjernica za izvođenje penetracijskog testiranja web aplikacija također uključuje upute vezane uz testiranje sigurnosti računalnih mreža, fizičke sigurnosti te izvođenje metoda društvenog inženjeringa. Standard se temelji na definiciji smjernica vezanih uz izvođenje 7 različitih faza testiranja:

1. Faza pripremnih radnji podrazumijeva usuglašivanje sadržaja penetracijskog testiranja s korisnikom. To prvenstveno podrazumijeva definiciju opsega analize („*scope*“),

[17] Vodič je moguće preuzeti korištenjem sljedećeg linka: https://owasp.org/www-project-web-security-testing-guide/

[18] Bazirano na literaturi pohranjenoj u sklopu sljedećeg web mjesta: http://www.pentest-standard.org/index.php/Main_Page

ciljeva, pravila i rokova izvođenja penetracijskog testa te odgovornosti i ovlasti izvršitelja penetracijskog testiranja.

2. Faza prikupljanja informacija podrazumijeva prikupljanje i bilježenje svih relevantnih informacija vezanih uz analizirani sustav, korištenjem svih raspoloživih izvora. Pri tome se koriste odgovarajući alati i tehnike (primjerice „*Maltego alat*“ i „*Google Hacking tehnika*“). Konkretna implementacija ove faze ovisi o količini potencijalno osjetljivih informacija koje je korisnik spreman otkriti izvršitelju penetracijskog testiranja.
3. Faza modeliranja prijetnji podrazumijeva korištenje prethodno definiranog modela prijetnji ili razvoj novog modela u svrhu okvirnog detektiranja i rangiranja ranjivosti koje je moguće eksploatirati tijekom izvođenja konkretnog napada.
4. Faza analize ranjivosti podrazumijeva validaciju pojedinih elemenata iz prethodno definiranog skupa potencijalnih ranjivosti.
5. Faza eksploatacije podrazumijeva konkretno izvršenje napada. Prilikom izvršenja napada se koriste prethodno verificirane ranjivosti web aplikacije. Smjernice vezane uz ovu fazu također upućuju na odgovarajuće oblikovanje napada s ciljem izbjegavanja detekcije. Na ovaj način se dodatno testira kvaliteta procesa odgovora na računalno sigurnosne incidente.
6. Faza nakon eksploatacije podrazumijeva preuzimanje odgovarajućih podataka, eskalaciju napada, instalaciju „*backdoora*“ te brisanje tragova vezanih uz izvršenje svih navedenih akcija.
7. Faza izvještavanja podrazumijeva dokumentiranje svih prethodno izvršenih aktivnosti. Dokumentacija treba biti oblikovana na način da bude razumljiva korisniku usluge penetracijskog testiranja. U sklopu izvješća se prvenstveno prezentiraju korištene metode odnosno izvršeni testovi, rezultati vezani uz uspješne proboje te prijedlozi vezani uz unaprjeđenje sigurnosti analiziranog sustava.

Proces penetracijskog testiranja je dodatno dokumentiran u sklopu poglavlja „Alati za penetracijsko testiranje web aplikacija“ i „Penetracijsko testiranje web aplikacije“.

3 IZDVOJENE METODE NAPADA NA WEB APLIKACIJE

U nastavku Specijalističkog rada su opisani ključni koncepti vezani uz izdvojen skup metoda napada na web aplikacije[19][20]. Primjena navedenih metoda, u kontekstu penetracijskog testiranja, će biti demonstrirana u sklopu poglavlja „Penetracijsko testiranje web aplikacije“.

3.1 Napadi na strani poslužitelja

3.1.1 SQl injection

„*SQL injection*“ predstavlja jednu od najpoznatijih metoda napada na web aplikacije. Navedenu metodu je relativno jednostavno razumjeti i iskoristiti u kontekstu izvršenja konkretnog napada. Unatoč relativno jednostavnoj implementaciji relevantnih sigurnosnih mehanizama, još uvijek postoji široka rasprostranjenost ranjivosti koje omogućuju uspješno ostvarenje „*SQL injection napada*“.

Preduvjet uspješnog izvršenja napada podrazumijeva specifičnu funkcionalnost web aplikacije. Ovdje je riječ o upotrebi korisničkog unosa prilikom oblikovanja „*SQL upita*“ prema „*SQL bazi podataka*“ i to bez primjene odgovarajućih sigurnosnih mehanizama. Osnovni oblik „*SQL injection napada*“ podrazumijeva mogućnost direktnog uvida u rezultat izvršenja transakcije oblikovane na osnovu korisničkog unosa. Rezultat se prikazuje u sklopu jedne ili više izlaznih točaka web aplikacije. Napad se temelji na oblikovanju odgovarajućeg oblika korisničkog unosa odnosno segmenta „*SQL upita*“. Navedeni korisnički unos se posredstvom aplikacijske logike ugrađuje u strukturu legitimnog „*SQL upita*“, što može rezultirati izlaskom izvan njegovog konteksta odnosno s izmjenom njegove strukture. Izlaz izvan konteksta originalnog „*SQl upita*“ omogućuje modifikaciju originalnog odnosno

[19] Stuattard D., Pinto M. (2011). The Web Application Hacker's Handbook - Finding and Exploiting Security Flaws. Izdavač: Wiley Publishing, Inc. (2011).

[20] Temeljeno na sadržaju u sklopu sljedećeg web mjesta: https://portswigger.net/web-security

definiranje dodatnih upita prema bazi podataka. To u konačnici otvara mogućnost za preuzimanje i/ili izmjenu potencijalno osjetljivih podataka. Rezultati transakcija temeljenih na korisničkom unosu se prikazuju u sklopu odgovarajućih izlaznih točaka web aplikacije. Ranjivost koja omogućuje izvođenje ovog oblika napada je relativno jednostavno detektirati korištenjem automatiziranih alata za izvođenje procesa penetracijskog testiranja.

„*Laravel okvir*“ sadrži niz metoda koje je moguće koristiti prilikom sigurnog oblikovanja upita prema bazi podataka, baziranih na korisničkom unosu. Ovdje je prvenstveno riječ o „*Eloquent ORM sustavu*[21]“. Bez obzira na navedeno, neodgovarajuća upotreba navedenog okvira može rezultirati s pojavom ranjivosti. Kao primjer se može navesti pokušaj brze integracije obrasca „*SQL upita*“ unutar „*Laravel okvira*“. „*Sirovi SQL upit*“ je najlakše integrirati korištenjem „*select metode*“, definirane nad „*DB fasadom*“. U slučaju ako korištenje navedenog obrasca podrazumijeva dodatnu ugradnju korisnički definiranih podataka, navedena integracija može rezultirati s ugradnjom „*SQL injection ranjivosti*“. Kao dodatni primjer se može navesti nesigurno korištenje „*raw metoda Laravel okvira*“. Navedene metode se prvenstveno koriste prilikom izvođenja složenih „*SQl upita*“ odnosno upita koje je teško definirati korištenjem lanca „*Eloquent metoda*“.

U nastavku je izložen jednostavan primjer nesigurnog korištenja „*select metode*“ u sklopu „*Laravel okvira*“. „*Varijabla $id*“ predstavlja numerički identifikator referencirane objave, dostavljen u sklopu odgovarajućeg korisničkog zahtjeva:

```
DB::select('SELECT title, description, image FROM posts WHERE id ='
. $id);
```

Slika 3.1. Primjer nesigurnog korištenja DB::select naredbe

[21] „*ORM sustav*“ omogućuje povezivanje relacijske baze podataka s odgovarajućim objektima unutar objektno orijentiranog jezika. Pojedine klase odnosno „modeli“ sadržavaju metode koje je moguće koristiti u svrhu jednostavnijeg generiranja upita prema bazi podataka. „ORM sustav“ dakle predstavlja dodatni sloj apstrakcije u kontekstu definicije komunikacije prema bazi podataka te olakšava održavanje izvornog koda aplikacije. U slučaju korištenja odgovarajućeg „ORM sustava“, eventualna promjena korištene baze podataka u pravilu ne zahtjeva značajnu modifikaciju izvornog koda aplikacije.

Prilikom oblikovanja koda za generiranje upita, razvojni programer je propustio izvršiti parametrizaciju „*SQl upita*“[22]. Navedeni propust otvara razne mogućnosti za uspješno izvršenje „*SQL napada*“.

„*SQL injection UNION napad*“ predstavlja naročito opasan oblik „*SQl injection napada*“. Ovaj napad podrazumijeva injekciju dodatne „*SQL select naredbe*“ unutar originalnog „*SQL upita*“ (baziranog na korištenju „*select naredbe*“). U svrhu povezivanja dvaju „*SQL naredbi*“ također se injektira i „*UNION operator*“. Uspješna realizacija navedenog napada može rezultirati preuzimanjem određenog skupa podataka sadržanih u sklopu proizvoljne referencirane tablice. Prilikom izvođenja navedene vrste napada potrebno je obratiti pozornost na odgovarajuća ograničenja. Injektirana „*select naredba*“ mora referencirati odnosno vraćati jednaki broj stupaca kao i originalni „*SQL upit*“. Tipovi podataka vezani uz pojedine referencirane stupce tablice također moraju biti jednaki iz perspektive obaju dijelova integriranog „*SQL upita*“.

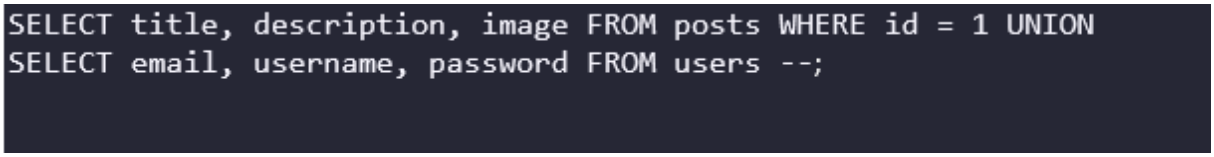
U nastavku je demonstriran primjer korisničkog unosa (vrijednost „*$id varijable*“) koji rezultira s eksploatacijom prethodno izloženog primjera nesigurnog generiranja „*SQl upita*“.

```
1 UNION SELECT email, username, password FROM users -
```

Slika 3.2. Primjer zlonamjernog korisničkog unosa

Korištenje navedenog malicioznog unosa može rezultirati s definicijom sljedećeg „*SQL upita*“:

[22] Parametrizacija *„SQL upita“* podrazumijeva odvajanje strukture pripremljenog *„SQL upita“* od korisničkog unosa. Korisnički unos se ugrađuje tek nakon jasne definicije strukture relevantnog upita. Na ovaj način se izbjegava mogućnost modifikacije strukture *„SQL upita“* od strane maliciozno oblikovanog korisničkog upita.



Slika 3.3. Rezultat zlonamjernog korisničkog unosa

Navedeni upit omogućuje preuzimanje svih redaka „*users tablice*“ odnosno preuzimanje adrese e-pošte, korisničkog imena i lozinke vezanih uz sve registrirane korisnike.

Korištenje „- - *izraza*“ je redundantno u kontekstu eksploatacije konkretnog primjera ranjivosti. Navedeni izraz naime predstavlja oznaku za početak komentara u kontekstu „*SQL jezika*“ te omogućuje ignoriranje eventualnog dijela originalnog „*SQL upita*“, generiranog nakon injektiranog sadržaja. Bez korištenja navedenog izraza, konačni „*SQl upit*“ bi u većini slučajeva rezultirao s pogreškom.

Iz perspektive izvršenja procesa manualnog penetracijskog testiranja naročito je zanimljiva „*Blind SQL injection metoda*“, koja predstavlja specifični oblik „*SQL injection napada*“. Ova metoda omogućuje otkrivanje ranjivosti koje je relativno teško ili nemoguće otkriti korištenjem automatiziranih alata za penetracijsko testiranje. Metodu je prikladno koristiti u situacijama kada web aplikacija ne vraća eksplicitni rezultat transakcije temeljene na korisničkom unosu ali ga svejedno koristi (bez primjene odgovarajućih sigurnosnih mehanizama) u kontekstu izvršenja određenih transakcija u sklopu baze podataka. Ovdje je dakle riječ o situaciji gdje web aplikacija sadrži ranjivost koja omogućuje izvršenje „*SQL injection napada*“, međutim dokaze o prisutnosti navedene ranjivosti odnosno rezultate izvršenja navedenog napada je potrebno prikupiti na indirektan način. To je naravno pod pretpostavkom da web aplikacija emitira informacije koje ovise o rezultatu izvršenja transakcije u bazi podataka, oblikovane na osnovu odgovarajućeg korisničkog unosa.

U svrhu izbjegavanja pojave ranjivosti koje omogućuju realizaciju obaju izloženih oblika „*SQl injection prijetnji*“, prvenstveno se predlaže primjena mehanizma za parametrizaciju „*SQL upita*“. Vezano uz konkretan primjer upotrebe, baziran na korištenju „*Laravel okvira*“, to prvenstveno podrazumijeva preoblikovanje „*SQl upita*“ u oblik kompatibilan s „*Eloquent sustavom*“. Ako je riječ o kompleksnom „*SQl upitu*“, kojeg nije moguće ili nije jednostavno zapisati u obliku „*Eloquent naredbi*“, preporučuje se odgovarajuća parametrizacija relevantne

„*Laravel raw metode*“ ili oblikovanje odgovarajućeg upita u sklopu „*PDO metode*“. „*PDO metoda*“ implicira parametrizaciju varijabli injektiranih prilikom definicije strukture upita.[23]

„*Laravel query builder*“ se bazira na korištenju „*PDO mehanizma*“ te ujedno predstavlja osnovu implementacije „*Eloquent ORM sustava*“. Prema tome, prilikom korištenja „*Laravel query builder-a*“ odnosno „*Eloquent ORM sustava*“, nije potrebno izvršavati provjeru odnosno modifikaciju konkretnih parametara metode, definiranih na osnovu korisničkog unosa.

Uz navedeno se također preporučuje provedba odgovarajuće validacije i/ili filtriranja[24] korisničkog unosa. Ovdje je riječ o implementaciji principa „zaštite po dubini“ odnosno implementaciji potencijalno redundantnih sigurnosnih mehanizama s ciljem minimiziranja ranjivosti.

U nastavku je izdvojen skup različitih pristupa koje je moguće koristiti u svrhu sigurnog generiranja analiziranog primjera „*SQL upita*“:

1. Primjena „*Eloquent ORM sustava*“ („*Post klasa*“ predstavlja model tablice koju želimo referencirati u sklopu odgovarajućeg upita):

```
Post::where('id', $id)
      ->get();
```

Slika 3.4. Primjena "Eloquent ORM sustava"

[23] Važno je naglasiti da korišteni *„PDO sustav“* (konkretno *„prepare metoda“)* ne podržava dinamičko definiranje naziva tablica odnosno stupaca u sklopu definicije strukture odgovarajućeg *„SQL upita“*.

[24] Filtriranje korisničkih podataka ovdje podrazumijeva zamjenu pojedinih znakova ili provedbu odgovarajućeg *„enkodiranja“*

2. Primjena „*Laravel query builder sustava*“:

```
DB::table('posts')
          ->where('id', '=', $id)
          ->get();
```

Slika 3.5. Primjena "Laravel query builder sustava"

3. Primjena „*raw metode*“ uz definiciju parametra korištenjem pozicijskog markera:

```
DB::select(DB::raw('SELECT * FROM posts WHERE id =?'), array($id));
```

Slika 3.6. Primjena "raw metode" uz korištenje pozicijskog markeraXSS

4. Primjena „*raw metode*“ uz definiciju parametra korištenjem imenovanog markera:

```
DB::select(DB::raw('SELECT * FROM posts WHERE id =:id'), array(
   'id' => $id,));
```

Slika 3.7. Primjena "raw metode" uz korištenje imenovanog markera

5. Primjena „*PDO mehanizma*“ uz definiciju parametra korištenjem pozicijskog markera:

```
$stmt = $pdo->prepare("SELECT * FROM posts WHERE id=?");
$stmt->execute([$id]);
$post = $stmt->fetch();
```

Slika 3.8. Primjena "PDO mehanizma" uz korištenje pozicijskog markera

6. Primjena „*PDO mehanizma*“ uz definiciju parametra korištenjem imenovanog markera:

```
$stmt = $pdo->prepare("SELECT * FROM posts WHERE id=:id");
$stmt->execute(['id' => $id]);
$post = $stmt->fetch();
```

Slika 3.9. Primjena "PDO mehanizma" uz korištenje imenovanog markera

3.1.2 Napad na kontrolu pristupa

Kontrola pristupa predstavlja primjenu odgovarajućih ograničenja vezanih uz prava pristupa korisnika. U kontekstu web aplikacija to podrazumijeva uvođenje ograničenja vezanih uz pristup pojedinim resursima odnosno podacima te ograničenja vezanih uz mogućnost izvršenja pojedinih funkcionalnosti aplikacije. Ovdje je dakle riječ o horizontalnoj i vertikalnoj kontroli pristupa.

Analiza ranjivosti vezanih uz kontrolu pristupa se u pravilu bazira na manualnoj analizi izvornog koda te izvođenju procesa manualnog penetracijskog testiranja. Pravila pristupa se implementiraju u ovisnosti o specifičnim zahtjevima organizacije. Analiza ranjivosti kontrole pristupa, vezana uz svaku pojedinu web aplikaciju, podrazumijeva upotrebu specijaliziranih alata uz odgovarajuću konfiguraciju.

U nastavku je izdvojen skup potencijalnih ranjivosti kontrola pristupa web aplikacija. Također se prezentiraju pojedine metode eksploatacije navedenih ranjivosti.

3.1.2.1 Skrivanje funkcionalnosti web aplikacije

U sklopu određenih web aplikacija osjetljive funkcionalnosti i resursi se pokušavaju zaštiti prikrivanjem relevantnih poveznica na navedene sadržaje. Poveznice prema osjetljivim funkcionalnostima i resursima se generiraju isključivo u sklopu odgovora na upit

privilegiranog korisnika. Ovdje je riječ o nesigurnoj metodi zaštite pristupa prema pojedinim komponentama web aplikacije. Napadač može pogoditi sadržaj skrivene poveznice, izvođenjem napada grubom silom. Korištenje kompleksnih „*URL-ova*“ također ne mora nužno osigurati zadovoljavajuću razinu sigurnosti. U ovom slučaju se ranjivost bazira na raznim oblicima curenja podataka vezanih uz korištene „*URL-ove*“. Informacije o korištenim „*URL-ovima*“ je primjerice moguće pronaći analizom generiranog „*javascript koda*“ te „*robots.txt datoteke*“.

3.1.2.2 Kontrola pristupa bazirana na klijentu

Aplikacija može teoretski koristiti funkcionalnost klijenta odnosno web preglednika u svrhu pohrane odnosno praćenja prava pristupa, dodijeljenih pojedinim korisnicima. Ovo rješenje uvodi ranjivost u kontekstu sustava za kontrolu pristupa. Krajnji korisnik ima punu kontrolu nad klijentom te je u mogućnosti modificirati podatke koje klijent dostavlja web poslužitelju.

3.1.2.3 Neusklađenost konfiguracije pojedinih dijelova sustava

Ranjivosti vezane uz kontrolu pristupa mogu nastati kao rezultat neusklađenosti konfiguracije pojedinih dijelova sustava. Neusklađenosti se mogu pojaviti u odnosu između pojedinih komponenti sustava web aplikacije te u odnosu između sustava web aplikacije i njegovog okruženja.

Web aplikacija može koristiti kontrolu pristupa, implementiranu u sklopu neke od komponenti njezinog okruženja. U tu svrhu se najčešće koristi „*forward proxy*“ ili web poslužitelj. Neusklađenost konfiguracije same web aplikacije i sustava koji implementira kontrolu pristupa može rezultirati s propuštanjem neautoriziranog zahtjeva prema web aplikaciji. Kao primjer se može navesti scenarij u kojem web aplikacija, prilikom referenciranja odgovarajuće rute umjesto „*URL-a*“ koristi podatke sadržane u sklopu

određenih „*HTTP zaglavlja" korisničkog zahtjeva*[25]. Ovdje je primjerice riječ o „*Original-URL*" i „*X-Rewrite-URL zaglavljima*". Web poslužitelj s druge strane može bazirati kontrolu pristupa na provjeri „*URL-a*". U ovoj situaciji se, prilikom generiranja odgovora iz perspektive same web aplikacije, ignorira navedeni „*URL*". Napadač može zaobići implementiranu kontrolu pristupa generiranjem korisničkog zahtjeva koji, uz dozvoljeni „*URL*", sadrži i zabranjenu rutu, definiranu u sklopu nekog od relevantnih „*HTTP zaglavlja*".

Slična ranjivost može proizaći iz neusklađenosti konfiguracije u odnosu između pojedinih komponenti web aplikacije. Komponenta u sklopu koje je implementirana kontrola pristupa, koristi određene kriterije provjere zaprimljenog „*URL-a*". Navedeni kriteriji se mogu razlikovati od kriterija primijenjenih u sklopu sustava web aplikacije zaduženog za prosljeđivanje korisničkih zahtjeva („*router*"). Kao primjer se može navesti scenarij u kojem „*router*" procesuira sve varijacije zaprimljene rute na isti način, neovisno o tome jesu li pojedini znakovi rute definirani kao malo ili veliko slovo. U slučaju ako sustav za kontrolu pristupa registrira navedene razlike, propustit će sve oblike ruta koje nisu striktno zabranjene u sklopu njegove konfiguracije[26]. Navedene rute će u konačnici biti proslijeđene na daljnju obradu, od strane „*router-a web aplikacije*".

3.1.2.4 Sigurnosne preporuke

U nastavku je izložen skup sigurnosnih preporuka koje mogu pomoći u svrhu minimiziranja ranjivosti kontrole pristupa:

1. Ne preporučuje se primjena tehnika zamagljivanja („*obfuscation tehnika*") te oslanjanje na funkcionalnost klijenta prilikom oblikovanja kontrole pristupa.
2. Zadano pravilo pristupa, vezano uz svaki novostvoreni objekt, bi trebalo podrazumijevati zabranu pristupa svim postojećim korisnicima.

[25] To je dakle pod pretpostavkom da je neki od relevantnih *„HTTP zaglavlja"* definiran u sklopu zaprimljenog korisničkog zahtjeva. U suprotnom će web aplikacija nastaviti koristiti *„zaprimljeni URL"* u svrhu referenciranja odgovarajuće rute.

[26] Npr. referenciranje zabranjene rute korištenjem velikih slova umjesto malih

3. Preporučuje se provođenje opsežnog manualnog testiranja kontrole pristupa.

3.1.3 Napadi na autentifikaciju

Autentifikacija predstavlja proces verifikacije identiteta određenog korisnika ili procesa. Autentifikacija korisnika aplikacije se bazira na korištenju odgovarajućih vjerodajnica odnosno dokaza o identitetu korisnika. Vjerodajnice je moguće kategorizirati u tri glavne kategorije. Navedene kategorije se označavaju i kao faktori autentifikacije:

1. **Ono što korisnik zna** (npr. korisničko ime i lozinka)
2. **Ono što korisnik ima** (npr. sigurnosni token)
3. **Ono što korisnik je** (biometrijski parametri)

Prijetnje usmjerene na eksploataciju sustava za autentifikaciju korisnika najčešće podrazumijevaju visoku razinu rizika. Preuzimanje kontrole nad visoko-privilegiranim korisničkim računom u pravilu omogućuje pristup i modifikaciju širokog raspona potencijalno osjetljive informacijske imovine.

Ranjivosti koje omogućuju uspješno izvođenje napada na autentifikacijski sustav se mogu općenito podijeliti u dvije kategorije. Ovdje je riječ o ranjivostima temeljenima na pogreškama koje se realiziraju u fazi dizajna (slabi autentifikacijski mehanizam) te u fazi implementacije (logičke pogreške odnosno loše programiranje).

Napad grubom silom („*Brute-force napad*“) predstavlja osnovni oblik napada na autentifikacijski sustav web aplikacije. Ovdje je prvenstveno riječ o generiranju slučajnih nizova znakova s ciljem pogađanja vrijednosti korisničkog imena i odgovarajuće lozinke. Prilikom izvođenja navedenog oblika napada preporučuje se analiza logike napadnute aplikacije, analiza obrazaca te korištenje javno dostupnih informacija. Na ovaj način je moguće ograničiti prostor mogućih vrijednosti vezan uz izvođenje konkretnog napada odnosno izvesti informirani napad grubom silom. U određenim situacijama je moguće izolirati problem pretraživanja prostora korisničkog imena od problema pretraživanja povezane lozinke. Ovdje je dakle riječ o tehnici enumeracije korisničkih imena odnosno eksploataciji odgovarajućih ranjivosti aplikacije vezanih uz curenje informacija. Napad se

temelji na detekciji specifičnog statusnog koda poruke, poruke o grešci ili vremenu obrade vezanog uz procesuiranje ispravnog korisničkog imena a neovisno o korištenoj lozinki.

U svrhu obrane od online napada grubom silom na autentifikacijski sustav web aplikacije preporučuje se poduzimanje sljedećih akcija:

1. Uvođenje odgovarajućih ograničenja u sklopu procesa generiranja korisničke lozinke odnosno osiguravanje zadovoljavajuće razine kompleksnosti lozinke.
2. Uvođenje zabrane korištenja onih lozinki koje su pohranjene u sklopu javno dostupnih baza provaljenih lozinki.
3. Korisnicima se preporučuje registriranje unikatne kombinacije korisničkog imena i lozinke za svaku od korištenih usluga.
4. Svi dijelovi autentifikacijskog sustava moraju sadržavati podjednaku razinu zaštite.
5. Privremeno blokiranje korisničkog računa u slučaju detektiranja prevelikog broja neuspjelih pokušaja prijave unutar određenog vremenskog intervala.
6. Privremeno blokiranje „*IP adrese korisnika*“ u slučaju detektiranja prevelikog broja neuspjelih pokušaja prijave unutar određenog vremenskog intervala.
7. Uvođenje „*CAPTCHA testova*“.
8. Provjera implementacije pojedinih sigurnosnih mehanizama odnosno potraga za eventualnim logičkim pogreškama.
9. Korištenje generičkih informacija o pogreškama odnosno emitiranje minimalnih količina informacija vezanih uz pojedine pogreške.

Ovdje je važno obratiti pozornost na potencijalne pogreške koje mogu nastati tijekom implementacije te nedostatke vezane uz dizajn pojedinih sigurnosnih mehanizama koji otežavaju izvršenje napada grubom silom. Primjena mehanizma baziranog na privremenom blokiranju korisničkog računa može otvoriti mogućnost za ostvarenje efikasnog „*DOS napada*“. Ovaj napad potencijalno onemogućuje pristup web aplikaciji od strane njezinih korisnika. Primjena mehanizma privremenog blokiranja korisničkog računa također otvara mogućnost za ugradnju dodatnih ranjivosti koje mogu olakšati izvođenje enumeracije korisničkih imena. Ovdje je primjerice riječ o situaciji gdje sustav obavještava korisnika o privremenoj blokadi referenciranog korisničkog računa. Uz sve navedeno također postoji i mogućnost zaobilaženja navedenog sigurnosnog mehanizma, naizmjeničnim testiranjem

različitih korisničkih imena. U sklopu svake instance testiranja pojedinog korisničkog imena vrši se testiranje relativno malog broja potencijalnih lozinki. Primjena mehanizma privremenog blokiranja „*IP adrese korisnika*“ se također može relativno lagano zaobići izmjenama korištene „*IP adrese*“ odnosno korištenjem raznih logičkih propusta ugrađenih tijekom implementacije navedenog sigurnosnog mehanizma.

U sklopu poglavlja „Implementacija unaprijeđene verzije web aplikacije“ je demonstrirana analiza implementacije primjera sigurnog sustava za autentifikaciju korisnika.

3.1.4 Server-side request forgery (SSRF)

Ovaj oblik napada se temelji na iskorištavanju odnosa povjerenja između pojedinih komponenti web aplikacije i njezinog poslužitelja ili između poslužitelja web aplikacije i ostalih povezanih sustava poslužiteljske strane. Navedeni odnosi mogu biti namjerno implementirani ili mogu nastati kao rezultat pogreške vezane uz dizajn i/ili implementaciju sustava web aplikacije. Cilj napada je inducirati izvršenje određenih privilegiranih akcija uz korištenje konteksta odnosno ovlasti sustava poslužiteljske strane.

Ranjivost web aplikacije koja omogućuje izvođenje „*SSRF napada*“ se temelji na korištenju korisničkog unosa prilikom oblikovanja odgovarajućih „*HTTP/S zahtjeva*“. Navedeni zahtjevi se pritom izvršavaju iz perspektive poslužiteljske strane.

Kao primjer ranjivosti se može navesti referiranje „*REST API-a*“ treće strane, a na osnovu „*URL-a*“ odnosno podataka dostavljenih od strane klijenta. Pri tome se navedeni „*REST API*“ referira iz perspektive web poslužitelja. Kao dodatni primjer se može navesti funkcionalnost softvera za analizu web prometa, implementiranog u sklopu poslužitelja. Navedeni softver u pravilu generira zahtjeve prema „*URL-u*“ definiranom u sklopu „*Referer zaglavlja*“ korisničkog zahtjeva. Navedena akcija se izvodi u svrhu analize web mjesta u sklopu kojeg je generiran zahtjev prema web poslužitelju. U oba slučaja napadač je u mogućnosti dostaviti proizvoljan „*URL*“ prema sustavu poslužiteljske strane, što može rezultirati s izvođenjem privilegirane akcije uz korištenje ovlasti samog web poslužitelja.

U svrhu obrane od „*SSRF napada*“ predlaže se izbjegavanje scenarija koji podrazumijevaju ugradnju korisnički definiranih podataka prilikom definiranja „*HTTP/S zahtjeva*“, iz

perspektive web aplikacije. U slučaju ako je navedeni oblik injekcije nužan, predlaže se ugradnja robusnog sustava za provjeru korisničkog unosa. Navedeni sustav se temelji na korištenju „bijelih“ ili „crnih lista“ u sklopu kojih se definira skup dopuštenih odnosno zabranjenih oblika korisničkog unosa.

Ovdje je važno naglasiti da postoje razne tehnike zaobilaženja sigurnosnog mehanizma, temeljenog na validaciji korisničkog unosa. Naročito zanimljiva metoda podrazumijeva indirektno korištenje „*open redirection ranjivosti*“. Ovdje se pretpostavlja prisutnost navedene ranjivosti u sklopu „sporedne“ web aplikacije čiji „*URL*“ se nalazi na listi dopuštenih „*URL-ova*“, u kontekstu web aplikacije koja predstavlja glavnu metu napada. Pretpostavlja se da je napadač u mogućnosti oblikovati odgovarajući zahtjev koji će rezultirati s izvršenjem preusmjeravanja prema proizvoljnom web resursu. Navedeno preusmjeravanje se dakle izvršava prilikom referiranja dopuštenog „*URL-a*“, iz perspektive primarno napadnute web aplikacije. Napadač na ovaj način može preusmjeriti „*HTTP/S*“ zahtjev, generiran iz perspektive web poslužitelja“, na proizvoljno odredište.

3.2 Napadi na strani klijenta

3.2.1 Cross-site scripting (XSS)

„*Cross-site-scripting*“ predstavlja ranjivost web aplikacija koja otvara mogućnost zaobilaženja „*SOP sigurnosnog mehanizma*“. Navedena ranjivost omogućuje ostvarenje napada na pojedinog korisnika web aplikacije. Cilj napada je izvođenje odgovarajućih akcija korištenjem konteksta napadnutog korisnika. Na ovaj način je moguće ostvariti pristup potencijalno osjetljivim funkcionalnostima i informacijama web aplikacije.

„*Cross-site-scripting napad*“ se najčešće bazira na raznim metodama manipulacije napadnute web aplikacije s ciljem dostave malicioznog „*Javascript koda*“ njezinim korisnicima. Napad najčešće podrazumijeva ugradnju malicioznog „*Javascript koda*“ unutar web preglednika pojedinog korisnika web aplikacije. Navedena skripta zatim omogućuje izvršavanje akcija iz perspektive napadnutog korisnika. Rezultat navedenih akcija se zatim može također proslijediti napadaču, korištenjem ugrađene skripte.

U nastavku su opisana tri glavna tipa „*Cross-site-scripting napada*“. Svaki od navedenih napada podrazumijeva različite pristupe ugradnji malicioznog „*Javascript koda*“ unutar web preglednika napadnutog korisnika. Svi oblici napada se baziraju na injekciji malicioznog „*Javascript koda*“, korištenjem pojedinih parametara „*HTTP/S zahtjeva*“.

3.2.1.1 Reflected-cross-site-scripting (Reflected XSS)

„*Reflected XSS*“ predstavlja najjednostavniji oblik „*XSS napada*“. Ovaj oblik napada podrazumijeva određeni način manipulacije napadnutog korisnika odnosno navođenje korisnika na dostavu malicioznog „*HTTP/S zahtjeva*“ napadnutoj web aplikaciji. Napadnuta web aplikacija koristi pojedine dijelove „*HTTP/S zahtjeva*“ prilikom generiranja određenih dijelova vezanog „*HTTP/S odgovora*“. Ako referirana web aplikacija ne vrši odgovarajuće provjere nad dostavljenim parametrima (validacija i/ili filtriranje) to može rezultirati s injekcijom malicioznog sadržaja. Uspješna realizacija napada podrazumijeva oblikovanje pojedinog parametra na način koji omogućuje izlazak izvan konteksta naredbe/strukture, unutar koje se injektira njegova vrijednost. „*HTTP zahtjev*“ s odgovarajuće oblikovanim parametrima se zatim dostavlja napadnutom korisniku. To može podrazumijevati jednostavnu dostavu malicioznog linka ili ugradnju zavaravajuće odnosno prikrivene funkcionalnosti unutar određene web aplikacije pod kontrolom napadača.

3.2.1.2 Stored-cross-site-scripting (Stored XSS)

„*Stored XSS napad*“ podrazumijeva ugradnju malicioznog dijela „*HTTP/S zahtjeva*“ unutar određene strukture za trajnu pohranu podataka referirane web aplikacije. Ranjiva web aplikacija koristi navedene podatke prilikom generiranja odgovora na određene oblike korisničkih zahtjeva. Mogućnost injekcije maliciozne skripte se slično kao i kod ostalih tipova „*XSS napada*“, bazira na odsustvu odgovarajućih mehanizama za validaciju i/ili filtriranje korisničkog unosa. U ovom slučaju napadač nije primoran navesti napadnutog korisnika na izvršenje posebno oblikovanog „*HTTP/S zahtjeva*“ prema napadnutoj web aplikaciji. Korisnik pristupa ranjivoj aplikaciji na standardan način, međutim aplikacija prilikom generiranja

odgovora koristi maliciozni sadržaj, prethodno injektiran unutar njezine strukture za trajnu pohranu podataka (npr. baze podataka).

3.2.1.3 DOM-based cross-site scripting (DOM XSS)

Ovaj oblik „*XSS napada*“ se temelji na nesigurnom procesuiranju podataka od strane legitimnog odnosno originalnog „*Javascript koda*“, generiranog od strane ranjive web aplikacije. Navedeni „*Javascript kod*“ izvršava injekciju podataka sadržanih u sklopu „*HTPP/S zahtjeva*“ ili podataka pohranjenih u sklopu određene strukture za trajnu pohranu podataka. Pri tome je navedeni „*Javascript kod*“ u mogućnosti modificirati „*DOM*“ trenutno renderiranog web dokumenta. U slučaju ako navedeni podaci nisu prethodno prošli kroz odgovarajuće provjere to može rezultirati s injekcijom maliciozne skripte tijekom izvođenja određenog dijela „*Javascript koda*“ napadnute web aplikacije. Maliciozna skripta se zatim izvršava tijekom izvođenja ostatka procesa obrade navedenog web dokumenta u kontekstu web preglednika napadnutog korisnika.

3.2.1.4 Preporučene sigurnosne kontrole

U svrhu obrane od raznih oblika „*XSS napada*“ preporučuje se primjena sljedećih kontrola:

1. Validacija i filtriranje ulaznih podataka.
2. Dodatno enkodiranje izlaznih podataka, ovisno o konkretnom kontekstu.
3. Deklaracija odgovarajućih „*HTTP/S zaglavlja*“ u sklopu odgovora web aplikacije. Ovdje je primjerice riječ o „*Content-Type zaglavlju*“ koje se koristi za definiciju tipa dostavljenih podataka te „*HttpOnly zastavici*“ koja onemogućuje prijenos kolačića korištenjem određene skripte.

4. Primjena „*CSP mehanizma*[27]“.

3.2.2 Cross-origin resource sharing (CORS)

„*CORS*“ predstavlja proširenje „*SOP mehanizma*“. „*SOP (same-origin-policy)*“ predstavlja sigurnosni mehanizam web preglednika. Navedeni mehanizam ograničava skriptama, izvođenim unutar web preglednika te vezanim uz određeno izvorište, pristup podacima web preglednika vezanim uz drugačije izvorište. Izvorište podrazumijeva kombinaciju referencirane „*URL sheme*“, domene i ulaza (*„port“*). Ovdje je dakle riječ o obliku kontrole pristupa, implementirane u sklopu web preglednika.

„*CORS mehanizam*“ omogućuje relaksaciju ograničenja definiranih u sklopu „*SOP mehanizma*“. Navedena relaksacija dopušta skriptama pristup podacima web preglednika odnosno „*HTTP/S odgovorima*“ vezanim uz drugačije izvorište. Navedeni mehanizam omogućuje normalno funkcioniranje web aplikacija koje podrazumijevaju određeni oblik interakcije sa sadržajem vezanim uz web aplikacije pohranjene u sklopu različitih domena ili poddomena. Navedena interakcija se izvršava na strani klijenta posredstvom odgovarajuće skripte. Dozvole vezane uz navedeni oblik pristupa se izdaju od strane referirane web aplikacije, u sklopu „*HTTP/S odgovora*“, pri čemu se koriste odgovarajuća „*HTTP zaglavlja*“. „*Access-Control-Allow-Origin zaglavlje*“ omogućuje označavanje izvorišta kojem je dozvoljen pristup sadržaju generiranog „*HTTP/S odgovora*“. Generiranjem „*Access-Control-Allow-Credentials zaglavlja“,* referirana web aplikacija dozvoljava pristup svom „*HTTP/S odgovoru*“ čak i u slučaju ako korespondirajući zahtjev podrazumijeva prijenos kolačića od strane web preglednika. Navedeno zaglavlje može sadržavati isključivo „*true*“ vrijednost.

Neodgovorno korištenje „*CORS mehanizma*“ odnosno pogreške tijekom njegove konfiguracije, podrazumijevaju nastanak specifičnog oblika ranjivosti web aplikacija.

[27] *„CSP odnosno Content security policy“* predstavlja sigurnosni mehanizam web preglednika. Navedeni mehanizam omogućuje definiranje niza restrikcija vezanih uz pojedine resurse. Ovdje je primjerice riječ o zabrani pokretanja svih skripti osim onih koje se preuzimaju sa točno specificirane domene (gledajući iz perspektive pojedinog web mjesta). Upravljanje navedenim mehanizmom se temelji na definiranju *„Content-Security-Policy zaglavlja“*, u sklopu odgovora web aplikacije. U sklopu navedenog zaglavlja se korištenjem odgovarajućih direktiva registrira skup željenih restrikcija.

Navedena ranjivost između ostalog može omogućiti uspješnu realizaciju "*CSRF napada*" (onemogućavanje "preflight provjera") te indirektnu realizaciju „*XSS napada*".

U nastavku je opisan naročito zanimljiv način zaobilaženja „*TLS zaštite*", temeljen na iskorištavanju primjera ranjivosti implementacije „*CORS mehanizma*".

3.2.2.1 Zaobilaženje TLS zaštite temeljeno na lošoj konfiguraciji CORS mehanizma

Ovaj oblik napada je izvediv u slučaju ako napadnuta web aplikacija koristi „*HTTPS protokol*", međutim istovremeno dopušta pristup skriptama koje koriste izvorište bazirano na „*HTTP protokolu*". Pretpostavka je da je napadač u mogućnosti izvesti „*MiTM napad*[28]" u kontekstu korištenog „*HTTP kanala*". U slučaju ako korisnik aplikacije generira bilo koji oblik „*HTTP zahtjeva*", napadač može preusmjeriti korisnikov web preglednik na odgovarajuće „*HTTP izvorište*". Ovdje je dakle riječ o izvorištu kojem napadnuta aplikacija dodjeljuje prije navedeno pravo pristupa (temeljeno na korištenju „*CORS mehanizma*"). Napadač zatim presreće novi korisnički zahtjev (generiran na osnovu naredbe za preusmjeravanje) te vraća krivotvoreni odgovor uz ugradnju „*CORS zahtjeva*". Navedeni „*CORS zahtjev*" referencira napadnutu web aplikaciju odnosno „*HTTPS izvorište*". Web preglednik napadnutog korisnika dopušta izvršavanje navedenog „*CORS zahtjeva*" prema napadnutoj web aplikaciji. Skripta integrirana u sklopu krivotvorenog „*HTTP odgovora*" u konačnici transferira sve prikupljene podatke sadržane u sklopu određenog „*HTTPS odgovora*" napadnute web aplikacije.

3.2.3 Cross-site request forgery (CSRF)

„*CSRF napad*" podrazumijeva navođenje napadnutog korisnika na izvršavanje određenih akcija prema napadnutoj web aplikaciji. Ovaj oblik napada, u određenim situacijama, praktički omogućuje zaobilaženje „*SOP mehanizma*".

[28] *„Man-in-the-middle napad"* podrazumijeva mogućnost presretanja i modifikacije podataka razmijenjenih korištenjem određenog komunikacijskog kanala.

Ovdje je riječ o primjeni sličnog koncepta kao i u sklopu „*XSS napada*“. „*CSRF napad*“ međutim ne podrazumijeva ugradnju maliciozne skripte unutar napadnute web aplikacije već samo eventualnu ugradnju struktura koje omogućuju izvršavanje „jednostavnih zahtjeva“ prema web aplikaciji[29]. Pritom se koristi odgovarajući kontekst napadnutog korisnika odnosno skup kolačića pohranjenih u sklopu njegovog web preglednika.

„*CSRF napad*“ može podrazumijevati ugradnju malicioznog „*GET zahtjeva*“ unutar sadržaja napadnute web aplikacije. Navedeni „*GET zahtjev*“ je primjerice moguće injektirati u sklopu „*src atributa*“, *img taga*“ ranjive web aplikacije.

„*CSRF napadi*“ bazirani na induciranju „*POST zahtjeva*“ u pravilu ne podrazumijevaju injekciju sadržaja unutar napadnute web aplikacije. Umjesto toga se koriste posebno oblikovane web aplikacije/web stranice koje se nalaze pod potpunom kontrolom napadača. U sklopu navedenih web mjesta je moguće definirati proizvoljne forme temeljene na korištenju „*POST metode*“. Također je moguće definirati proizvoljne skripte koje će inducirati odgovarajuće „jednostavne zahtjeve“ prema napadnutoj web aplikaciji. Ovi zahtjevi će biti automatski generirani prilikom obrade zaprimljenog „*HTTP/S odgovora*“ u kontekstu web preglednika napadnutog korisnika.

3.2.3.1 „Nedostaci SOP mehanizma“

Važno je naglasiti da „*SOP mehanizam*“ dopušta izvršenje određenog skupa akcija bez odgovarajućih provjera („*preflight checks*“), korištenjem web preglednika. Ovdje je riječ o „jednostavnim zahtjevima“ („*simple requests*“) odnosno zahtjevima koji koriste sljedeće metode:

- *GET*
- *HEAD*

[29] „*XSS napadi*“ dakle podrazumijevaju preuzimanje osjetljivih informacija te izvršavanje osjetljivih funkcionalnosti vezanih uz web aplikaciju unutar koje je izvršena odgovarajuća injekcija „*Javascript koda*“. „*CSRF napadi*“ podrazumijevaju induciranje potencijalno neželjenih zahtjeva prema proizvoljnoj domeni uz korištenje konteksta napadnutog korisnika. „*Javascript kod*“ injektiran u sklopu „*XSS napada*“ se može teoretski iskoristiti i za izvršenje ciljeva „*CSRF napada*“.

- *POST*

Navedeni zahtjevi moraju zadovoljiti razna dodatna ograničenja. „*Content-Type zaglavlje*“ primjerice mora sadržavati jednu od sljedećih vrijednosti:

- *application/x-www-form-urlencoded*
- *multipart/form-dana*
- *text/plain*

Postoje i razni dodatni, manje važni, uvjeti koji identificiraju pojedini zahtjev kao „*simple request*“ odnosno rezultiraju dopuštenjem izvođenja navedenog zahtjeva bez potrebe za izvršenjem odgovarajućih provjera. Navedeni uvjeti su detaljno definirani u sklopu sljedećeg web mjesta: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#simple_requests.

Postavlja se pitanje zbog čega „*SOP mehanizam*“ dopušta izvođenje određenih oblika „*HTTP/S zahtjeva*“ koji podrazumijevaju korištenje „*POST metode*“ i to bez slanja „*preflight zahtjeva*“. Uvođenje navedenog pravila bi rezultiralo s velikom količinom nefunkcionalnih web aplikacija. Naime veliki broj web mjesta podrazumijeva slanje „*POST zahtjeva*“ prema domeni koja se razlikuje od izvorišne domene „*HTTP/S zahtjeva*“. Referirana web mjesta pri tome ne implementiraju „*CORS mehanizam*“ odnosno ne generiraju odgovarajuća „*HTTP zaglavlja*“ na osnovu kojih je moguće autorizirati pojedini zahtjev.

3.2.3.2 Preduvjeti uspješnog izvođenja „CSRF napada“

„*CSRF napad*“ se bazira na iskorištenju prethodno navedenih „nedostataka“ u dizajnu „*SOP mehanizma*“. Dostava određenih oblika „jednostavnih zahtjeva“ ponekad može rezultirati s izvršenjem privilegiranih akcija u kontekstu napadnute web aplikacije. Pri tom se dakle koristi kontekst napadnutog korisnika. U nastavku su navedeni ključni preduvjeti uspješnog izvršenja „*CSRF napada*“:

- Postoji mogućnost referiranja privilegiranih akcija web aplikacije, korištenjem „jednostavnih zahtjeva“.
- Sustav održavanja sesije se temelji isključivo na korištenju kolačića.

- Web aplikacija ne podrazumijeva korištenje nepredvidljivih parametara prilikom referiranja pojedinih privilegiranih akcija. Ovdje je primjerice riječ o dostavi korisničke lozinke u sklopu zahtjeva za promjenu lozinke.

3.2.3.3 Preporučeni sigurnosni mehanizmi koji omogućuju obranu od „CSRF napada“

U nastavku je naveden skup najčešće korištenih kontrola koje omogućuju zaštitu od uspješne realizacije „*CSRF napada*“:

- Primjena „*CSRF tokena*“.
- Primjena „*SameSite sigurnosnog mehanizma*“.
- Validacija zahtjeva bazirana na korištenju „*Referer zaglavlja*“.

„*CSRF token*“ predstavlja unikatnu, tajnu i nepredvidljivu vrijednost, generiranu od strane web aplikacije. Navedena vrijednost se generira za svaki zahtjev ili za svaku sjednicu te se dostavlja u sklopu odgovarajućeg „*HTTP/S odgovora*“. S obzirom da se navedeni „*token*“ dostavlja u sklopu tijela odgovora web aplikacije, web preglednik ga neće automatski uključiti u sklopu naknadnog zahtjeva upućenog prema predmetnoj web aplikaciji. Ako napadač ne raspolaže s informacijom vezanom uz vrijednost odgovarajućeg „*CSRF tokena*“, neće biti u mogućnosti provesti „*CSRF napad*“ nad štićenom rutom web aplikacije.

„*SameSite*“ predstavlja sigurnosni mehanizam web preglednika koji omogućuje definiranje restrikcija vezanih uz pristup pojedinim kolačićima. Korištenjem navedenog mehanizma se uvode ograničenja vezana uz uključivanje pojedinog kolačića unutar određenog „*cross-site zahtjeva*[30]“. Različite vrste ograničenja je moguće definirati u sklopu odgovora web aplikacija i to unutar zaglavlja vezanog uz definiciju pojedinog kolačića. U slučaju ako web aplikacija

[30] „*Cross-site zahtjev*“ predstavlja zahtjev koji se generira tijekom obrade „*HTTP/S odgovora*“. Stranica generiranog zahtjeva se pritom razlikuje od stranice navedenog odgovora. „Stranica“ u navedenom kontekstu predstavlja kombinaciju „*URL sheme*“, vršne domene („*TLD*“) i jedne poddomene (domena koje je za jednu razinu niža od „*TLD-a*“). Važno je primijetiti razliku između pojmova „*cross-**site** zahtjeva*“ i „*cross-**origin** zahtjeva*“.

ne definira odgovarajuće atribute u sklopu „*Set-Cookie zaglavlja*“, pojedini web preglednici mogu primijeniti zadanu restrikciju za navedeni kolačić.

Slabiji oblik zaštite je moguće implementirati korištenjem „*Referer zaglavlja*“. Ovdje je riječ o zaglavlju koje se definira prilikom generiranja odgovarajućeg „*HTTP/S zahtjeva*“. Web aplikacija može iskoristiti navedenu informaciju u svrhu detektiranja adrese web mjesta u sklopu čijeg odgovora je definiran navedeni zahtjev. U slučaju ako se navedena adresa razlikuje od adrese referirane web aplikacije te ako je referirana potencijalno osjetljiva ruta, navedeni zahtjev je preporučljivo odbaciti.

4 ALATI ZA PENETRACIJSKO TESTIRANJE WEB APLIKACIJA

Penetracijsko testiranje web aplikacija jeste jedna od metoda analize web aplikacija, orijentirana na pronalazak sigurnosnih ranjivosti odnosno pogrešaka i propusta koji ugrožavaju sigurnost analiziranog sustava. To je jedna od metoda etičkog hakiranja. Etičko hakiranje podrazumijeva skup tehnika sigurnosnog testiranja koje problemu analize sigurnosti sustava pristupaju s pozicije hipotetskog napadača. Promjena perspektive otvara mogućnost za kvalitetniju analizu sigurnosti promatranog sustava web aplikacije. Penetracijsko testiranje se temelji na izvršenju unaprijed definiranog skupa ciljeva napada koji se u pravilu odnose na područja više razine procijenjenog rizika te podrazumijeva napore usmjerene prema pronalasku odgovarajućeg skupa ranjivosti uz analizu potencijala njihove eksploatacije.

Prilikom izvođenja procesa penetracijskog testiranja web aplikacija moguće je koristiti dvije skupine alata: alate specijalizirane za analizu sigurnosti web aplikacija i univerzalne alate koji obuhvaćaju šire područje analize kao što su komunikacijske mreže, analiza sigurnosti poslužitelja i slično.

Izvršenje procesa penetracijskog testiranja bazira se na generiranju raznih oblika automatizirane interakcije između napadača i web aplikacije nad kojom se provodi navedeno testiranje. Ovaj proces podrazumijeva visoku razinu ekspertnog znanja i zahtijeva značajni utrošak vremenskih resursa visokokvalificirane radne snage.

U svrhu optimizacije procesa penetracijskog testiranja stručnjaci iz područja sigurnosti web aplikacija koriste razne alate za penetracijsko testiranje. Navedeni alati omogućuju kvalitetniju organizaciju podataka otkrivenih u sklopu procesa penetracijskog testiranja i sadrže predefinirane podatke vezane uz skup poznatih ranjivosti i potencijalnih metoda njihove eksploatacije. Alati za penetracijsko testiranje web aplikacija također omogućuju relativno jednostavnu automatizaciju raznih procesa korištenih u sklopu analize sigurnosti web aplikacije. Moguće ih je koristiti tijekom faze istraživanja potencijalnih ranjivosti gdje omogućuju automatsko otkrivanje općepoznatih potencijalnih ranjivosti web aplikacije. U sklopu faze eksploatacije moguće ih je koristiti u svrhu jednostavnijeg oblikovanja i izvršavanja raznih kategorija napada. U sklopu navedene faze, alati za penetracijsko testiranje

također omogućuju lakšu organizaciju relevantnih dokaza vezanih uz pojedini slučaj eksploatacije te automatsko generiranje odgovarajućih izvješća.

Fokus analize je usmjeren na dva najpopularnija okvira za penetracijsko testiranje web aplikacija u užem smislu riječi. Ovdje je riječ o „*OWASP ZAP*“ i „*Burp Suite okvirima*“. „*OWASP ZAP*“ predstavlja besplatan („*opensource*“) okvir za penetracijsko testiranje web aplikacija, dok „*Burp Suite*“ omogućuje besplatno korištenje samo određenog podskupa njegovih funkcionalnosti. U sklopu ovog rada analiziraju se prednosti i nedostaci navedenih alata. Izdvajaju se najvažnije funkcionalnosti alata za penetracijsko testiranje koje podupiru otkrivanje ranjivosti te izvršavanje pojedinih oblika eksploatacije web aplikacija. Posebna pozornost se pridaje alatima koji pružaju podršku prilikom izvršenja „*fuzzing metode*“.

4.1 Burp Suite

„*Burp Suite*“ predstavlja integriranu platformu koja omogućuje automatsko pasivno i dinamičko testiranje ranjivosti web aplikacija. Također sadrži niz funkcionalnosti koje podržavaju proces manualnog oblikovanja raznih oblika napada. Alat dakle pruža podršku u okviru automatskog ili manualnog procesa mapiranja web aplikacije, pronalaska sigurnosnih ranjivosti i eksploatacije na osnovu navedenih ranjivosti.

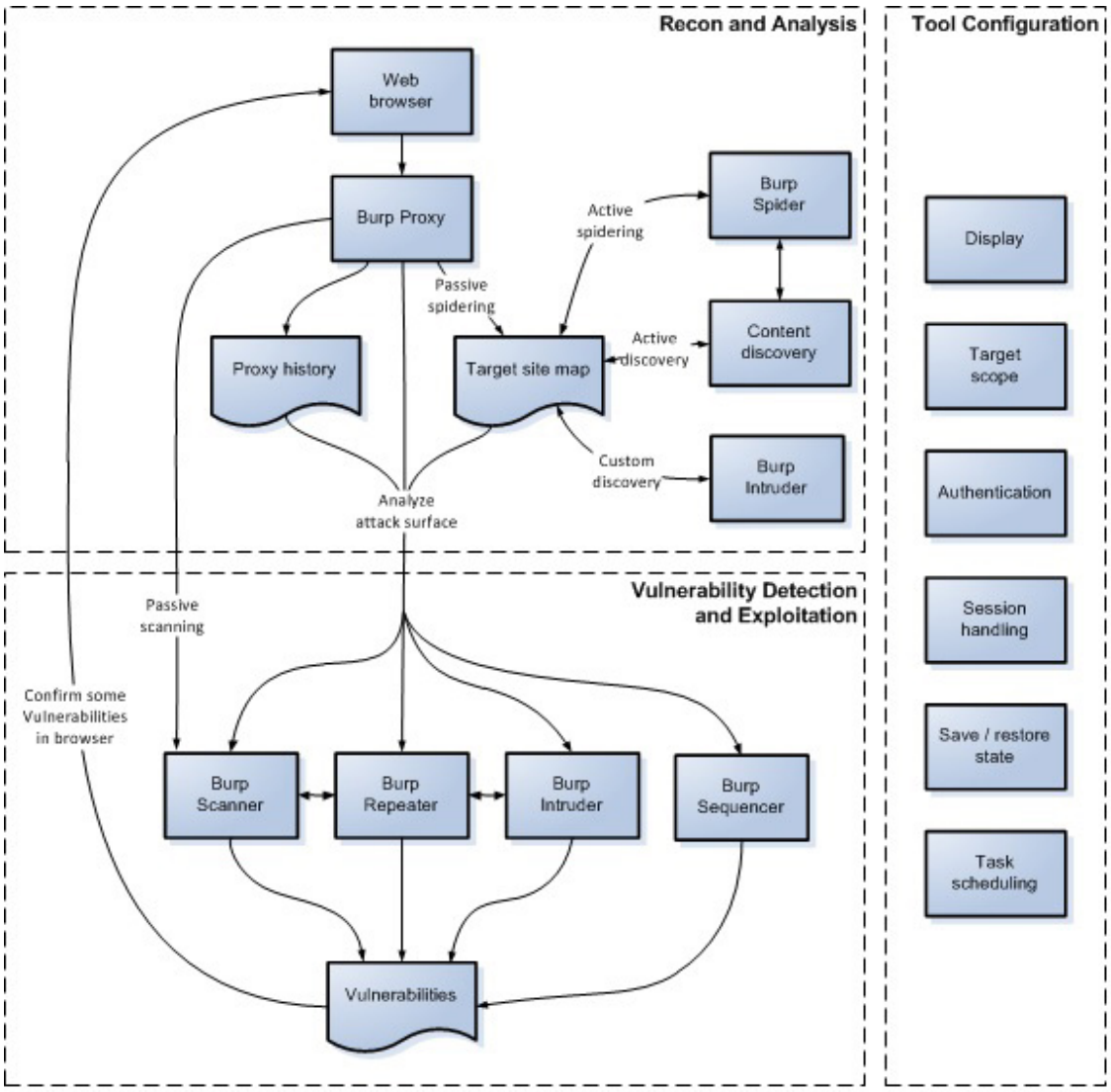
„*Burp Suite*“ ekosustav također sadrži rješenje koje je moguće koristiti u svrhu implementacije „*CI/CD sustava*“ odnosno u svrhu integracije procesa razvoja softvera i sigurnosnog testiranja. Riječ je „*Dastardly alatu*“, koji predstavlja procesorski manje zahtjevnu verziju sigurnosnog pretraživača web aplikacija.

„*Burp Suite*“ je napisan u „*Java programskom jeziku*“ te je razvijen od strane „*PortSwigger Web Security organizacije*“. Navedeni okvir se pojavljuje u tri različite varijante: „*Community Edition*“ (koji predstavlja besplatnu verziju softvera) te „*Proffesional Edition*“ i „*Enterprise Edition*“ (koji predstavljaju komercijalne verzije softvera).

U nastavku Specijalističkog rada prvenstveno se analiziraju pojedine funkcionalnosti „*Burp Suite Community Edition okvira*“. Na određenim mjestima u ovom Specijalističkom radu se

također vrši analiza izdvojenih funkcionalnosti sadržanih isključivo u sklopu komercijalnih verzija okvira.

Na sljedećoj slici su prikazane faze procesa penetracijskog testiranja web aplikacija, baziranog na korištenju „*Burp Suite okvira*“:



Slika 4.1. Proces penetracijskog testiranja (Burp Suite). Bazirano na sljedećem online sadržaju: https://portswigger.net/burp/documentation/desktop/testing-workflow

Proces penetracijskog testiranja je moguće razložiti na grupu procesa vezanih uz mapiranje i analizu površine napada te na grupu procesa vezanih uz otkrivanje ranjivosti i eksploataciju

web aplikacije. U sklopu navedenih grupa procesa koriste se odgovarajući moduli odnosno alati „*Burp Suite okvira*“. Funkcionalnosti „*Burp Scanner*“ i „*Burp Intruder alata*“ su primjenjive u kontekstu cjelokupnog procesa penetracijskog testiranja dok se ostali alati primjenjuju u sklopu jedne od prethodno definiranih grupa procesa.

4.1.1 Mapiranje i analiza površine napada

„*Burp Suite okvir*“ sadrži skup alata koje je moguće koristiti u svrhu manualnog ili automatskog mapiranja vidljivog i skrivenog sadržaja web aplikacije. Pod vidljivim sadržajem se podrazumijevaju krajnje točke koje se eksplicitno koriste odnosno koje su eksplicitno referencirane u sklopu analizirane domene (u sklopu pojedinih „*HTTP/S odgovora*“).

Mapiranje vidljivog sadržaja je moguće provesti manualno, korištenjem „*Burp proxy alata*“. Pritom je potrebno iskoristiti eventualno raspoložive vjerodajnice, u svrhu izvršenja korisničke prijave odnosno autentifikacije. Prilikom manualnog istraživanja prostora web aplikacije „Burp Suite“ analizira posjećene krajnje točke web mjesta na osnovu inicijalno postavljenog „*Live task-a* ***(Live passive crawl)***“. Pritom generira listu eksplicitno referenciranih krajnjih točaka web aplikacije, koje još uvijek nisu zaprimile korisnički zahtjev. „*Live task*“ predstavlja pozadinski proces „*Burp Suite okvira*“. Navedeni proces omogućuje automatsko pretraživanje ranjivosti vezanih uz pojedinu krajnju točku ili pronalazak te dodavanje novootkrivenih krajnjih točaka u mapu stranica („*Site map*“). Pronađene i neposjećene krajnje točke se označavaju sivom bojom u sklopu grafičkog sučelja platforme.

Automatsko izvođenje mapiranja se bazira na korištenju „*Burp Spider alata*“ odnosno web pauka integriranog u sklopu komercijalne verzije „*Burp Suite okvira*“. Ovaj alat se temelji na rekurzivnoj analizi, gdje se na osnovu sadržaja početne liste krajnjih točaka otkrivaju poveznice prema ostalim krajnjim točkama web aplikacije. Web pauk automatski upućuje zahtjeve prema novootkrivenim krajnjim točkama te nad istima ponavlja prethodno definirani postupak.

Važno je naglasiti da „*Burp Suite okvir*“ sadrži vlastiti web pretraživač odnosno „*Burp's browser alat*“. Navedeni pretraživač je inicijalno konfiguriran na način da podržava

presretanje komunikacije od strane „*Proxy alata*“. Osim toga podržava i korištenje „*HTTPS protokola*“ u kontekstu indirektne komunikacije putem „*Proxy-a*“. To znači da nije potrebno provoditi dodatnu konfiguraciju vezanu uz digitalne certifikate u sklopu navedenog pretraživača.

4.1.2 Proxy alat

„*Proxy alat*“ omogućuje presretanje, modificiranje i pohranu podataka vezanih uz komunikaciju iniciranu korištenjem „*Burp preglednika*“, odnosno općenito korištenjem „*Burp Suite okvira*“. Na osnovu zaprimljenih podataka generira se povijest korespondencije prema referenciranim krajnjim točkama. Povijest razmjene poruka se grupira u ovisnosti o korištenom protokolu. Ovdje je riječ o „*HTTP/HTTPS*“ i „*Websocket protokolima*“. Podaci vezani uz poruke razmijenjene tijekom mapiranja web aplikacije se pohranjuju i grupiraju u sklopu mape stranica. Mapa stranica sadrži hijerarhijsku reprezentaciju posjećenih „*URL-ova*“, podatke o pronađenim potencijalnim ranjivostima te podatke o zahtjevima i odgovorima vezanim uz pojedinu krajnju točku. U sklopu navedene strukture stabla pojedini „*URL-ovi*“ se grupiraju na osnovu odgovarajuće domene odnosno direktorija. Podatke vezane uz pojedine korespondencije, izlistane u sklopu mape stranica i povijesti korespondencija, je moguće proslijediti drugim alatima okvira u svrhu daljnje obrade.

„*Proxy alat*“ je moguće konfigurirati na način da prosljeđuje presretnute poruke prema zadanom poslužitelju i ulazu, neovisno o njihovom izvorno definiranom odredištu. Alat sadrži opciju korištenja „*HTTPS protokola*“ u sklopu izlazne komunikacije (usmjerene prema poslužitelju) čak i u slučajevima kada ulazni zahtjev koristi „*HTTP protokol*“. Ovdje je riječ o „*Force use of TLS opciji*“. Inverznu operaciju, odnosno transformaciju odgovora poslužitelja iz „*HTTPS*“ u „*HTTP oblik*“, moguće je definirati odabirom opcije „*Convert HTTPS links to HTTP*“. Obje opcije se koriste tijekom izvođenja „*SSL strip metode*“. Metoda pruža podršku prilikom izvođenja „*Man in the middle napada*“ u slučajevima kada referencirani poslužitelj koristi „*HTTPS protokol*“. Pretpostavka jest da je napadač prethodno eksploatirao odgovarajuću ranjivost odnosno da je ostvario pristup komunikaciji između krajnjeg korisnika i poslužitelja. „*SSL strip metoda*“ podrazumijeva transformaciju korisničkih zahtjeva na putu prema poslužitelju odnosno prevođenje korisničkog zahtjeva u oblik koji zadovoljava pravila

„*HTTPS protokola*“. Također se vrši transformacija presretnutog odgovora poslužitelja u oblik kompatibilan s „*HTTP protokolom*“. Na ovaj način se zaobilazi sigurnosni mehanizam implementiran u sklopu poslužitelja odnosno neutralizira se mogućnost korištenja „*HTTPS protokola*“ tijekom komunikacije između krajnjeg korisnika i poslužitelja. Napadač je u mogućnosti iskoristiti navedenu metodu napada u slučaju kada krajnji korisnik vrši inicijalno referenciranje poslužitelja na osnovu korištenja „*HTTP zahtjeva*“.

Kao primjer se može navesti scenarij kada krajnji korisnik u sklopu web pretraživača upisuje odgovarajuću domenu, bez specificiranja „*HTTPS protokola*“. U tom slučaju, napadač je u mogućnosti preuzeti kompletnu kontrolu nad komunikacijskim kanalom. U normalnim okolnostima krajnji korisnik će na osnovu prethodnog zahtjeva biti preusmjeren na odgovarajuću „*HTTPS krajnju točku*“ u sklopu referenciranog poslužitelja. S obzirom na činjenicu da napadač prisluškuje komunikaciju između krajnjeg korisnika i poslužitelja, on je u mogućnosti generirati odgovarajući odgovor namijenjen krajnjem korisniku. Sadržaj navedenog odgovora predstavlja „*HTTP transformaciju*“ „*HTTPS odgovora*“, koji je napadač zaprimio od strane izvorno referenciranog poslužitelja, a na osnovu proslijeđenog korisničkog zahtjeva.

U svrhu uspješnog izvršenja navedene vrste napada potrebno je izbrisati „*Secure zastavicu*“ iz presretnutog odgovora web poslužitelja. „*Secure zastavica*“ se koristi u svrhu označavanja onih kolačića koje je dozvoljeno transmitirati isključivo korištenjem sigurne konekcije („*HTTPS*“). U situaciji kada je postavljena navedena zastavica, web pretraživač ne prosljeđuje kolačiće u sklopu konekcija koje se baziraju na korištenju nesigurnog oblika komunikacijskog protokola („*HTTP*“). Važno je naglasiti da određeni web pretraživači registriraju navedenu zastavicu isključivo u sklopu uspostavljene sigurne konekcije. U suprotnom, ako je zastavica deklarirana u sklopu zaglavlja „*HTTP protokola*“, web pretraživač može ignorirati navedeno zaglavlje.

Automatsko brisanje „*Secure zastavice*“ iz presretnutog odgovora je također moguće definirati u sklopu postavki „*Proxy alata*“. Navedeni alat općenito omogućuje definiranje proizvoljne modifikacije zahtjeva, koja se primjenjuje prije njihovog prosljeđivanja referenciranom poslužitelju, odnosno modifikaciju zaprimljenih odgovora, prije njihovog prosljeđivanja krajnjem korisniku.

Postoje situacije kada je teško eliminirati „*TLS pogreške*“, koje se registriraju u sklopu klijenta uslijed neadekvatne konfiguracije „*Proxy alata*“. Ovaj problem se može pojaviti prilikom testiranja mobilnih aplikacija koje implementiraju „*SSL/TLS pinning sigurnosnu tehniku*“[31]. „*SSL/TLS pinning*“ podrazumijeva definiranje očekivanog digitalnog certifikata vezanog uz pojedini poslužitelj, u sklopu odgovarajućeg spremišta podataka korištenog od strane klijentske aplikacije (mobilna aplikacija, web pretraživač i slično). Navedeni sigurnosni mehanizam pruža zaštitu od scenarija u kojem napadač uspije na neki način pribaviti potpis pouzdanog certifikacijskog autoriteta („*CA*“) u sklopu krivotvorenog digitalnog certifikata. Aplikacija uspoređuje zaprimljeni digitalni certifikat s očekivanim (prethodno definiranim) certifikatom referenciranog poslužitelja. U slučaju ako se detektira nepoklapanje, obustavlja se konekcija između aplikacije i referenciranog poslužitelja. Ukoliko navedena aplikacija koristi više različitih domena, odnosno koristi kombinaciju „*HTTP*“ i „*HTTPS protokola*“, moguće je propustiti „*TLS komunikaciju*“ prema problematičnim poslužiteljima uz presretanje preostalog dijela komunikacije. Ovo se postiže korištenjem „*TLS pass through postavki*“ koje je moguće pronaći unutar „*Proxy settings kartice*“ sadržane unutar „*Proxy alata*“.

4.1.3 Message editor

U svrhu modifikacije pojedinih zahtjeva odnosno odgovora, vezanih uz izvršenje odgovarajućeg komunikacijskog protokola, koristi se „*Message Editor panel*“. On je sastavljen od „*Text Editor*“ i „*Inspector panela*“. „*Message Editor paneli*“ se koriste u sklopu „*Target*“ (mapa stranica), „*Repeater*“, „*Proxy*“ i „*Logger alata*“. „*Text editor panel*“ sadrži jednostavan alat koji se koristi u svrhu prezentacije i uređivanja teksta. Pri tome se mogu koristiti razni oblici reprezentacije analiziranog teksta („*Pretty*“, „*Raw*“, „*Hex*“). Također postoji opcija renderiranja pojedinih odgovora generiranih u sklopu analizirane komunikacije. Prema tome, alat sadrži mogućnost prezentacije navedenih podataka, na sličan način kao i web preglednik. Ovisno o potrebi, korisnik može aktivirati odnosno deaktivirati prikaz „*non-printable znakova*“ odnosno znakova koji se u pravilu ne prezentiraju u eksplicitnom obliku. Ovdje je riječ o posebnim nizovima znakova koji primjerice mogu označavati prelazak u novi

[31] https://www.sectigo.com/resource-library/what-is-certificate-pinning

red. Prelazak u novi red se u sklopu „*Windows okruženja*“ kodira korištenjem sljedećeg niza znakova: „\r\n“.

Uz svaki od navedenih tabova vežu se specifične akcije unutar kontekstualnog odnosno „*Actions izbornika*“. „*Actions izbornik*“ prvenstveno omogućuje prosljeđivanje pojedinih zahtjeva/odgovora u svrhu daljnje obrade unutar odabranog „*Burp Suite okvira*“. U sklopu navedenog izbornika se, između ostalog, nalazi i grupa funkcija vezana uz kodiranje odnosno dekodiranje prethodno označenog dijela teksta unutar analizirane poruke („*Convert Selection*“). Navedena grupa funkcionalnosti se često primjenjuje prilikom testiranja kvalitete filtera korisničkih zahtjeva, implementiranih u sklopu analizirane web aplikacije. „*Construct string funkcija*“ čak omogućuje automatsko generiranje koda kojeg je moguće koristiti u kontekstu testiranja navedenog filtera. Ovdje je riječ o naredbama oblikovanim na način da se tijekom njihovog izvršenja generira potencijalno maliciozni sadržaj. Navedeni kod se dostavlja u okviru korisničkog zahtjeva te se potencijalno interpretira tijekom izvođenja odgovarajućeg softvera u sklopu testiranog sustava. U slučaju ako je navedeni zahtjev zaobišao kontrole implementirane u sklopu filtera korisničkih zahtjeva, evaluacija navedenog koda može rezultirati s injektiranjem malicioznog sadržaja. Postoje različite opcije konverzije označenog teksta u odgovarajuće, potencijalno maliciozne naredbe. Pri tome je moguće testirati sustave bazirane na „*Javascript*“, „*MySQL*“, „*Oracle*“ i „*Microsoft SQL Server tehnologijama*“. Gledajući iz perspektive „*Javascript tehnologije*“, alat na osnovu označenog teksta može generirati odgovarajuću „*String.fromCharCode() metodu*“. Sličan efekt, u kontekstu „*Javascript programskog jezika*“, moguće je ostvariti i korištenjem „*eval() metode*“.

4.1.4 Inspector alat

„*Inspector alat*“ pruža podršku „*Text editor alatu*“. On omogućuje grupiranje podataka vezanih uz pojedini zahtjev/odgovor te automatski izvršava osnovno dekodiranje označenog teksta (bez potrebe za prosljeđivanjem podataka prema „*Decoder alatu*“). Ukoliko korisnik modificira dekodiranu verziju označenog teksta, sve promjene se preslikavaju unutar originalnog teksta, u sklopu „*Text Editor-a*“. Na osnovu kategorizacije, implementirane u sklopu navedenog alata, moguće je jednostavno pristupiti ključnim detaljima vezanim uz

poruke korištenog protokola. Navedeno grupiranje podataka je moguće implementirati i u sklopu samog „*Text Editor alata*“, konfiguriranjem dodatnih kartica („*tabs*“). U slučaju ako poslužitelj koristi „*HTTP/2 protokol*“, „*Inspector alat*“ opcionalno omogućuje prikaz zaglavlja, generiranih u sklopu komunikacije temeljene na navedenom protokolu. Na ovaj način se omogućuje testiranje ranjivosti baziranih na neadekvatnoj implementaciji „*HTTP/2 protokola*“, od strane poslužitelja web aplikacije. „*Text Editor*“ s druge strane transformira zaglavlja „*HTTP/2 protokola*“ u odgovarajuća zaglavlja „*HTTP/1 protokola*“ u svrhu jednostavnije upotrebe od strane korisnika. Prije slanja poruka s modificiranim zaglavljima, „*Text Editor*“ transformira „*HTTP/1 zaglavlja*“ u odgovarajuća zaglavlja koja zadovoljavaju pravila „*HTTP/2 protokola*“.

4.1.5 Target alat

Ovaj alat sadrži odgovarajuće funkcionalnosti, koje je moguće koristiti u kontekstu analize prostora napada testirane web aplikacije. On također omogućuje definiciju opsega napada te pohranu podataka, vezanih uz pojedine stranice testiranog web mjesta (mapa stranica).

Prije izvođenja procesa mapiranja web aplikacije korisno je definirati opseg napada („*target scope*“). To podrazumijeva definiranje skupa „*URL-ova*“ koji predstavljaju relevantan sadržaj, odnosno nad kojima se planira provesti odgovarajuća analiza iz perspektive konkretnog procesa penetracijskog testiranja. Navedena ograničenja je zatim moguće koristiti u svrhu konfiguracije raznih alata unutar „*Burp Suite okvira*“. Opseg napada se definira označavanjem pojedinih domena ili „*URL-ova*“, prezentiranih u sklopu mape stranica. Ograničenja je također moguće definirati korištenjem odgovarajućih pravila odnosno regularnih izraza.

Definiranjem opsega napada omogućuje se filtriranje relevantnog sadržaja u sklopu mape stranica i povijesti korespondencije. Na ovaj način se povećava preglednost korisničkog sučelja te se izbjegava nepotrebna analiza referenciranih „*URL-ova*“, koji nisu direktno vezani uz napadnutu web aplikaciju. Korištenjem opsega napada moguće je konfigurirati „*Proxy alat*“ odnosno ograničiti presretanje skupa zahtjeva i odgovora u skladu s definiranom listom „*URL-ova*“. U sklopu „*Burp Scanner-a*“, moguće je konfigurirati „*Live Task*“ na način da pretražuje ranjivosti isključivo unutar krajnjih točaka koja zadovoljavaju navedena

ograničenja. Također je moguće konfigurirati „*Intruder*“ i „*Repeater alate*“ na način da prate preusmjeravanja upućena isključivo prema „*URL-ovima*“, deklariranim u sklopu opsega napada. Uz sve navedeno, definirani opseg napada omogućuje smanjenje rizika vezanih uz eventualno izvršenje napada na web mjesta koja nisu predmet sigurnosnog testiranja.

Ovaj alat također omogućuje pristup mapama stranica, čija je primjena prethodno pojašnjena u sklopu analize „*Proxy alata*“. Ovdje je korisno izdvojiti još jednu funkcionalnost kojoj se pristupa korištenjem mape stranica. Riječ je o „*Comparing site maps funkciji*“. Ona omogućuje usporedbu sadržaja dvaju različitih mapa stranica. Navedeni proces komparacije je moguće konfigurirati, definiranjem odgovarajućih pravila, koja se koriste u svrhu utvrđivanja poveznice između zahtjeva, generiranih unutar različitih sesija. Ovdje je dakle riječ o utvrđivanju kriterija na osnovu kojih se definiraju korespondirajući zahtjevi, prisutni unutar različitih mapa stranica. Nad navedenim parovima se zatim izvršava odgovarajući proces usporedbe. Ovu funkcionalnost je prikladno koristiti tijekom testiranja potencijalnih ranjivosti, vezanih uz kontrolu pristupa. U tom kontekstu se mogu navesti sljedeći scenariji korištenja:

1. Mapiranje aplikacije korištenjem profila s različitim razinama ovlasti te zatim uspoređivanje generiranih mapa stranica. Na ovaj način je moguće identificirati dijelove funkcionalnosti koji su vidljivi u kontekstu samo jedne od korištenih sesija. Navedene dijelove funkcionalnosti je zatim potrebno testirati na odgovarajući način.
2. Drugi scenarij korištenja podrazumijeva prethodno utvrđivanje privilegiranih funkcija aplikacije. U ovom slučaju se iznova dostavlja skup zahtjeva, sadržan unutar određene mape stranica. Navedeni proces se izvršava iz perspektive korisničkog računa niže razine ovlasti odnosno korištenjem drugačije sesije. Za razliku od prvog scenarija, ovdje je riječ o direktnom testiranju mehanizma kontrole pristupa, vezanog uz detektirani skup privilegiranih funkcionalnosti. Korištenjem „*Comparing site maps funkcionalnosti*“ moguće je jednostavno provjeriti eventualnu mogućnost pristupa privilegiranim funkcijama od strane korisnika koji ne posjeduje odgovarajuću razinu ovlasti.
3. Na sličan način je moguće testirati sustav kontrole pristupa, baziran na korištenju unikatnih identifikatora osjetljivih resursa, iz perspektive svakog pojedinog korisnika. Dakle uspoređuju se rezultati mapiranja web aplikacije iz perspektive dvaju

korisničkih računa. Nakon toga se provjerava eventualno prisustvo resursa u kontekstu mape stranice vezane uz korisnika koji ne posjeduje pravo pristupa navedenom resursu.

Prilikom izvršenja navedenog oblika testiranja potrebno je na odgovarajući način konfigurirati mehanizam upravljanja sesijom. To podrazumijeva definiranje odgovarajućih pravila, primjenjivih u kontekstu određenih dijelova aplikacije. U konkretnom slučaju navedena pravila moraju referencirati „*Target alat*“. Najjednostavniji primjer konfiguracije podrazumijeva ugradnju novih instanci kolačića prije ponovljenog slanja pojedinih zahtjeva registriranih u sklopu mape stranica. Pri tome se koristi „*Cookie jar*“, odnosno spremnik kolačića. Vrijednosti spremljenih kolačića se ažuriraju tijekom izvršavanja akcija, pokrenutih korištenjem odgovarajućih „*Burp Suite alata*“. U konkretnom slučaju „*Target alat*“ mijenja vrijednost navedenog kolačića na osnovu prvog zaprimljenog odgovora web poslužitelja (generiranje novih tokena).

S obzirom na potencijalno značajne razlike prisutne unutar pojedinih implementacija mehanizama kontrole pristupa, „*Burp Suite*“ ne pokušava automatizirati kompletan proces analize navedenih mehanizama. Analiza ranjivosti prisutnih u sklopu sustava kontrole pristupa podrazumijeva izvršenje manualnog oblika testiranja (uz korištenje alata prisutnih u sklopu „*Burp Suite okvira*“).

U sklopu besplatne verzije „*Burp Suite okvira*“ moguće je jedino izvršiti usporedbu na osnovu trenutno generirane mape stranica.

„*Target alat*“ također sadrži poveznicu na odgovarajuću funkcionalnost, koju je moguće koristiti u svrhu analize prostora napada testirane web aplikacije. Riječ je o „*Target analyzer funkcionalnosti*“, sadržanoj u sklopu komercijalne verzije „*Burp Suite okvira*“. Ova funkcija omogućuje analizu ukupnog broja statičkih i dinamičkih „*URL-ova*“, sadržanih u sklopu registrirane mape stranica. Pri tome se također prikazuje broj parametara vezan uz pojedini „*URL*“ te broj „*URL-ova*“, vezanih uz pojedini parametar. Korištenjem navedenih alata, korisnik može lakše izdvojiti prioritetna područja u sklopu prostora napada te procijeniti vrijeme potrebno za izvršenje procesa penetracijskog testiranja.

Prema tome, analiza prostora napada se koristi u svrhu planiranja daljnjeg tijeka procesa penetracijskog testiranja odnosno u svrhu identifikacije dijelova aplikacije za koje se sumnja

da bi mogli sadržavati određeni skup ranjivosti. Prilikom izvođenja navedenog oblika analize prikladno je koristiti i odgovarajuće opcije filtriranja, sortiranja i označavanja pojedinih zahtjeva.

4.1.6 Mapiranje skrivenog sadržaja

U svrhu mapiranja skrivenog sadržaja koristi se „*Content Discovery alat*“. Ovdje je riječ o komercijalnom alatu odnosno funkcionalnosti koja je sadržana isključivo u sklopu plaćene verzije „*Burp Suite okvira*“. Otkrivanje skrivenih „*URL-ova*“ web aplikacije se bazira na informiranom pretraživanju odnosno ekstrapolaciji raspoloživih podataka vezanih uz posjećenu listu „*URL-ova*“. Navedena analiza se izvršava rekurzivno pri čemu se prvo emitiraju zahtjevi za koje je procijenjena najviša razina vjerojatnosti uspješnog referenciranja krajnje točke testirane web aplikacije. Novootkrivena lista krajnjih točaka je reprezentirana u sklopu zasebne mape stranica odnosno panela integriranog unutar „*Content Discovery alata*“. Navedene podatke je moguće proslijediti prema osnovnoj mapi stranica. Alat je također moguće konfigurirati na način da pokreće automatsko mapiranje dodatnog vidljivog sadržaja, na osnovu liste novootkrivenih „*URL-ova*“. Sličan rezultat je moguće postići korištenjem „*Intruder alata*“. Navedeni alat će biti detaljno dokumentiran u nastavku ovog Specijalističkog rada.

4.1.7 Automatska analiza ranjivosti i eksploatacija

„*Burp suite okvir*“ sadrži alate za automatsku i manualnu analizu ranjivosti te eksploataciju web aplikacija. U svrhu automatskog (pasivnog i aktivnog) testiranja koristi se „*Burp Scanner alat*“, sadržan u sklopu komercijalne verzije „*Burp Suite okvira*“. U nastavku je dokumentiran skup ključnih alata koje je moguće koristiti u kontekstu manualne analize ranjivosti i eksploatacije testiranog sustava. Navedene alate je također moguće koristiti u kombinaciji s rezultatima generiranim tijekom izvršenja automatskog testiranja.

4.1.8 Sequencer alat

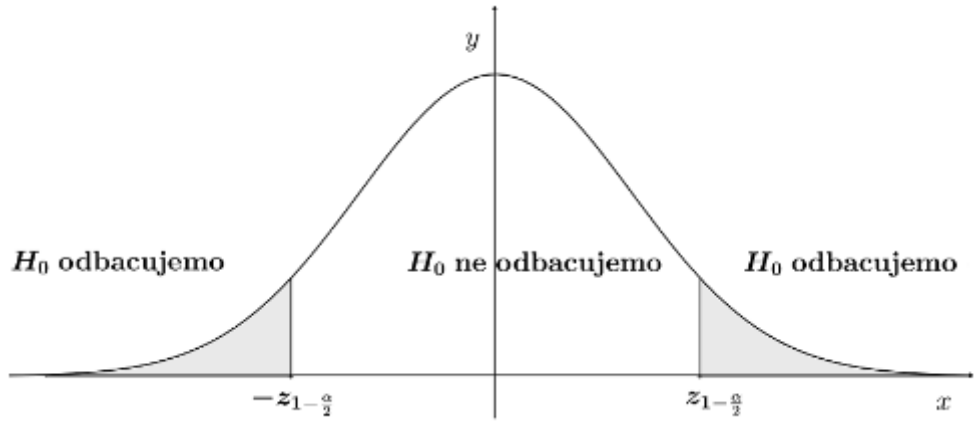
Upotrebom ovog alata analizira se kvaliteta tokena generiranih u sklopu web aplikacije. To podrazumijeva izračunavanje količine entropije odnosno neodređenosti svojstvene procesu generiranja navedenih tokena. Viša količina entropije podrazumijeva nižu razinu predvidljivosti funkcioniranja generatora tokena odnosno manju vjerojatnost predviđanja budućih vrijednosti tokena.[32]

Prije započinjanja konkretne analize potrebno je prikupiti određenu količinu tokena odnosno definirati uzorak na osnovu kojeg će biti moguće izvršiti odgovarajuću statističku analizu. Korištenje „*Sequencer alata*“ podrazumijeva odabir odgovarajućeg zahtjeva, odnosno označavanje tokena kojeg želimo testirati te pokretanje „*Live capture procesa*“. Navedeni proces podrazumijeva ponavljano emitiranje izabranog zahtjeva uz korištenje odgovarajućih modifikacija s ciljem manipuliranja referenciranog poslužitelja. Kao primjer modifikacije može se navesti brisanje tokena vezanog uz korisničku sesiju u sklopu emitiranog zahtjeva. Web aplikacija će na osnovu navedenog zahtjeva generirati novi token sesije te ga proslijediti krajnjem korisniku u sklopu „*HTTP/S odgovora*“.

Razina efektivne entropije se izračunava u ovisnosti o specificiranoj razini značajnosti testa. Razina značajnosti testa ukazuje na vjerojatnost donošenja krivog zaključka u sklopu određenog statističkog testa odnosno preciznije, pojave pogreške prvog tipa. Alat izračunava razine entropije zasebno za svaki element predefiniranog skupa razina značajnosti testa.

Testiranje statističkih hipoteza se temelji na korištenju distribucije vjerojatnosti istinitosti početne hipoteze u ovisnosti o vrijednostima izračunatima na osnovu prikupljenih uzoraka. U nastavku je prikazan primjer distribucije vjerojatnosti uz pretpostavku korištenja dvostranog testa:

[32] Kensler J, Statistical Hypothesis Testing, 2018.



Slika 4.2. Distribucija vjerojatnosti istinitosti početne hipoteze

Odluku o odbacivanju ili prihvaćanju početne hipoteze (H0) donosimo u ovisnosti o tome pripada li vrijednost uzorka kritičnom području testa. X os označava skup mogućih vrijednosti uzoraka. Y os označava interval vjerojatnosti. Granice kritičnog područja testa definirane su korištenjem odgovarajućih kvantila. Ako je F funkcija distribucije slučajne varijable X onda se rješenje jednadžbe F(x) = p naziva kvantilom reda p. Varijabla p prema tome označava zbroj vjerojatnosti istinitosti početne hipoteze, vezanih uz sve vrijednosti uzorka koje su manje od vrijednosti kvantila. U konkretnom primjeru se, u svrhu definiranja kritičnog područja testa, koristi kvantil reda (1- α/2) i kvantil reda (α/2), koji u sklopu analizirane distribucije vjerojatnosti predstavlja zrcaljenu vrijednost kvantila (1- α/2), s obzirom na Y os. Pri tome α predstavlja razinu značajnosti testa.

U slučaju ako se vrijednost uzorka nalazi između navedenih granica kritičnog područja, početna hipoteza se prihvaća. U suprotnom se odbacuje hipoteza o zadovoljavajućoj razini kvalitete pseudoslučajnog generatora tokena (vezane uz pojedini bit ili znak tokena).

Prilikom izvođenja testa hipoteza pojavljuju se dvije klase pogrešaka. Pogreška prvog tipa podrazumijeva vjerojatnost odbacivanja početne hipoteze u slučaju kada je hipoteza zapravo točna. Ovdje je riječ o razini značajnosti testa odnosno o zbroju vjerojatnosti istinitosti početne hipoteze, vezanih uz skup mogućih vrijednosti uzoraka, koji se nalaze izvan kritičnog područja testa. Pogreška drugog tipa podrazumijeva vjerojatnost prihvaćanja početne hipoteze u slučaju kada hipoteza zapravo nije točna. Jačina testa označava vjerojatnost izbjegavanja pojave greške drugog tipa te se po konvenciji referencira korištenjem sljedeće oznake: 1- β.

Pri tome oznaka β predstavlja vjerojatnost pojave greške drugog tipa tijekom testiranja statističkih hipoteza.

Dakle tijekom izvođenja testa hipoteze očekuje se da će određeni postotak testova rezultirati s pogrešnim zaključcima. Količina bitova efektivne entropije se temelji na zbrajanju onih bitova analiziranog tokena koji su zadovoljili statističku analizu.

„*Sequencer alat*“ podrazumijeva izvršavanje skupa statističkih testova nad komponentama analiziranog tokena. Pri tome se koriste dvije kategorije testova odnosno testovi temeljeni na analizi pojedinih znakova i testovi temeljeni na analizi pojedinih bitova tokena.

4.1.8.1 Testovi na razini pojedinih znakova tokena

4.1.8.1.1 Analiza broja znakova

Ovaj test se temelji na analizi distribucije znakova, vezane uz svaku pojedinu poziciju tokena sadržanih u sklopu određenog uzorka. Hipoteza slučajnog generiranja tokena implicira visoku vjerojatnost pojave uniformne distribucije znakova na svakoj od pozicija u sklopu navedenog uzorka tokena. Test rezultira prihvaćanjem početne hipoteze u slučaju kada distribucija znakova, kalkulirana na određenoj poziciji unutar tokena, ne odskače značajno od uniformne distribucije. Konkretna odluka naravno ovisi o definiranoj razini značajnosti testa.

4.1.8.1.2 Analiza tranzicije znakova

Ovaj test analizira tranziciju pojedinih pozicija tokena u sklopu generiranog uzorka. U slučaju istinitosti pretpostavke vezane uz slučajno generiranje tokena, za svaki detektirani znak u sklopu dane pozicije unutar pojedinog tokena, postoji jednaka vjerojatnost pojave bilo kojeg znaka u sklopu iste pozicije unutar sljedećeg analiziranog tokena.

4.1.8.2 Testovi na razini pojedinih bitova tokena

Testovi na razini pojedinih bitova tokena obuhvaćaju skup „*FIPS testova*“ te dodatne testove temeljene na analizi odnosa između pojedinih pozicija bitova testirane grupe tokena. „*FIPS*“ podrazumijeva skup sigurnosnih standarda definiranih od strane vlade Sjedinjenih Američkih Država. Izvršenje navedene grupe testova podrazumijeva prethodnu konverziju svakog tokena u eksplicitni niz bitova.

4.1.8.2.1 FIPS monobit test

Ovaj test se temelji na analizi distribucije binarnih vrijednosti odnosno distribucije jedinica i nula na svakoj poziciji unutar grupe tokena. U slučaju ako je skup tokena slučajno generiran, ukupan broj nula i jedinica, u sklopu svake analizirane pozicije, mora biti približno jednak.

4.1.8.2.2 FIPS poker test

Test se temelji na analizi distribucije nizova od 4 bita. Prije početka izvođenja testa, sekvence bitova, izlučene na osnovu analize pojedine pozicije, se integriraju u grupe od po četiri bita. Navedene grupe bitova se translatiraju u odgovarajući broj (16 različitih vrijednosti). U slučaju ako je skup tokena slučajno generiran, distribucija navedenih brojeva mora biti uniformna.

4.1.8.2.3 FIPS runs test

U sklopu ovog testa sekvence bitova, vezane uz pojedinu poziciju, se dijele na nizove uzastopnih bitova koji sadrže jednaku vrijednost. Nakon toga se analizira broj pojavljivanja nizova uzastopnih bitova i to u ovisnosti o duljini navedenih nizova. Ako je analizirani uzorak slučajno generiran, broj pojedinih nizova bi se trebao nalaziti u rasponu koji ovisi o veličini uzorka.

4.1.8.2.4 FIPS long run test

Ovaj test se temelji na mjerenju najdužeg niza bitova, s jednakim uzastopnim vrijednostima, vezanih uz određenu poziciju analiziranog skupa tokena. Ako je analizirani uzorak generiran slučajno, duljina navedenog niza bi se trebala nalaziti u rasponu koji ovisi o veličini uzorka.

4.1.8.2.5 Spektralni test

Ovdje je riječ o zanimljivom pristupu testiranju sekvence uzastopnih bitova vezanih uz pojedine pozicije skupa analiziranih tokena. Pri tome se izdvojeni nizovi bitova interpretiraju kao brojevi. Važno je napomenuti da se u sklopu navedenog testa provodi analiza skupa nizova uzastopnih bitova **proizvoljnih** vrijednosti. „*Sequencer alat*“ registrira svaku od analiziranih serija unutar n-dimenzionalnog prostora, pri čemu se korespondirajući brojevi pojedinih serija (vrijednosti pojedinog niza bitova) koriste kao koordinate. U slučaju kada analizirani sustav generira nepredvidljive tokene očekuje se pojava uniformne distribucije navedenih koordinata u sklopu n-dimenzionalnog prostora. Alat izvršava navedeni koncept testa zasebno za razne duljine nizova znakova i za različiti broj dimenzija. Ovaj test omogućuje otkrivanje predvidljivosti u sklopu analiziranog generatora tokena, čak i u situacijama kada većina preostalih testova rezultira s prihvaćanjem početne hipoteze.

4.1.8.2.6 Korelacijski test

Ovaj test provjerava eventualno postojanje statistički značajne veze između različitih pozicija bitova u sklopu analiziranog skupa tokena. U slučaju kada je početna hipoteza istinita, za svaku zadanu poziciju bita se pretpostavlja jednaka vjerojatnost pojavljivanja jedinica odnosno nula na bilo kojoj drugoj poziciji unutar pojedinog tokena.

4.1.8.2.7 Kompresijski test

Kompresijski test podrazumijeva transformaciju odnosno kompresiju sekvence, vezane uz svaku poziciju bita unutar analizirane grupe tokena. Pri tome se koristi „*ZLIB algoritam kompresije*“. U slučaju registriranja visoke razine kompresije vezane uz analiziranu sekvencu postoji mala vjerojatnost slučajnog generiranja vrijednosti u sklopu analizirane pozicije tokena.

4.1.8.2.8 Zaključak

Provođenje različitih statističkih testova je korisno s obzirom da svaki test analizira sekvence vezane uz pojedine pozicije tokena iz drugačije perspektive. Na ovaj način se ostvaruje sveobuhvatnija analiza kvalitete vezane uz proces generiranja tokena.

Nakon inicijalnog izvršenja skupa statističkih testova, alat omogućuje isključivanje pojedinih testova te ponavljanje izvođenja analize uz korištenje nove konfiguracije.

4.1.9 Burp Intruder

Ovaj alat omogućuje konfiguraciju odgovarajućih napada na web aplikacije odnosno automatsko generiranje i slanje grupe posebno oblikovanih „*HTTP zahtjeva*“. Pri tome se u sklopu predefiniranih pozicija, unutar svakog od navedenih zahtjeva, ugrađuju različiti „*payload-i*“. Ovdje je riječ o sadržaju koji može dovesti do izvođenja malicioznih akcija nakon njegove interpretacije u kontekstu testirane web aplikacije.

Funkcionalnosti sadržane u sklopu „*Burp Intruder alata*“ je moguće koristiti u svrhu izvođenja raznih oblika napada. Alat se prvenstveno koristi prilikom izvođenja „*brute force*“ i „*fuzzing napada*“. Također se koristi u kontekstu enumeracije identifikatora odnosno otkrivanja skrivenog sadržaja web aplikacije.

Prije izvođenja konkretnog napada potrebno je proslijediti odgovarajući „*HTTP zahtjev*“ prema „*Burp Intruder alatu*“ te zatim izvršiti odgovarajuću konfiguraciju. Prvi korak

podrazumijeva definiranje pozicija rezerviranih za ugradnju skupa „*payload-a*“, u sklopu unesenog „*HTTP zahtjeva*“. Nakon toga slijedi konfiguriranje tipa napada i „*payload-a*“. Opcionalno je moguće definirati odgovarajuća pravila vezana uz obradu pojedinog skupa „*payload-a*“ te alocirati odgovarajuće resurse odnosno konfigurirati okruženje vezano uz izvođenje pojedinog napada. To između ostalog podrazumijeva konfiguraciju takozvanog „stupca rezultata“ u sklopu kojeg se prikazuje broj relevantnih izraza pronađenih u sklopu pojedinog odgovora web poslužitelja. Relevantni izrazi se prethodno definiraju u sklopu liste izraza (pri čemu je moguće koristiti obične nizove znakova ili regularne izraze). Također je moguće konfigurirati ekstrakciju sadržaja vezanih uz određenu poziciju unutar primljenih odgovora te označiti one odgovore koji reflektiraju dostavljene „*payload-e*“. Mogućnost detektiranja refleksije dostavljenih „*payload-a*“ je korisna prilikom izvođenja testiranja korištenjem „*XSS*“ i sličnih napada, koji mogu potencijalno rezultirati s generiranjem „*payload-a*“ u sklopu odgovora web poslužitelja.

U nastavku su opisani pojedini tipovi napada i mehanizmi za generiranje „*payload-a*“, koje je moguće koristiti u sklopu „*Burp Intruder alata*“.

4.1.9.1 Tipovi „payloada-a“

„*Burp intruder*“ sadrži mogućnost odabira različitih tipova „*payload-a*“. Svaki tip „*payload-a*“ podrazumijeva specifičan proces automatskog generiranja skupa „*payload-a*“, baziranog na korisničkom unosu. U nastavku su opisane ključne značajke navedenih mehanizama odnosno mogućnosti njihove konfiguracije.

4.1.9.1.1 Simple list

Ovdje je riječ o najjednostavnijem tipu „*payload-a*“. Temelji se na korištenju liste nizova znakova, definiranih od strane korisnika sustava, bez dodatnih modifikacija.

4.1.9.1.2 Runtime file

Ova opcija omogućuje definiranje datoteke, na osnovu koje će se generirati odgovarajući skup „*payload-a*“. Korištenje datoteka je prikladno u situacijama kada raspolažemo s jako dugačkim listama nizova znakova. Mehanizam omogućuje čitanje liste iz priložene datoteke tijekom izvođenja programa. Na ovaj način se izbjegava potreba za istovremenim učitavanjem cjelokupne liste unutar radne memorije.

4.1.9.1.3 Custom iterator

Navedeni mehanizam se temelji na permutaciji znakova ili općenito liste elemenata, definiranih od strane korisnika sustava. Pri tome se koristi odgovarajući obrazac u sklopu kojeg korisnika može definirati do 8 pozicija, uz mogućnost korištenja odgovarajućih separatora. Svaku od navedenih pozicija je moguće povezati s odgovarajućom listom elemenata. Sustav zatim generira sve permutacije navedenog skupa pozicija. Pritom svaka pozicija može sadržavati različiti skup mogućih vrijednosti.

4.1.9.1.4 Character substitution

Ovaj mehanizam se bazira na supstituciji pojedinih znakova unutar liste odgovarajućih nizova znakova. On se često koristi prilikom izvođenja napada vezanih uz pogađanje lozinki odnosno u svrhu generiranja varijacija nad skupom često korištenih riječi.

4.1.9.1.5 Case modification

U sklopu navedene opcije, moguće je izabrati određenu metodu vezanu uz transformaciju velikih/malih slova na odgovarajućim pozicijama u sklopu korisnički definirane liste znakova.

4.1.9.1.6 Recursive Grep

Ovaj tip „*payload-a*“ se najčešće koristi u kontekstu izvršenja „*SQL injekcije*“. Kao primjer se može navesti injektiranje malicioznog zahtjeva s ciljem pribavljanja objekata sadržanih u sklopu proizvoljne baze podataka unutar testiranog sustava. Pri tome se često injektira sljedeći zahtjev:

„UNION SELECT name FROM sysobjects WHERE name > 'a'“

S obzirom da se kao prvi parametar tablice baze podataka, u pravilu koristi identifikacijski broj, poslužitelj će vratiti odgovarajuću pogrešku. Greška proizlazi iz činjenice što instance rezultata, vraćene u sklopu injektirane „*SELECT naredbe*“, ne predstavljaju „*integer tip podataka*“. Poslužitelj ponekad vraća informativnu poruku o pogrešci, u sklopu koje je sadržana vrijednost prve instance rezultata, evociranog korištenjem injektirane „*SQL naredbe*“. U nastavku je demonstriran primjer relevantne poruke o pogrešci[33], uz pretpostavku da prva instanca rezultata, kalkuliranih na osnovu injektirane „*SQL naredbe*“, sadrži vrijednost: „otkrivena_vrijednost“.

„Syntax error converting the varchar value ' otkrivena_vrijednost ' to a column of data type int.“

„*Recursive grep funkcionalnost*“ je moguće koristiti u svrhu automatskog rekurzivnog otkrivanja sadržaja, baziranog na prethodno pojašnjenom procesu. U konkretnom se primjeru vrijednost, sadržana s desne strane „*> operatora*“, rekurzivno zamjenjuje s prvim pronađenim rezultatom prethodno injektirane „*SELECT naredbe*“.

4.1.9.1.7 Illegal Unicode

Ova funkcionalnost omogućuje generiranje skupa „*palyoad-a*“, na osnovu zadane liste elemenata, pri čemu se u sklopu pojedinih znakova koriste ilegalni oblici kodiranja (bazirani na „*Unicode standardu*“). Generirani oblik „*payload-a*“ omogućuje zaobilaženje određenih sigurnosnih mehanizama (filtera). Zaobilaženje sigurnosnih filtera se općenito bazira na

[33] https://portswigger.net/blog/using-recursive-grep-for-harvesting-data

eksploataciji eventualnih razlika vezanih uz mehanizam dekodiranja, koji se koristi u sklopu različitih komponenti analiziranog sustava. Ovdje je primjerice riječ o razlikama mehanizama dekodiranja, implementiranih u sklopu vatrozida/"*front-end poslužitelja*" i „*backend poslužitelja*". U slučaju postojanja navedenih razlika, komponente sustava zadužene za filtriranje korisničkih zahtjeva, mogu propustiti potencijalno maliciozni sadržaj. Navedeni sadržaj se zatim interpretira u svom eksplicitnom obliku odnosno dekodira se na drugačiji način, prilikom izvršenja ostalih komponenti sustava. To u konačnici rezultira uspješnim izvršenjem malicioznog koda u kontekstu napadnutog sustava.

4.1.9.1.8 Character blocks

Ovaj mehanizam omogućuje generiranje multipliciranog oblika specificiranog znaka odnosno nizova znakova. Najčešće se koristi prilikom testiranja „*buffer overflow ranjivosti*". Prilikom konfiguracije navedenog mehanizma, definira se početni niz znakova, minimalna i maksimalna multiplikacija početnog niza znakova te korak odnosno razlika između duljina dvaju uzastopnih blokova.

4.1.9.1.9 Brute forcer

Ova opcija omogućuje generiranje permutacija niza znakova, uz prethodnu konfiguraciju duljine niza i korištenog skupa znakova, vezanog uz sve pozicije generiranog niza.

4.1.9.1.10 Null payloads

Ovdje je riječ o generiranju skupa „*payload-a*", koji se sastoji od skupa praznih „*string-ova*" odnosno praznih nizova znakova. Navedeni mehanizam je prikladno koristiti u sklopu sekvencijalne analize (oblikovanje zahtjeva koji iniciraju regeneraciju odgovarajućih tokena) i prilikom testiranja ranjivosti sustava u kontekstu izvođenja „*DOS napada*".

4.1.9.1.11 Character frobber

Ovaj mehanizam omogućuje izvršavanje specifičnog oblika modifikacije vrijednosti u sklopu svake pozicije unutar korisničkog unosa. To podrazumijeva uvećanje vrijednosti „*ASCII koda znaka*“, vezanog uz svaku pojedinu poziciju korisničkog unosa, za jedan. U sklopu svake instance generiranja payload-a se mijenja jedan znak korisničkog unosa. Korisnički unos se pritom odnosi na skup nizova znakova označenih s „*payload markerima*“ odnosno na niz znakova koji je eksplicitno definiran u sklopu konfiguracije navedenog mehanizma.

Navedeni alat je prikladno koristiti prilikom analize utjecaja pojedinih dijelova korisničkog zahtjeva na oblik odgovora, generiranog od strane web aplikacije. Moguće je primjerice analizirati strukturu tokena sesije, s ciljem registriranja dijelova tokena, koji se koriste u sklopu poslužiteljske strane i to u svrhu praćenja stanja navedene sesije. U slučaju ako poslužitelj generira jednaki odgovor, nakon modifikacije pojedinog znaka unutar tokena sesije, tada postoji velika vjerojatnost da se navedeni dio tokena ne obrađuje prilikom utvrđivanja stanja korisničke sesije.

4.1.9.1.12 Bit flipper

Riječ je o varijaciji „*Character frobber mehanizma*“, koja omogućuje ostvarivanje više razine kontrole, u kontekstu modifikacije pojedinih znakova korisničkog unosa. Ovaj alat dakle omogućuje modifikaciju pojedinog znaka vezanog uz korisnički unos i to na razini pojedinog bita. Prilikom konfiguracije, korisnik specificira odgovarajuće pozicije bitova. Na ovaj način se definiraju mjesta unutar skupa znakova korisničkog unosa, nad kojima se izvršava odgovarajuća modifikacija. Navedena modifikacija podrazumijeva izmjenu vrijednosti „*0*“ u vrijednost „*1*“ i obratno.

4.1.9.1.13 Username generator

Korištenjem ove opcije, korisnik može ekstrapolirati skup imena, potencijalno korištenih u sklopu testiranog sustava, na osnovu predefinirane liste imena. Početni podaci se

transformiraju u oblike kompatibilne sa skupom uobičajeno korištenih shema, koje se koriste prilikom generiranja korisničkih imena/email adresa i slično.

4.1.9.1.14 ECB block shuffler

Ovaj mehanizam omogućuje izvođenje napada baziranih na eksploataciji ranjivosti sadržane u sklopu „*ECB algoritma*“. Ovdje je dakle riječ o nesigurnom algoritmu šifriranja podataka. On se temelji na kriptiranju pojedinih blokova izvornog teksta, neovisno o ostatku izvornog/kriptiranog teksta.

Korištenjem „*ECB block shuffler mehanizma*“ korisnik može modificirati redoslijed pojedinih blokova šifriranog teksta. Postoji određena vjerojatnost da će izvršenje navedene akcije rezultirati modifikacijom parametara korištenih u sklopu odgovarajuće sesije. Moguće je primjerice promijeniti vrijednost parametra, koji web aplikacija koristi u svrhu identifikacije trenutnog korisnika. Ovo podrazumijeva da konkretna aplikacija ne vrši odgovarajuću analizu nad kompletnim sadržajem zaprimljenog tokena sesije.

Dodatni problem vezan uz navedeni algoritam za šifriranje podataka proizlazi iz činjenice što se jednaki blokovi originalnog niza znakova prevode u jednaki šifrirani niz. Ovo svojstvo olakšava izvršavanje odgovarajućih metoda kriptoanalize.

4.1.9.1.15 Copy other payload

Ovaj mehanizam se koristi prilikom izvršenja napada koji podrazumijevaju korištenje više skupova „*payload-a*“. On omogućuje kopiranje pojedinog skupa “*payload-a*“ s jedne na drugu „*payload poziciju*“.

Ovo je korisno u situacijama kada određeni parametri, korišteni u sklopu izvršenja napada baziranih na korištenju različitih skupova „*payload-a*“, moraju sadržavati jednaku vrijednost, u sklopu svake pojedine instance generiranog zahtjeva. Mehanizam se općenito koristi u slučajevima kada je potrebno uspostaviti odgovarajuću vezu između navedenih parametara. Kao primjer se može navesti slučaj kada određeni parametar sadrži „*hash vrijednost*“,

izračunatu na osnovu vrijednosti nekog drugog parametra, korištenog u sklopu navedenog tipa napada. Ovaj scenarij korištenja, podrazumijeva dodatnu konfiguraciju tipa „*payload-a*“ odnosno definiranje odgovarajućeg pravila za procesuiranje skupa „*payload-a*“ (korištenje „*hash funkcije*“). „*Burp Inruder alat*“ uz navedeno sadrži i tipove „*payload-a*“, u sklopu kojih je moguće konfigurirati odgovarajuće nizove brojeva odnosno datuma. Također je moguće koristiti skup „*payload-a*“, generiran od strane odgovarajuće „*Burp ekstenzije*“.

4.1.9.2 Tipovi napada

Jedan od koraka vezanih uz konfiguraciju „*Burp Intruder alata*“ podrazumijeva odabir konkretnog tipa napada. Tip napada definira način korištenja skupa „*payload-a*“ odnosno način na koji se pojedini „*payload-i*“ povezuju s odgovarajućim „*payload pozicijama*“, definiranim u sklopu trenutnog „*HTTP/S zahtjeva*“. Pri tome je moguće koristiti jedan ili više skupova „*payload-a*“. Sustav je moguće konfigurirati na način da se „*payload-i*“ pojedinačno dodjeljuju svakoj od deklariranih „*payload pozicija*“, na način da se pojedini „*payload-i*“ dodjeljuju istovremeno skupu deklariranih „*payload pozicija*“ i slično. U nastavku su opisana četiri tipa napada odnosno četiri opcije koje je moguće odabrati tijekom konfiguracije „*Burp Intruder alata*“.

4.1.9.2.1 Sniper

Ovaj oblik napada podrazumijeva ugrađivanje jedne instance skupa „*payload-a*“, unutar jedne od mogućih „*payload pozicija*“, u sklopu pojedinog „*HTTP/S zahtjeva*“. U sklopu navedenog napada koristi se jedan skup „*payload-a*“. Pri tom se generira zasebni „*HTTP/S zahtjev*“ za svaku kombinaciju instance „*payload-a*“ i moguće „*payload pozicije*“. Dakle ukupan broj zahtjeva je jednak umnošku ukupnog broja „*payload pozicija*“ i „*instanci payload-a*“ sadržanih unutar „*skupa payload-a*“.

Navedeni tip napada je prikladno koristiti u slučaju ako želimo izolirati utjecaj promjene pojedinog parametara korisničkog unosa na odgovor web aplikacije.

4.1.9.2.2 Battering ram

Ovaj oblik napada podrazumijeva istovremenu ugradnju jedne instance „*skupa payloada*“, unutar svih „*payload pozicija*“, definiranih u okviru pojedinog „*HTTP/S zahtjeva*“. Pri tome se koriste instance sadržane unutar jednog „*skupa payload-a*“. Ukupan broj zahtjeva, generiran tijekom izvođenja napada, je jednak ukupnom broju instanci sadržanih unutar „*skupa payload-a*“.

„*Battering ram napad*“ pronalazi primjenu u situacijama kada je potrebno koristiti jednake vrijednosti parametara na različitim pozicijama unutar određenog „*HTTP/S zahtjeva*“, primjerice unutar kolačića i tijela navedenog zahtjeva.

4.1.9.2.3 Pitchfork

U sklopu navedenog tipa napada, koristi se više različitih „*skupova payload-a*“. Napad podrazumijeva naizmjeničnu iteraciju kroz različite „*skupove payload-a*“. Iz pojedinog „*skupa payload-a*“ se izuzima jedna instanca „*payload-a*“ te se ista ugrađuje unutar sljedeće slobodne „*payload pozicije*“, unutar određenog „*HTTP/S zahtjeva*“. Nakon toga se izvršava jednaki slijed akcija, ali korištenjem sljedećeg „*skupa payload-a*“. Svaki generirani „*HTTP/S zahtjev*“ podrazumijeva ugradnju novih instanci „*payload-a*“ unutar svih „*payload pozicija*“. Ukupan broj generiranih zahtjeva je jednak broju instanci sadržanih unutar najmanjeg „*skupa payload-a*“.

Navedeni mehanizam pronalazi primjenu u situacijama kada je potrebno ugraditi različiti ali koreliran sadržaj unutar pojedinih „*payload pozicija*“. Kao primjer se može navesti ugrađivanje korisničkog imena unutar prve pozicije te povezanog identifikacijskog broja u sklopu sljedeće „*payload pozicije*“

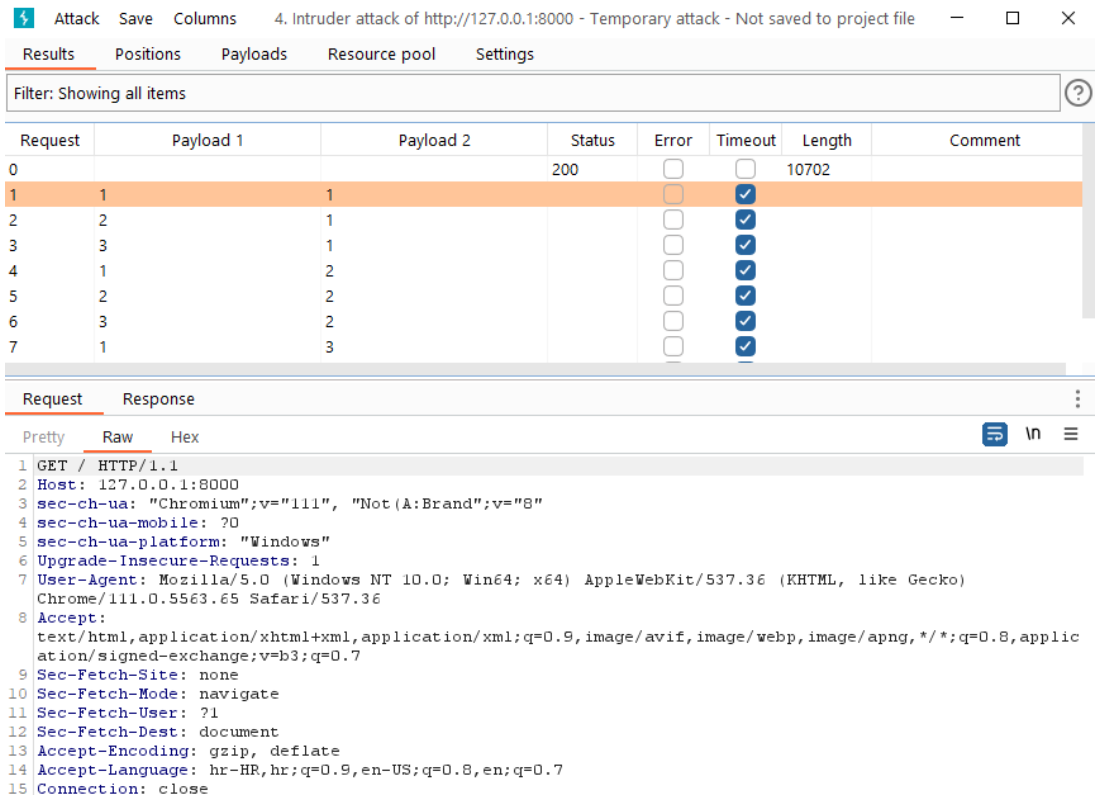
4.1.9.2.4 Cluster bomb

U sklopu ovog napada se također koristi više različitih „*skupova payload-a*“. Svaka „*payload pozicija*“ može sadržavati bilo koju od vrijednosti, sadržanih unutar pridruženog „*skupa payload-a*“. Napad rezultira s ugrađivanjem svih mogućih kombinacija vrijednosti vezanih uz pojedine „*payload pozicije*“. Prema tome, ukupan broj generiranih „*HTTP/S zahtjeva*“ je jednak umnošku ukupnog broja instanci vezanog uz svaki pojedini „*skup payload-a*“.

Ovaj napad je prikladno koristiti u situacijama kada ne postoji poznata veza između pojedinih parametara odnosno „*payload pozicija*“. Kao primjer se može navesti izvršenje napada grubom silom s ciljem pogađanja kombinacije korisničkih imena i lozinki.

4.1.9.3 Analiza rezultata izvođenja napada

Nakon pokretanja napada, otvara se novi prozor u sklopu kojeg se prikazuju rezultati izvođenja napada („*results kartica*“). Navedeni prozor također sadrži kopiju sadržaja svih konfiguracijskih kartica, definiranih prije pokretanja navedenog napada. U nastavku je prikazan primjer navedenog prozora, generiran na osnovu pokretanja „*cluster bomb napada*“ uz korištenje „*number payload-a*“:



Slika 4.3. Primjer izvođenja "cluster bomb napada"

Svaki pojedini redak, reprezentiran u sklopu prikazane tablice, referencira pojedinu kombinaciju zahtjeva i odgovora, generiranih tijekom izvođenja napada. Svaki redak sadrži odgovarajući skup stupaca, koji prezentiraju ključne podatke vezane uz pojedini zahtjev odnosno odgovor.

Zanimljive odgovore web poslužitelja odnosno rezultate izvođenja napada je moguće identificirati analizom odgovarajućih stupaca, generiranih u sklopu navedene tablice.

U slučaju ako odgovor sadrži „*HTTP statusni kod*“, koji se razlikuje od većine ostalih odgovora, tada postoji velika vjerojatnost da je vezani „*HTTP/S zahtjev*“ rezultirao s uspješnim izvršenjem napada na web aplikaciju. To znači da je potrebno izvršiti detaljnu analizu konkretne kombinacije zahtjeva i odgovora. Do istog zaključka se dolazi u slučaju ako:

1. Odgovor sadrži količinu znakova koja se razlikuje od količine znakova sadržane u sklopu većine ostalih odgovora.
2. Odgovor sadrži ili ne sadrži određeni skup izraza.

3. Odgovor vraća odgovarajuću pogrešku.
4. Odgovor podrazumijeva duže vrijeme čekanja u odnosu na većinu ostalih odgovora.

4.1.10 Burp Repeater

Ovaj alat omogućuje modifikaciju određenog „*HTTP/S zahtjeva*“ te slanje modificiranih oblika navedenog zahtjeva prema web poslužitelju. Navedenu transformaciju je moguće ostvariti korištenjem „*Proxy alata*“, međutim u tom slučaju je potrebno iznova manualno referencirati početni „*HTTP/S zahtjev*“, nakon slanja svake njegove modificirane instance. Za razliku od „*Burp Intruder alata*“, ovaj alat omogućuje precizniju konfiguraciju pojedinih aspekata izvođenja napada. Ovo se prvenstveno odnosi na konfiguraciju tipa korištene konekcije prilikom slanja grupa „*HTTP/S zahtjeva*“.

„*Burp Repeater alat*“ se prvenstveno koristi u situacijama kada nismo unaprijed sigurni na koji način želimo oblikovati zahtjev, odnosno kada navedena odluka ovisi o rezultatu prethodno iniciranih zahtjeva. Dakle u situacijama kada nemamo unaprijed poznatu listu zahtjeva koje želimo pokrenuti ili smo već izveli automatizirani oblik napada korištenjem „*Burp Intruder-a*“ te nastavljamo s istraživanjem koristeći manualni pristup.

Alat je moguće koristiti u kontekstu potvrđivanja prethodno utvrđenih ranjivosti (primjerice korištenjem „*Burp scanner-a*“) odnosno u svrhu daljnje analize zanimljivih odgovora, generiranih tijekom korištenja ostalih alata u sklopu „*Burp suite okvira*“. Uz navedeno, koristi se i u svrhu jednostavnijeg definiranja i automatskog izvršavanja sekvenci „*HTTP/S zahtjeva*“ (koji se međusobno razlikuju u značajnoj mjeri). Navedene sekvence zahtjeva se mogu dostaviti korištenjem zajedničke konekcije ili zasebno, korištenjem posebne konekcije za svaki pojedini zahtjev. Mogućnost korištenja zajedničke konekcije, se često koristi tijekom analize mogućnosti uspješnog izvršavanja „*client-side desync napada*“.

4.2 OWASP ZAP

„*OWASP ZAP*“ predstavlja grupu alata razvijenih specifično u svrhu pružanja podrške procesu penetracijskog testiranja web aplikacija. Riječ je o sustavu otvorenog koda koji je razvijen i održavan od strane neprofitne organizacije „*OWASP (Open Worldwide Application Security Project)“*. Navedeno svojstvo omogućava analizu detalja vezanih uz implementaciju „*OWASP sustava*“. Ovo može biti korisno u kontekstu razvoja ekstenzija te pronalaska i ispravljanja eventualnih pogrešaka prisutnih u sklopu određenih funkcionalnosti „*OWASP ZAP okvira*“.

„*OWASP ZAP*“ u osnovi predstavlja „*man-in-the-middle proxy*“ odnosno sustav koji presreće komunikaciju na liniji između web preglednika i referencirane web aplikacije. U slučaju kada testirani sustav koristi dodatne „*proxy poslužitelje*“, „*OWASP ZAP*“ je moguće konfigurirati na način da presreće komunikaciju na liniji između web preglednika i prvog „*proxy poslužitelja*“ referenciranog sustava.

Ovaj okvir omogućuje trajnu pohranu podataka vezanih uz pojedinu sesiju penetracijskog testiranja. U slučaju ako ne želimo pohraniti navedene podatke u sklopu lokalne baze podataka („*HSQLDB*“), moguće je odabrati opciju privremene pohrane. Važno je naglasiti da navedena opcija nije raspoloživa u sklopu besplatne verzije „*Burp Suite okvira*“.

U nastavku će biti opisane pojedine faze procesa penetracijskog testiranja, uz korištenje „*OWASP ZAP okvira*“. To podrazumijeva dokumentiranje procesa mapiranja te analize ranjivosti web aplikacije i njezine eksploatacije. Cilj penetracijskog testiranja pritom podrazumijeva pronalazak odgovarajućih ranjivosti odnosno validaciju nepostojanja navedenih ranjivosti u sklopu testirane web aplikacije.

4.2.1 Osnovni pojmovi

Prije započinjanja analize pojedine web aplikacije potrebno je izvršiti odgovarajuću konfiguraciju sustava. U svrhu lakšeg razumijevanja detalja vezanih uz navedeni proces,

potrebno je pojasniti određeni skup ključnih pojmova, korištenih u sklopu „*OWASP ZAP okvira*“.

4.2.1.1 Kontekst i korisnik

„*OWASP okvir*“ omogućuje registriranje pojedinih web korisnika. Podaci o korisnicima se koriste u svrhu izvršenja pojedinih funkcija okvira, iz perspektive pojedinih korisnika. Korisnici se definiraju u sklopu pojedinog konteksta („*Context*“). Kontekst pritom podrazumijeva određeni, prethodno definirani, skup „*URL-ova*“. Kontekst se u pravilu koristi u svrhu označavanja pojedinog stabla stranica vezanog uz pojedinu testiranu web aplikaciju te se definira kao skup regularnih izraza. Pri tome se navedeni regularni izrazi moraju preklapati s cjelokupnim sadržajem pojedinih „*URL-ova*“ koje želimo označiti kao dijelove pojedinog konteksta. Definicije korisnika se dakle prvenstveno koriste u svrhu automatskog generiranja odgovarajućih „*HTTP/S zahtjeva*“ iz perspektive pojedinog korisnika (definiranog u sklopu pojedinog konteksta). Pojedini „korisnik“ se definira i koristi u ovisnosti o konfiguraciji funkcionalnosti „autentifikacije“ i „upravljanja sesijom“. Dakle uz svaku definiciju „korisnika“ se vezuje odgovarajuća metoda autentifikacije i upravljanja sesijom. Navedeni mehanizmi se aktiviraju prilikom izvršenja odgovarajućih akcija iz perspektive pojedinog „korisnika“. Oni se prvenstveno koriste u svrhu automatskog održavanja korisničke sesije u sklopu određenog „konteksta“. Prilikom definicije pojedinog „korisnika“ otvara se dijalog čiji sadržaj ovisi o prethodno konfiguriranoj metodi autentifikacije.

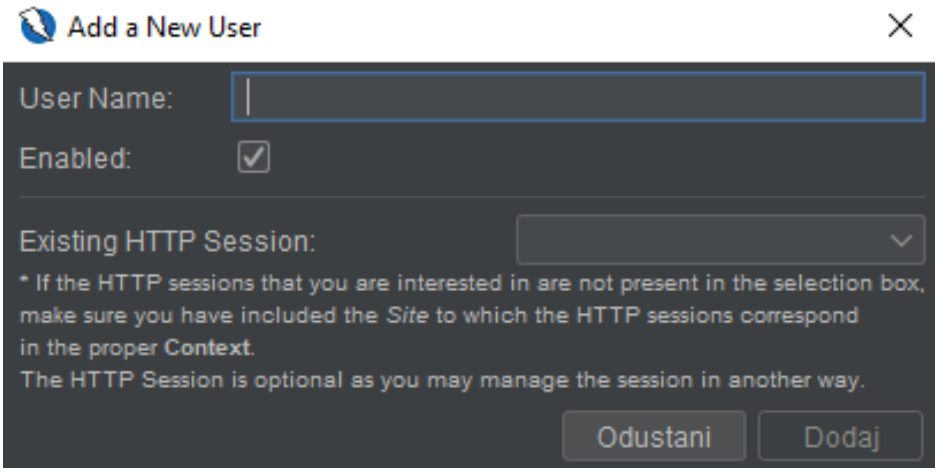
4.2.1.2 Sustav za autentifikaciju

Sustav za autentifikaciju podrazumijeva konfiguriranje odgovarajuće autentifikacijske metode i strategije za verifikaciju autentifikacije.

Autentifikacijska metoda opisuje način autentifikacije korisnika odnosno način dodjeljivanja autentificirane sesije pojedinom „korisniku“. U nastavku su pojašnjene pojedine metode autentifikacije implementirane u sklopu „*OWASP ZAP okvira*“.

4.2.1.2.1 Manual Authentification

Ova metoda podrazumijeva manualno izvršenje autentifikacije korisnika. Zahtjevi vezani uz navedeni proces autentifikacije se pritom presreću korištenjem „*proxy alata*“, integriranog unutar „*OWASP ZAP okvira*“. Nakon izvršenja autentifikacije, generiranu sesiju je moguće povezati s odgovarajućim predefiniranim „korisnikom“. Također je moguće definirati novog „korisnika“ te ga povezati s navedenom autentificiranom sesijom. Pri tome se otvara dijalog sljedećeg oblika:



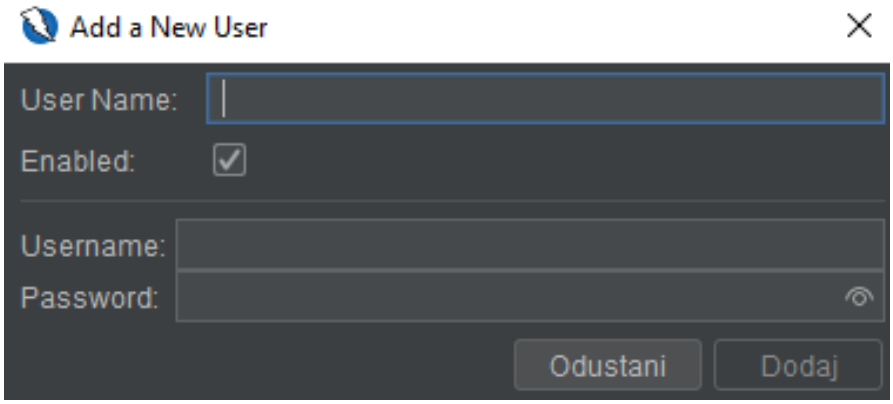
Slika 4.4. Definiranje novog korisnika (Manual Authentification)

U sklopu navedenog dijaloga se definira naziv novog „korisnika“ te je istog moguće povezati s jednom od prethodno generiranih autentificiranih sesija. Korištenjem ove metode nije moguće konfigurirati automatsku autentifikaciju korisnika. To zapravo znači da nije moguće osigurati ispravan nastavak izvršavanja odgovarajućih alata, u slučaju ako web aplikacija izvrši odjavu trenutnog korisnika.

4.2.1.2.2 Form-Based Authentication

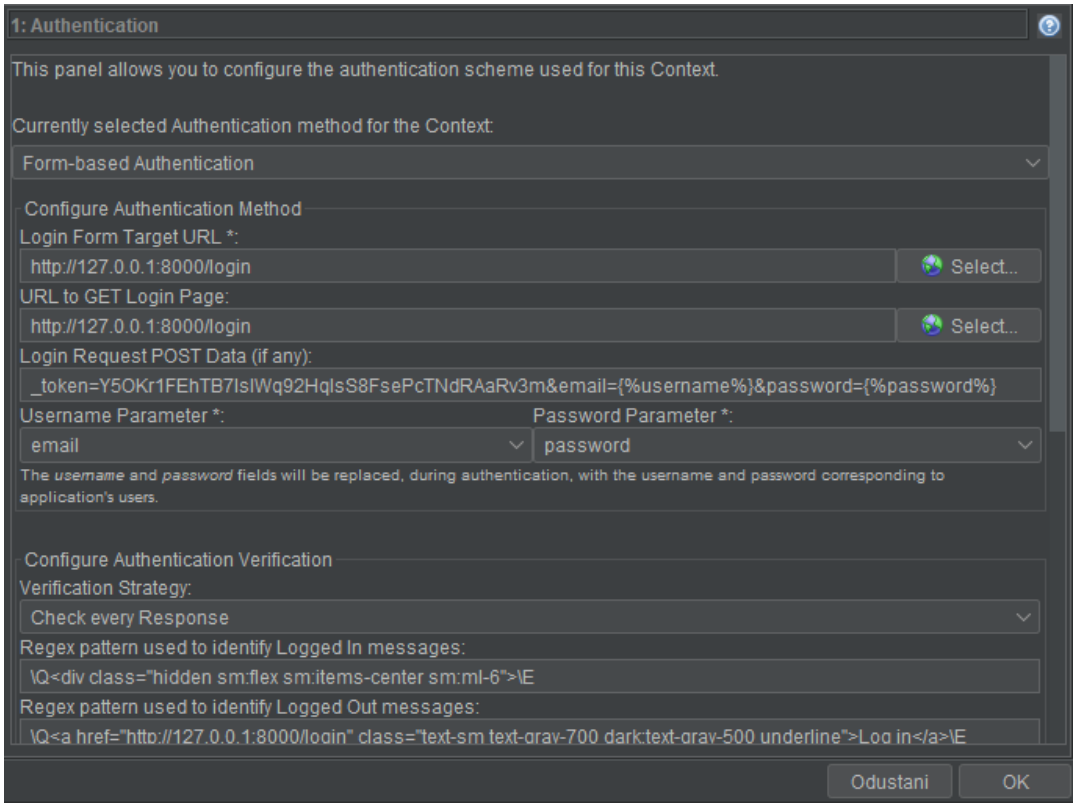
Navedena metoda se koristi prilikom testiranja web aplikacija, čiji se autentifikacijski sustav na strani klijenta bazira na korištenju „*HTML formi*“. Ona ujedno predstavlja najčešće

korištenu autentifikacijsku metodu te omogućuje konfiguraciju automatske autentifikacije korisnika. Prilikom definicije novog korisnika, uz pretpostavku korištenja navedene metode autentifikacije, otvara se sljedeći dijalog:



Slika 4.5. Definiranje novog korisnika (Form-Based Authentification)

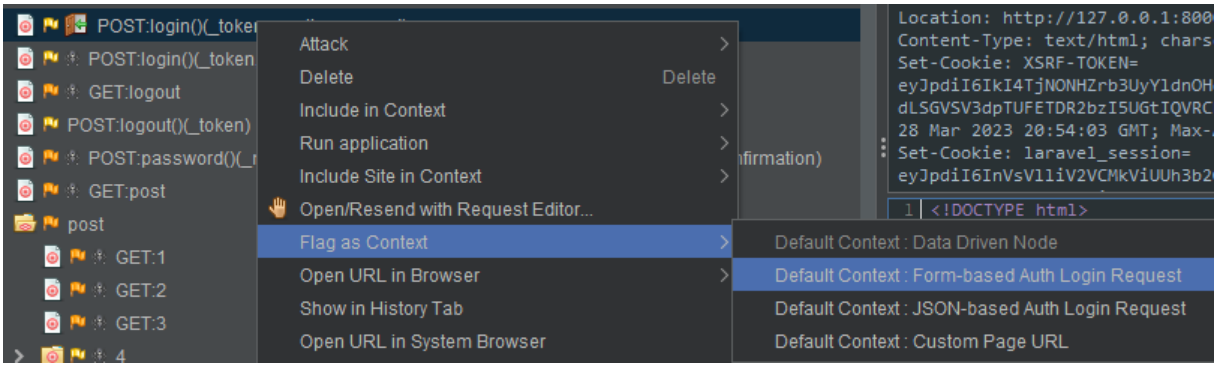
U sklopu navedenog dijaloga, definira se naziv novog „korisnika“ te se također unosi odgovarajući skup vjerodajnica. U konkretnom slučaju je riječ o korisničkom imenu i lozinki. Navedeni podaci se pohranjuju u sklopu strukture podataka vezane uz pojedinog korisnika te predstavljaju nužan preduvjet za ispravno izvršenje mehanizma automatske autentifikacije korisnika. Tijekom izvršavanja procesa automatske autentifikacije, vrijednosti navedenih parametara se ugrađuju u sklopu odgovarajućih pozicija, definiranih tijekom konfiguracije navedene metode autentifikacije. U nastavku je demonstriran primjer navedene konfiguracije:



Slika 4.6. Konfiguracija Form-Based Authentification metode

Prilikom konfiguracije korisnik, između ostalog, definira odgovarajuća imena parametara („*Username Parameter*“ i „*Password Parameter*“). Na ovaj način se dakle registriraju imena parametara koji se, u sklopu testirane web aplikacije, koriste za označavanje korisničkog imena odnosno lozinke.

U slučaju ako smo prethodno izvršili autentifikaciju odgovarajućeg korisnika te ako je navedena autentifikacija snimljena u sklopu „*OWASP ZAP okvira*“, možemo označiti relevantni zahtjev za prijavu, korištenjem opcije „*Flag as Context :: Form-based Auth Login Request*“:



Slika 4.7. Označavanje zahtjeva za prijavu

U ovom slučaju, sustav automatski definira većinu polja vezanih uz konfiguraciju metode autentifikacije, bazirane na korištenju „*HTML formi*“. Također se registriraju pojedini nazivi parametara, korišteni u sklopu označenog zahtjeva korisničke prijave. Navedene nazive parametara je moguće odabrati u sklopu padajućeg izbornika, tijekom definicije vrijednosti „*Username Parameter*“ i „*Password Parameter polja*“.

U situacijama kada aplikacija koristi „*anti-CSRF token*“, potrebno je registrirati naziv navedenog tokena, konfiguriranjem „*Anti CSRF Tokens opcije*“. U sklopu navedene opcije se registrira skup najčešće korištenih naziva „*anti-CSRF tokena*“. Navedena lista predstavlja dio izvornih postavki „*OWASP ZAP okvira*“.

4.2.1.2.3 JSON-Based Authentication

Ova metoda autentifikacije se primjenjuje prilikom testiranja web aplikacija koje implementiraju autentifikacijski sustav, baziran na korištenju „*JSON objekta*“. Navedeni „*JSON objekt*“ sadrži odgovarajuće vjerodajnice te se dostavlja u sklopu „*HTTP/S zahtjeva*“, upućenog krajnjoj točci autentifikacijskog sustava. Ova metoda također implementira mehanizam za automatsku autentifikaciju korisnika te podrazumijeva konfiguriranje odgovarajućih vjerodajnica prilikom definiranja novog „korisnika“.

4.2.1.2.4 HTTP/NTLM Authentication

Navedena metoda se primjenjuje prilikom testiranja web aplikacija koje implementiraju autentifikacijski sustav, baziran na korištenju odgovarajućih „*HTTP zaglavlja*“.

Moguće ju je koristiti u svrhu automatske autentifikacije korisnika te također podrazumijeva konfiguraciju odgovarajućih vjerodajnica u sklopu definicije pojedinog „korisnika“.

4.2.1.2.5 Script-Based Authentication

U slučaju kada testirana web aplikacija implementira složeni odnosno nestandardni autentifikacijski sustav, potrebno je definirati odgovarajuće skripte, u svrhu automatskog upravljanja s navedenim autentifikacijskim procesom. Korištenje ove metode autentifikacije omogućuje ugradnju odgovarajuće skripte za automatsku autentifikaciju korisnika. Definiranje pojedinog korisnika pri tome ovisi o definiciji navedene skripte.

4.2.1.2.6 Strategija verifikacije korisnika

U slučaju kada želimo konfigurirati sustav automatske autentifikacije korisnika, uz konfiguraciju odgovarajuće metode za autentifikaciju, potrebno je konfigurirati i strategiju za verifikaciju korisnika. Strategija za verifikaciju korisnika predstavlja način na koji se registrira autentificirana sesija odnosno provjerava stanje vezano uz autentifikaciju korisnika. Moguće je koristiti pristup temeljen na analizi svakog pojedinog zahtjeva ili odgovora te kombinacija zahtjeva i odgovora. Navedena analiza se odnosi na usporedbu sadržaja pojedinih zahtjeva i odgovora s odgovarajućim regularnim izrazom. Regularni izraz pritom referencira dio sadržaja zahtjeva ili odgovora, na osnovu kojeg je moguće zaključiti je li trenutni korisnik prijavljen ili odjavljen iz testirane web aplikacije. Također je moguće konfigurirati strategiju koja se bazira na analizi odgovora iniciranih korištenjem samog sustava za autentifikaciju. Pri tome se definira frekvencija emitiranja dodatnih zahtjeva prema

prethodno definiranom „*URL-u*“. Ovdje se pretpostavlja da navedeni „*URL*“ sadrži odgovarajući sadržaj na osnovu kojeg je moguće izvesti zaključak o tome je li korisnik trenutno autentificiran u sklopu testirane web aplikacije.

Korištenje svake strategije za verifikaciju autentifikacije podrazumijeva definiciju jednog ili dvaju regularnih izraza. Pri tome se prvi regularni izraz koristi u svrhu registriranja stanja prijave a drugi u svrhu registriranja stanja odjave korisnika. Navedene regularne izraze je moguće registrirati unosom odgovarajućih vrijednosti u sklopu „*Authentication panela*“ ili označavanjem dijelova pojedinih zahtjeva i/ili odgovora. U svrhu označavanja sadržaja pojedinih poruka, koristi se „*Flag as Context:: Logged in/out indicator opcija*“.

Prije pokretanja procesa automatskog mapiranja ili skeniranja mape stranica, potrebno je konfigurirati odgovarajući mehanizam za automatsku autentifikaciju korisnika. Proces automatskog skeniranja naime referencira sav vidljivi sadržaj u sklopu određenog web mjesta. To znači da će referencirati i „URL“, vezan uz funkcionalnost odjave testirane web aplikacije. Sustav testirane web aplikacije može izvršiti odjavu korisnika i u drugim slučajevima, primjerice u slučaju detektiranja neuobičajenog prometa.

4.2.1.3 ZAP modes

U nastavku su pojašnjeni različiti načini rada, koje je moguće odabrati prilikom konfiguracije „*ZAP alata*“.

4.2.1.3.1 Safe mode

Nije dozvoljeno izvršavati potencijalno opasne akcije.

4.2.1.3.2 Protected mode

Dozvoljeno je izvršavanje potencijalno opasnih akcija, ali isključivo nad listom „*URL-ova*“, definiranih u sklopu „opsega napada“.

4.2.1.3.3 Standard mode

Dozvoljeno je izvršavanje potencijalno opasnih akcija tijekom referenciranja bilo kojeg „*URL-a*“.

4.2.1.3.4 ATTACK mode

Dozvoljeno je izvršavanje potencijalno opasnih akcija. Navedena konfiguracija također podrazumijeva izvršenje aktivnog skeniranja pojedinog „*URL-a*“, odmah nakon njegove registracije korištenjem manualnog ili automatskog pretraživanja testiranog web mjesta.

4.2.1.4 Pravila skeniranja

„*OWASP ZAP*“ sadrži skup pravila vezanih uz izvršenje pasivnog ili aktivnog skeniranja. Ovdje je zapravo riječ o konkretnom kodu (napisanom u „*Java programskom jeziku*“), na osnovu kojeg se izvršava analiza ranjivosti i eksploatacija web aplikacija. Navedeni kod je sadržan u sklopu odgovarajućih ekstenzija. Inicijalna konfiguracija okvira podrazumijeva instalaciju skupa ekstenzija, uključujući skup ekstenzija vezanih uz definiciju pravila aktivnog i pasivnog skeniranja.

„*ZAP*“ sadrži mogućnost konfiguriranja navedenih pravila, a to se prvenstveno odnosi na definiciju praga i snage pojedinog pravila.

Prag pravila se odnosi na vjerojatnost generiranja upozorenja vezanog uz određenu ranjivost, tijekom izvršavanja navedenog pravila. Viša razina praga pravila rezultira generiranjem manjeg broja potencijalnih ranjivosti, odnosno s filtriranjem većeg broja potencijalnih problema vezanih uz testirani sustav. Na ovaj način se smanjuje količina lažno pozitivnih rezultata vezanih uz proces pasivnog odnosno aktivnog testiranja web aplikacije. Također je moguće deaktivirati pojedino pravilo, odabirom „*OFF vrijednosti*“ u sklopu padajućeg izbornika vezanog uz definiciju praga pravila.

Snaga pravila definira broj napada odnosno zahtjeva koji će se generirati prilikom korištenja navedenog pravila. Odabir više razine snage pravila, prema tome rezultira dužim izvođenjem aktivnog skeniranja. Navedenu opciju je naime moguće konfigurirati isključivo u kontekstu pravila aktivnog skeniranja.

Uz navedeno postoji mogućnost dodatnog konfiguriranja određenog skupa pravila, koje ovisi o korisničkom sučelju implementiranom u sklopu odgovarajućih ekstenzija.

4.2.1.5 Opseg napada

„Opseg napada“ podrazumijeva registraciju „*URL-ova*“ na osnovu kojih se, korištenjem odgovarajuće konfiguracije, definiraju ograničenja vezana uz pojedine alate „*OWAS ZAP okvira*“. Registriranje „opsega napada“, podrazumijeva odabir skupa „konteksta“ odnosno indirektnu definiciju skupa „*URL-ova*“. To se razlikuje od pristupa korištenog u sklopu „*Burp Suite okvira*“, gdje opseg napada podrazumijeva direktnu definiciju odgovarajuće liste „*URL-ova*“.

4.2.1.6 Upravljanje sesijom

Svaki „kontekst“ podrazumijeva odgovarajuću konfiguraciju opcije upravljanja sesijom („*Session Management*“) odnosno odabir odgovarajuće metode za upravljanje sesijom. Navedena metoda definira način dostave prethodno registriranih identifikatora sesije odnosno način održavanja korisničke sesije. Pri tome je moguće izabrati metodu baziranu na korištenju kolačića ili „*HTTP zaglavlja*“. Također postoji opcija odabira metode bazirane na korištenju skripti, definirane od strane korisnika sustava. Svi „korisnici“, definirani u sklopu određenog „konteksta“, referenciraju jednaku metodu za upravljanje sesijom.

4.2.2 Mapiranje sadržaja web aplikacije

Mapiranje sadržaja web aplikacije, korištenjem „*OWASP ZAP okvira*“, prvenstveno se bazira na upotrebi „*Spider alata*“. Riječ o ekstenziji navedenog okvira, koja se koristi u svrhu automatskog pretraživanja vidljivog sadržaja testirane web aplikacije. Prije izvođenja navedenog procesa potrebno je definirati skup početnih krajnjih točaka („*seeds*“). Izvršavanje „*Spider alata*“ započinje analizom sadržaja navedenog skupa krajnjih točaka. U sklopu navedene analize registriraju se poveznice na nove krajnje točke testirane web aplikacije. Ovaj proces se izvršava rekurzivno te završava nakon registriranja odnosno mapiranja kompletnog vidljivog sadržaja vezanog uz testirano web mjesto. Pri tome se poštuju odgovarajuća ograničenja definirana u sklopu „opsega napada“. Upotreba „*Spider alata*“, također rezultira s aktiviranjem procesa pasivnog skeniranja, koji se izvršava nad listom registriranih „*HTPP/S odgovora*“. Navedeni alat je moguće konfigurirati na način da se proces pretraživanja izvršava iz perspektive određenog autentificiranog korisnika.

U svrhu pretraživanja sadržaja web aplikacija, baziranih na korištenju „*AJAX tehnologije*“, moguće je koristiti „*AJAX Spider ekstenziju*“. Navedena ekstenzija omogućuje analizu sadržaja, generiranog kao rezultat izvršenja odgovarajućeg koda u sklopu web pretraživača odnosno klijenta.

„*Burp Suite okvir*“ sadrži slične funkcionalnosti, implementirane uz pomoć „*Burp scanner alata*“. Ove funkcionalnosti su međutim implementirane isključivo u sklopu komercijalne verzije okvira.

Sadržaj web aplikacije je moguće mapirati i presretanjem komunikacije, generirane tijekom izvršenja manualnog pretraživanja. U tu svrhu se, prilikom manualnog pretraživanja, najčešće koristi jedan od web pretraživača integriranih u sklopu samog okvira. Moguće je koristiti i bilo koji drugi web pretraživač, međutim u tom slučaju je potrebno provesti odgovarajuću konfiguraciju (preusmjeravanje prometa i ugradnja odgovarajućih digitalnih certifikata).

U svrhu otkrivanja skrivenog sadržaja web aplikacije, koristi se „*Forced Browse alat*“. Izvršenje navedene ekstenzije se bazira na pogađanju naziva krajnjih točaka web aplikacije. Pri tome se koristi odgovarajuća datoteka koja sadrži listu potencijalnih naziva direktorija i datoteka pohranjenih u sklopu testiranog web poslužitelja. Navedena lista se, prilikom

izvršenja alata, dodaje na kraj prethodno odabrane bazne putanje odnosno „*URL-a*“ testirane web aplikacije. Alat je moguće konfigurirati na način da se pretraživanje izvršava rekurzivno, u sklopu svakog novootkrivenog direktorija. Rezultati navedenog pretraživanja se prikazuju u sklopu mape stranica. Pojedini rezultat je moguće dodatno konfigurirati na način da se onemogući njegov prikaz u sklopu mape stranica. Uz navedeno moguće je zabraniti izvršavanje „*Spider alata*“ te mehanizma aktivnog skeniranja nad odabranim rezultatom pretraživanja.

4.2.3 Analiza ranjivosti i eksploatacija web aplikacije

„*OWASP ZAP okvir*“ sadrži skup alata koje je moguće koristiti u svrhu izvršavanja manualne odnosno automatske analize ranjivosti te eksploatacije web aplikacije.

U kontekstu automatske analize ranjivosti, prvenstveno se koriste alati za pasivno i aktivno skeniranje web aplikacije.

4.2.3.1 Pasivno skeniranje

Pasivno skeniranje podrazumijeva automatsko registriranje potencijalnih ranjivosti vezanih uz testiranu web aplikaciju, na osnovu analize grupe zaprimljenih „*HTTP/S zahtjeva i odgovora*“. Navedena analiza se izvršava bez slanja posebno oblikovanih zahtjeva odnosno potencijalno opasnog sadržaja prema web poslužitelju. Osim detektiranja potencijalnih ranjivosti, proces pasivnog skeniranja omogućuje i automatsko označavanje određenog skupa zahtjeva, na osnovu prethodno izvršene konfiguracije. To podrazumijeva automatsko dodjeljivanje skupa „*tagova*“ (kratki niz znakova) onim zahtjevima koji su uspješno referencirani, korištenjem odgovarajućih regularnih izraza. U nastavku je pojašnjen izdvojen skup pravila za pasivno skeniranje, sadržanih u sklopu inicijalne konfiguracije „*OWASP ZAP okvira*“.

4.2.3.1.1 Anti-clickjacking Header

Ovo pravilo se bazira na provjeri postojanja odnosno ispravne definicije odgovarajućih „*HTTP zaglavlja*“, u sklopu skupa analiziranih odgovora web aplikacije. Riječ je o validaciji sadržaja „*X-Frame-Options zaglavlja*“ odnosno prisustva „*frame-ancestors direktive*“ u sklopu „*Content-Security-Policy zaglavlja*“. Navedena analiza se izvršava s ciljem otkrivanja eventualne mogućnosti izvođenja „*Clickjacking napada*“ u sklopu određenog dijela testirane web aplikacije.

4.2.3.1.2 Cookie HttpOnly

Uz pomoć navedenog pravila se vrši provjera ispravne konfiguracije zaglavlja kolačića, u kontekstu zaštite od eventualnog izvršenja oblika „*XSS napada*“. Prisustvo „*HttpOnly zastavice*“, u sklopu „*Cookie zaglavlja*“, onemogućava iniciranje slanja kolačića korištenjem „*Javascript koda*“. Na ovaj način se onemogućava neautorizirano slanje kolačića, čak i u slučaju uspješnog injektiranja maliciozne skripte u sklopu pojedinog odgovora napadnute web aplikacije.

4.2.3.1.3 Cookie Secure Flag

Navedeno pravilo podrazumijeva analizu kolačića definiranih u sklopu „*HTTPS sesije*“. Pravilo generira odgovarajuće upozorenje u slučaju ako navedeni kolačići ne sadrže „*Secure zastavicu*“.

„*Secure zastavica*“ se koristi u svrhu označavanja onih kolačića koje je dozvoljeno transmitirati isključivo korištenjem sigurne konekcije („*HTTPS*“). U situaciji kada je postavljena navedena zastavica, web preglednik ne prosljeđuje kolačiće u sklopu konekcija, koje se baziraju na korištenju nesigurnog oblika komunikacijskog protokola („*HTTP*“). Važno je naglasiti da web preglednik registrira navedenu zastavicu isključivo u sklopu uspostavljene sigurne konekcije. U suprotnom, ako je zastavica deklarirana u sklopu zaglavlja „*HTTP protokola*“, web preglednik ignorira navedeno zaglavlje.

Navedena zastavica se koristi kao sigurnosni mehanizam u kontekstu obrane od eventualnog izvršenja „*Man-in-the-middle napada*“ odnosno preuzimanja kolačića sesije korištenjem “*HTTP konekcije*”.

4.2.3.2 Aktivno skeniranje

Aktivno skeniranje podrazumijeva automatsko izvršavanje poznatih napada odnosno slanje potencijalno opasnog sadržaja prema web aplikaciji, u svrhu registriranja njezinih potencijalnih ranjivosti. U nastavku je pojašnjen izdvojen skup pravila za aktivno skeniranje, sadržanih u sklopu inicijalne konfiguracije „*OWASP ZAP okvira*“.

4.2.3.2.1 Cross Site Scripting (reflected)

Ovo pravilo podrazumijeva detektiranje lokacija unutar odgovora web aplikacije u sklopu kojeg se renderira određeni, prethodno dostavljeni, korisnički unos. Navedene pozicije se detektiraju na osnovu slanja određenog skupa sigurnih vrijednosti, u sklopu zahtjeva upućenog prema testiranoj web aplikaciji. U ovisnosti o detektiranom okruženju svake pojedine lokacije, generira se skup posebno oblikovanih napada odnosno emitira se odgovarajući skup „*HTTP/S zahtjeva*“. Svaki od navedenih zahtjeva može rezultirati s izvršenjem malicioznog koda u sklopu web pretraživača pojedinog korisnika web aplikacije. To je pod pretpostavkom da je sadržaj dostavljenog „*payload-a*“ omogućio izlazak izvan konteksta određene programske strukture odnosno da je omogućio injektiranje malicioznog koda u sklopu generiranog odgovora web aplikacije. Ovaj oblik napada podrazumijeva dostavu posebno oblikovanog „*URL-a*“ nekom od korisnika navedene web aplikacije.

4.2.3.2.2 Cross Site Scripting (persistent)

Ovo pravilo se koristi u svrhu testiranja mogućnosti izvršenja napada baziranih na injekciji malicioznog koda u sklopu odgovarajućeg (trajnog) spremišta podataka određene web

aplikacije. Navedeno testiranje se ne razlikuje bitno od procesa opisanog prilikom analize „*Cross Site Scripting (reflected) pravila*“. Ključna razlika je u tome što „*Cross Site Scripting (persistent) pravilo*“ podrazumijeva pretraživanje cjelokupne web aplikacije (korištenjem „*Spider alata*“), s ciljem pronalaska svih lokacija, unutar kojih se renderira dostavljeni „sigurni“ sadržaj. To znači da se prilikom izvođenja navedenog pravila, provjeravaju odgovori određenog skupa krajnjih točaka testirane web aplikacije.

4.2.3.2.3 htaccess Information Leak

Navedeno pravilo se koristi u svrhu testiranja mogućnosti preuzimanja „*.htaccess datoteke*“, pohranjene u sklopu web poslužitelja testirane web aplikacije. Navedena datoteka može omogućiti pristup tajnim informacijama u slučaju kada se unutar nje pohranjuju podaci vezani uz korisnička imena, upravljanje s greškama, preusmjeravanja i slično.

4.2.3.2.4 Buffer Overflow

Ovo pravilo omogućuje detektiranje „*Buffer Overflow ranjivosti*“ web aplikacije. Bazira se na generiranju te ugradnji dugačkih nizova znakova u sklopu pojedinih parametara korisničkog unosa, unutar pojedinog „*HTTP/S zahtjeva*“. Nakon toga se izvršava analiza naknadnog izvršavanja web aplikacije s ciljem detektiranja eventualnog pada sustava odnosno neuobičajenog načina zatvaranja korisničke sesije.

4.2.3.3 Pregled alata za provođenje manualne analize ranjivosti i eksploataciju

U svrhu detektiranja dodatnih ranjivosti testirane web aplikacije moguće je primijeniti razne metode vezane uz manualno penetracijsko testiranje. Pojedine oblike ranjivosti je naime teško detektirati korištenjem automatskog procesa. To se prvenstveno odnosi na ranjivosti koje proizlaze iz propusta nastalih prilikom implementacije sustava za kontrolu pristupa. „*OWASP*

ZAP okvir" sadrži skup alata koje je prikladno koristiti u svrhu izvršenja procesa manualne analize ranjivosti web aplikacije.

„*Requester alat*" je moguće koristiti u svrhu izvršavanja detaljne analize komunikacije, vezane uz pojedinu krajnju točku testirane web aplikacije. On dakle omogućuje kontinuirano slanje modificiranih oblika početnog zahtjeva te analizu generiranih odgovora. Funkcionalnost navedenog alata se ne razlikuje u značajnoj mjeri od funkcionalnosti implementirane u sklopu „*Burp Repeater alata*".

U svrhu izvršavanja usporedbe sadržaja dvaju zahtjeva odnosno odgovora, koristi se funkcionalnost implementirana u sklopu „*Diff alata*". Pojedine nizove znakova unutar zahtjeva odnosno odgovora je moguće zamijeniti s novim nizom znakova, korištenjem „*Replacer alata*".

U slučaju ako želimo omogućiti prikaz skrivenih polja ili aktivirati prethodno deaktivirana polja forme, prikladno je koristiti „*Reveal alat*".

„*Form Handler*" omogućuje automatsku definiciju vrijednosti polja formi sadržanih u sklopu pojedinog zahtjeva. To je pod pretpostavkom da navedene vrijednosti nisu definirane korištenjem nekog od drugih alata „*OWAS ZAP okvira*". Pri tome se koriste prethodno definirane liste parova naziva pojedinih polja formi i njima pridruženih vrijednosti.

U nastavku su pojašnjeni detalji vezani uz funkcioniranje odnosno konfiguraciju ključnog alata, korištenog u kontekstu manualnog penetracijskog testiranja.

4.2.3.4 Fuzzer

„*Fuzzer alat*" omogućuje izvršavanje „*fuzzing tehnike*" [34] odnosno generiranje i dostavljanje skupa neispravnih odnosno neočekivanih podataka prema testiranoj web aplikaciji.

Ovaj alat podrazumijeva označavanje početnog „*HTTP/S zahtjeva*" na osnovu kojeg će se generirati odgovarajući skup zahtjeva s ugrađenim „*payload-ima*". „*Payload*" općenito

[34] „*Fuzzing*" predstavlja tehniku testiranja koja se svodi na slanje neispravnih, neočekivanih ili slučajnih podataka u sklopu zahtjeva te analizu odgovora od strane aplikacije u svrhu identifikacije ranjivosti.

predstavlja skup podataka, koji mogu rezultirati s izvođenjem malicioznih akcija, nakon njihove interpretacije u kontekstu testirane web aplikacije.

„*HTTP/S zahtjeve*“ je moguće na različite načine proslijediti na daljnju obradu prema „*Fuzzer alatu”:*

- Označavanjem pojedinog zahtjeva desnim klikom miša te odabirom opcije „*Attack::Fuzz...*“
- Označavanjem niza znakova, sadržanog u sklopu pojedinog zahtjeva, desnim klikom miša te odabirom opcije „*Fuzz…*“
- Odabirom odgovarajućeg zahtjeva u sklopu „*Tools::Fuzz opcije*“.

U nastavku su opisane ključne funkcije „*Fuzzer alata*“.

4.2.3.4.1 Payload Generators

„*Fuzzer alat*“ omogućuje generiranje različitih oblika „*payload-a*“. Pojedine grupe „*payload-a“* odnosno liste nizova znakova je zatim moguće integrirati u sklopu određenih lokacija unutar pojedinog „*HTTP/S zahtjeva*“, s ciljem izvršenja napada na testiranu web aplikaciju.

Svaka pojedina grupa „*payload-a*“ se vezuje uz prethodno označenu poziciju unutar originalnog „*HTTP zahtjeva*“.

U nastavku je opisan skup generatora, integriranih u sklopu inicijalne konfiguracije „*Fuzzer alata*“:

- **Empty/Null generator** podrazumijeva generiranje liste praznih „*string-ova*“ odnosno praznih nizova znakova.
- **File generator** koristi datoteku, definiranu od strane korisnika sustava, u svrhu generiranja odgovarajuće liste „*payload-a*“.
- **File Fuzzers generator** podrazumijeva odabir kombinacije „*fuzzing datoteka*“ registriranih u sklopu „*ZAP okvira*“.
- **Numberzz generator** omogućuje generiranje različitih oblika sekvenci brojeva.
- **Regex generator** omogućuje generiranje liste svih nizova znakova koji zadovoljavaju odgovarajuća ograničenja, definirana korištenjem regularnih izraza.

- **Strings generator** podrazumijeva manualnu ugradnju liste nizova znakova, koji će biti korišteni prilikom izvođenja određenog napada.
- **Script generator** omogućuje odabir skripte koja će biti iskorištena u svrhu generiranja odgovarajućeg skupa „*payload-a*“.
- **Json generator** omogućuje integriranje predefinirane liste (potencijalno neočekivanih) vrijednosti u sklopu svakog parametra, definiranog u sklopu označene „*JSON strukture*“. Pri tome svaka od generiranih „*JSON struktura*“ sadrži jednu od mogućih kombinacija vrijednosti pojedinih parametara, uz poštivanje ograničenja vezanog uz maksimalni broj generiranih „*payload-a*“.

4.2.3.4.2 Payload Processor

„*Payload Processor*“ omogućuje modifikaciju pojedine prethodno definirane grupe „*payload-a*“, prije njihove dostave testiranoj web aplikaciji. Navedena funkcionalnost se najčešće koristi u svrhu kodiranja odnosno dekodiranja sadržaja vezanog uz pojedini „*payload*“.

4.2.3.4.3 Fuzz Location Processor

Ovdje je riječ o funkcionalnosti koja omogućuje modifikaciju svih generiranih grupa „*payload-a*“, prije njihove dostave testiranoj web aplikaciji.

4.2.3.4.4 HTTP Message Processors

Ovdje je riječ o skupu funkcija koje je moguće primijeniti u svrhu analize te modifikacije sadržaja vezanog uz pojedine „*HTTP/S poruke*“, generirane tijekom izvođenja „*Fuzzing alata*“.

Navedeni mehanizmi također omogućuju kontrolu izvođenja samog „*fuzzing procesa*“ te izvršavanje interakcije s korisničkim sučeljem „*OWASP ZAP okvira*“. „*Fuzzer ekstenzija*“ sadrži definiciju skupa „*HTTP Message Processor-a*“.

Dodatne procesore je moguće definirati korištenjem dodatnih ekstenzija. U nastavku je opisan skup procesora, integriranih u sklopu inicijalne konfiguracije „*Fuzzer alata*“:

- **Anti-CSRF Token Refresher** omogućuje regeneraciju „*anti-CSRF tokena*“ u sklopu pojedinog zahtjeva. Ova funkcionalnost podrazumijeva prethodnu registraciju liste mogućih imena „*anti-CSRF tokena*“.
- **Fuzzer HTTP Processorr (Script)** predstavlja okvir za izvršenje odabrane „*Fuzzer HTTP Proccessor skripte*“. Navedena skripta modificira konfiguraciju vezanu uz pojedine dijelove „*fuzzing procesa*“. Tu je primjerice riječ o definiciji novih potencijalnih stanja vezanih uz pojedinu instancu izvođenja napada, promjeni maksimalnog dopuštenog broja pogrešaka vezanih uz pojedini poslani zahtjev i slično.
- **Payload Reflection Detector** omogućuje registriranje situacija kada se „*injektirani payload*“ pojavljuje u sklopu odgovora web poslužitelja. To podrazumijeva pridruživanje vrijednosti „*Reflected*“ unutar „*State stupca*“, vezanog uz pojedini rezultat izvršenja „*fuzzing metode*“
- **Request Content-Length Updater** izvršava modifikaciju vrijednosti „*Content-Length zaglavlja*“, definirane u sklopu inicijalnog „*HTTP/S zahtjeva*“, korištenog prilikom izvođenja „*fuzzer metode*“. Na ovaj način se vrijednost navedenog zaglavlja usklađuje s duljinom tijela svakog generiranog zahtjeva.
- **Tag Creator** omogućuje ugradnju dodatnih oznaka vezanih uz pojedini rezultat izvođenja „*fuzzer metode*“. Navedene oznake se generiraju u ovisnosti o sadržaju zaprimljenih odgovora poslužitelja te se registriraju u sklopu „*State stupca*“.
- **User Message Processor** omogućuje izvođenje „*fuzzer metode*“ iz perspektive jednog od prethodno definiranih „korisnika“ odnosno korištenjem mehanizma za automatsku autentifikaciju i održavanje autentificirane sesije.

4.3 Odnos između OWASP ZAP i Burp Suite okvira

U nastavku su opisane relevantne razlike između navedenih okvira. Važno je napomenuti da priložena usporedba nije sveobuhvatna te se odnosi na stanje navedenih okvira tijekom 2023. godine. Programeri su svjesni nadogradnji prisutnih u sklopu većine relevantnih okvira te općenito tehnoloških trendova. U skladu s tim, nastoje kontinuirano unaprjeđivati programski kod koji se nalaze pod njihovom kontrolom.

Većina funkcionalnosti potrebnih za izvršenje procesa penetracijskog testiranja je implementirana u sklopu obaju navedenih okvira. „*Community verzija Burp suite okvira*“ ipak ne sadrži implementaciju skupa ključnih funkcionalnosti implementiranih u sklopu „*OWASP ZAP okvira*“. Ovdje je prvenstveno riječ o funkcijama vezanim uz automatsku analizu ranjivosti, automatsko mapiranje web mjesta („*Spider alat*“) te mogućnost pohrane podataka, vezanih uz pojedinu sesiju penetracijskog testiranja.

Besplatna verzija „*Burp Intruder alata*“ implicira ograničenu brzinu generiranja odnosno ispostave odgovarajućih „*payload-a*“. Navedena verzija alata također ne sadrži listu prethodno definiranih skupova „*payload-a*“. Ova ograničenja podrazumijevaju duže vrijeme izvođenja „*fuzzing metode*“. „*OWASP ZAP okvir*“ također sadrži alat koji pruža podršku prilikom primjene „*fuzzing tehnike*“. Navedena podrška je implementirana u sklopu „*Fuzzer alata*“. Jedna od prednosti korištenja „*Burp Intruder alata*“, u odnosu na „*Fuzzer alat*“ jest mogućnost jednostavnijeg definiranja tipa napada. Ovdje je riječ o mogućnosti preciznog konfiguriranja načina ugradnje elemenata pojedinih grupa „*payload-a*“ u ovisnosti o grupi definiranih lokacija. Ukoliko želimo iskoristiti sličnu funkcionalnost u sklopu „*ZAP okvira*“, potrebno je isprogramirati odgovarajuću „*Fuzzer HTTP Processor skriptu*“. Također je važno naglasiti da komercijalna verzija „*Burpe Suite okvira*“ omogućuje korištenje kvalitetnijih „*payload-a*“, u usporedbi s onima raspoloživima u sklopu „*OWASP ZAP ekosustava*“.

Funkcionalnost vezana uz mapiranje nevidljivog sadržaja također nije raspoloživa u sklopu besplatne verzije „*Burp Suite okvira*“. „*OWASP ZAP okvir*“ implementira navedenu funkcionalnost u sklopu „*Forced Browse alata*“.

„*Requester alat*“ ne sadrži mogućnost definiranja određenih detalja vezanih uz „*HTTP konekciju*“ odnosno konfiguraciju zajedničke ili pojedinačne konekcije. Navedena

funkcionalnost je implementirana u sklopu „*Burp Repeater alata*“. Ona se prvenstveno koristi prilikom analize ranjivosti koja može rezultirati s uspješnim izvođenjem „*client-side desync napada“*.

Za pojedine funkcionalnosti „*Target alata*“, implementiranog u sklopu „*Burp Suite okvira*“, ne postoji odgovarajući ekvivalent u sklopu „*ZAP okvira*“. Ovdje je riječ o mogućnosti analize prostora napada te o funkcionalnosti vezanoj uz usporedbu sadržaja dvaju mapa stranica. U sklopu besplatne verzije „*Burp Suite okvira*“ moguće je izvršiti usporedbu skupa zahtjeva jedino na osnovu trenutno generirane mape stranica uz korištenje različitih korisničkih podataka (ne postoji mogućnost korištenja mape stranica iz drugog projekta).

„*ZAP okvir*“ implementira dvije značajne funkcionalnosti koje nisu sadržane u sklopu „*Burp Suite okvira*“. Riječ je o okviru za automatizaciju upravljanja s „*OWASP ZAP okvirom*“ te „*The Heads Up Display (HUD) funkcionalnosti*“. „*HUD*“ predstavlja korisničko sučelje okvira, koje se generira u sklopu jednog od kompatibilnih web preglednika. Navedeno korisničko sučelje je implementirano korištenjem „*Vue.js okvira*“ te omogućuje pristup ograničenom skupu funkcionalnosti, vezanih uz „*OWASP ZAP okvir*“.

U nastavku je prikazan skup parova alata koji implementiraju sličnu funkcionalnost iz perspektive „*Burp Suite*“ odnosno „*OWASP ZAP okvira*“. Također su označene one funkcionalnosti, koje nisu raspoložive ili su implementirane uz određena ograničenja upotrebe, u sklopu besplatne verzije „*Burp Suite okvira*“.

Burp Feature	Community	Notes	ZAP Equivalent(s)
Collaborator	✗		OAST Support Add-on
Comparer	✓		Diff
Decoder	✓		Encoder
DOM Invader	✓		Eval Villian Add-on
Extender	✓		Marketplace , Scripts
Intercept	✓		Breakpoints
Intruder	✓	Throttled	Fuzzer
Live scan	✗		ATTACK Mode
Project Files	✗		Session Files
Proxy	✓		Proxy
Repeater	✓		Manual Request Editor , Requestor Add-on
Scanner	✗		Active Scanner
Sequencer	✓		Token Generation and Analysis
Target	✓		Contexts

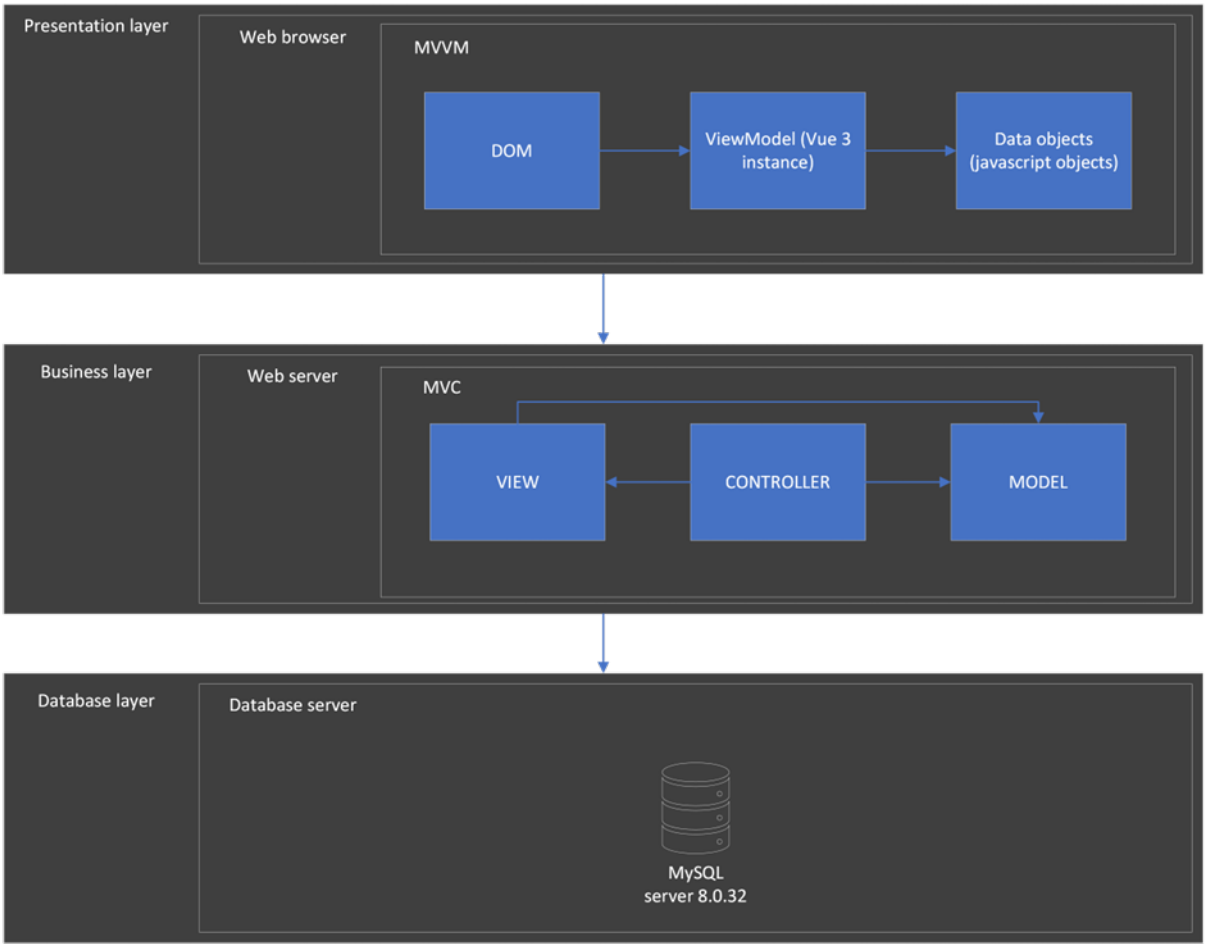
Slika 4.8. Usporedivi alati OWASP ZAP i Burp Suite okvira

5 MODELIRANJE PRIJETNJI NA KONKRETNOM PRIMJERU

U nastavku Specijalističkog rada je demonstriran konkretan proces modeliranja prijetnji na primjeru ranjive web aplikacije. Modeliranje prijetnji se bazira na hipotetskoj ranjivoj web aplikaciji koja implementira funkcionalnost društvene mreže. Cilj je na osnovu navedenog primjera izdvojiti najčešće potencijalne sigurnosne probleme vezane uz dizajn web aplikacije. Prilikom modeliranja se koristi kombinacija odgovarajućih „***OWASP i Microsoft metodologija*“**.

Model ranjive web aplikacije sadržava opis inicijalne arhitekture aplikacije, opis aplikacije iz raznih aspekata te inicijalne zahtjeve web aplikacije (kategorizirane po grupama funkcionalnosti). Proces modeliranja prijetnji web aplikacije podrazumijeva detekciju relevantnih ranjivosti i prijetnji na osnovu modela web aplikacije. Detektirane prijetnje se rangiraju na osnovu procjene rizika pri čemu se koristi odgovarajuća implementacija „*DREAD metode*“. Na osnovu rezultata procesa modeliranja prijetnji izrađuje se revidiran model arhitekture web aplikacije te popis dodatnih sigurnosnih zahtjeva. Izlistane sigurnosne kontrole je moguće iskoristiti u svrhu umanjenja rizika proizašlih iz detektiranog skupa prijetnji i ranjivosti web aplikacije. Navedene kontrole se, u kombinaciji s nepromijenjenim inicijalnim zahtjevima, koriste kao podloga tijekom implementacije unaprjeđene verzije web aplikacije. Detalji vezani uz sigurnu implementaciju unaprjeđene verzije web aplikacije su demonstrirani u sljedećem poglavlju.

5.1 Početna arhitektura web aplikacije



Slika 5.1. Početna arhitektura web aplikacije

5.2 Razine povjerenja

Razine povjerenja se koriste za označavanje skupa prava pristupa dodijeljenih određenom skupu entiteta odnosno korisnika web aplikacije. Ovdje je riječ o ulogama koje se dodjeljuju pojedinim entitetima. Dodjela razine povjerenja podrazumijeva prihvaćanje određene razine ranjivosti sustava u odnosu prema subjektu kojemu su dodijeljene odgovarajuće ovlasti.

Razine povjerenja su dokumentirane u sklopu tablice uz korištenje sljedećih atributa:

1. **ID:** unikatni identifikator vezan uz svaku pojedinu razinu povjerenja. Može se koristiti u svrhu sažetog referenciranja razine povjerenja u sklopu tablica koje dokumentiraju imovinu te ulazne i izlazne točke web aplikacije.
2. **Naziv:** deskriptivni identifikator koji označava razinu povjerenja te ujedno opisuje entitete kojima se dodjeljuje navedena razina povjerenja.
3. **Opis:** detaljniji opis entiteta kojima je dodijeljena promatrana razina povjerenja koji može uključivati opis njihovih ovlasti u sklopu promatranog sustava.

Tablica 5.1. Razine povjerenja

ID	Naziv	Opis
1	Anonimni korisnik	Korisnik kojem je omogućen pristup web aplikaciji ali nije predao odgovarajuće vjerodajnice ili je predao nevaljane vjerodajnice
2	Korisnik s valjanim vjerodajnicama	Korisnik koji se uspio prijaviti u web aplikaciju korištenjem valjanih vjerodajnica
3	Administrator web poslužitelja	Korisnik ovlašten za upravljanje web poslužiteljem odnosno proces samog web poslužitelja
4	Administrator baze podataka	Korisnik ovlašten za upravljanje bazom podataka
5	Read/Write korisnik baze podataka	Korisnički račun ovlašten za čitanje odnosno pisanje u sklopu baze podataka

5.3 Imovina

Imovina podrazumijeva sve resurse odnosno karakteristike web aplikacije koje je potrebno zaštititi na prikladan način. Navedene karakteristike predstavljaju određeni interes iz perspektive napadača odnosno metu tijekom izvođenja napada. Na osnovu izlistane imovine se definiraju odgovarajući sigurnosni ciljevi. To podrazumijeva definiciju odgovarajućih sigurnosnih zahtjeva vezanih uz pojedine elemente imovine. Sve implementirane kontrole direktno ili indirektno doprinose ostvarenju sigurnosnih ciljeva.

Pojedini elementi imovine su prikazani u sklopu tablice uz korištenje sljedećih atributa:

1. **ID:** predstavlja unikatni identifikator pojedinog elementa imovine.
2. **Naziv:** predstavlja deskriptivni identifikator pojedinog elementa imovine
3. **Opis:** dodatne informacije vezane uz pojedini element imovine
4. **Razine povjerenja:** razine povjerenja odnosno razine ovlasti potrebne za pristup odnosno kontrolu nad pojedinim elementom imovine

Tablica 5.2. Elementi imovine

ID	Naziv	Opis	Razine povjerenja
1	Web aplikacija		
1.1	Detalji korisničke prijave	Vjerodajnice koje omogućuju prijavu korisnika u web aplikaciju (korisničko ime i lozinka)	(2) Korisnik s valjanim vjerodajnicama (3) Administrator web poslužitelja (kroz sistemske zapise-ranjivost) (4) Administrator baze podataka

			(5) Read/Write korisnik baze podataka
	Podaci o korisničkoj sesiji	Kolačić koji se koristi u svrhu autentifikacije uzastopnih zahtjeva korisnika (nakon prethodne prijave)	(2) Korisnik s valjanim vjerodajnicama
1.2	Korisnički podaci	Osjetljivi korisnički podaci pohranjeni u sklopu korisničkog profila	(2) Korisnik s valjanim vjerodajnicama (4) Administrator baze podataka (5) Read/Write korisnik baze podataka
1.3	Kreiranje, izmjena ili brisanje podataka korisnika	Izvođenje privilegiranih metoda nad podacima vezanim uz određenog korisnika	(2) Korisnik s valjanim vjerodajnicama (4) Administrator baze podataka (5) Read/Write korisnik baze podataka
	Praćenje ili otpraćanje profila korisnika	Povećanje odnosno umanjenje broja pratitelja određenog profila iz perspektive određenog korisnika	(2) Korisnik s valjanim vjerodajnicama

			(4) Administrator baze podataka (5) Read/Write korisnik baze podataka
1.4	Raspoloživost web aplikacije	Mogućnost kontrole nad raspoloživosti web aplikacije (aktivacija/deaktivacija)	(3) Administrator web poslužitelja
2	Sistemsko okruženje		
2.1	Izvođenje proizvoljnog koda na web poslužitelju	Potencijalno izvođenje malicioznog koda u kontekstu web poslužitelja	(3) Administrator web poslužitelja
2.2	Raspoloživost web poslužitelja	Kontrola nad mogućnosti pristupa web poslužitelju	(3) Administrator web poslužitelja
2.3	Raspoloživost baze podataka	Kontrola nad mogućnosti pristupa bazi podataka	(4) Administrator baze podataka
2.4	Pristup sistemskim i operativnim zapisima	Analiza te potencijalno brisanje sistemskih i operativnih zapisa	(3) Administrator web poslužitelja
2.5	Izvođenje proizvoljnih transakcija baze podataka	Izvršenje proizvoljnih SQL transakcija nad određenom bazom podataka	(4) Administrator baze podataka

			(5) Read/Write korisnik baze podataka
2.6	Direktan pristup bazi podataka	Potpuna kontrola nad određenom bazom podataka	(4) Administrator baze podataka

Na osnovu analize imovine se definiraju odgovarajući **sigurnosni ciljevi:**

1. Zaštita integriteta i raspoloživosti web poslužitelja.
2. Zaštita povjerljivosti, integriteta i raspoloživosti baze podataka.
3. Zaštita povjerljivosti i integriteta sistemskih i operativnih zapisa.
4. Zaštita raspoloživosti web aplikacije.
5. Zaštita integriteta i povjerljivosti korisničkih podataka.

5.4 Ulazne točke

Ulazne točke podrazumijevaju mjesta kroz koja vanjski podaci ulaze u sustav web aplikacije. Ovdje je dakle riječ o sučeljima koja zaprimaju te prosljeđuju korisničke zahtjeve na daljnju obradu. Ulazne točke ujedno predstavljaju mjesta u sklopu kojih napadač može izvršiti potencijalno malicioznu interakciju s napadnutim sustavom te također definiraju granice povjerenja između sustava i njegovog okruženja.

Ulazne točke su dokumentirane u sklopu tablice uz korištenje sljedećih atributa:

1. **ID:** unikatni identifikator ulazne točke
2. **Naziv:** deskriptivni identifikator ulazne točke
3. **Opis:** kontekst referenciranog sučelja

4. **Razina povjerenja:** zahtijevana razina ovlasti potrebna za pristup referenciranom sučelju

Tablica 5.3. Ulazne točke

ID	Naziv	Opis	Razine povjerenja
1	HTTP port	Ulaz u sustav web aplikacije. Svi korisnički zahtjevi prvo prolaze kroz jezgru web aplikacije te se zatim prosljeđuju na daljnju obradu u sklopu pojedinih stranica odnosno metoda web aplikacije	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
1.1	Profile.edit	Izmjena privatnih podataka vezanih uz korisnički profil	(2) Korisnik s valjanim vjerodajnicama
1.2	User-details.edit	Izmjena podataka vezanih uz detalje korisnika koji su javno dohvatljivi	(2) Korisnik s valjanim vjerodajnicama
1.3	Follow/unfollow	Asinkrona dostava zahtjeva za praćenjem odnosno	(2) Korisnik s valjanim vjerodajnicama

		otpraćivanjem određenog profila	
1.4	Prijava korisnika	Sučelje za unos korisničkog imena i lozinke	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
1.5	Registracija korisnika	Sučelje za unos podataka potrebnih za registraciju korisnika	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
1.6	Post.create	Stvaranje novog posta u sklopu „story-a“ trenutnog korisnika	(2) Korisnik s valjanim vjerodajnicama

5.5 Izlazne točke

Izlazne točke podrazumijevaju sučelja web aplikacije u sklopu koji se prezentira rezultat obrade određenog skupa podataka, prethodno zaprimljenih preko ulaznih točaka.

Izlazne točke omogućuju odnosno olakšavaju izvođenje određenih oblika napada. Postojanje sučelja koje prezentira rezultate obrade je nužno prilikom izvođenja određenih oblika napada koji imaju potencijal ugroziti integritet i/ili povjerljivost napadnutog sustava. Ovdje je primjerice riječ o „*Cross-site-scripting*“ ili „*SQL injection napadu*“. Izlazne točke se također mogu koristiti za informirano izvođenje napada grubom silom na korisničku prijavu odnosno kao pomoć prilikom oblikovanja „*SQL injekcije*“. To je pod pretpostavkom da napadnuti

sustav dopušta curenje podataka te prikazuje informativnu obavijest vezanu uz svaki neuspjeli pokušaj prijave odnosno uz svaku pogrešku generiranu u sklopu referencirane baze podataka.

U tablici je prikazan skup izlaznih točaka web aplikacije uz korištenje jednakih atributa kao u sklopu dokumentacije ulaznih točaka web aplikacije.

Tablica 5.4. Izlazne točke

ID	Naziv	Opis	Razine povjerenja
1	Post.show	Prikaži detalje vezane uz pojedinu objavu (post)	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
2	Izlistaj sve objave	Prikaži sve objave registrirane u sustavu u nasumičnom poretku (korištenje „pagination“ sustava)	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
3	Story.show	Prikaži objave i javno dohvatljive detalje vezane uz referenciranog korisnika (uključivo rezultate follow/unfollow operacija)	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama

4	Frontend template	Zajednička komponenta svih frontend komponenti. Podrazumijeva prezentaciju podataka vezanih uz identitet prijavljenog korisnika (dohvat podataka iz baze podataka)	(2) Korisnik s valjanim vjerodajnicama
5	Neuspjela prijava	Prezentiranje obavijesti o neuspjeloj korisničkoj prijavi	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
6	Neuspjela registracija	Prezentiranje obavijesti o neuspjeloj registraciji korisnika	(1)Anonimni korisnik (2) Korisnik s valjanim vjerodajnicama
7	Profile.edit	Prikaži detalje vezane uz privatne podatke korisnika (ovo je ujedno ulazna i izlazna točka)	(2) Korisnik s valjanim vjerodajnicama

8	User-details.edit	Prikaži detalje vezane uz javne podatke korisnika (ovo je ujedno ulazna i izlazna točka)	(2) Korisnik s valjanim vjerodajnicama

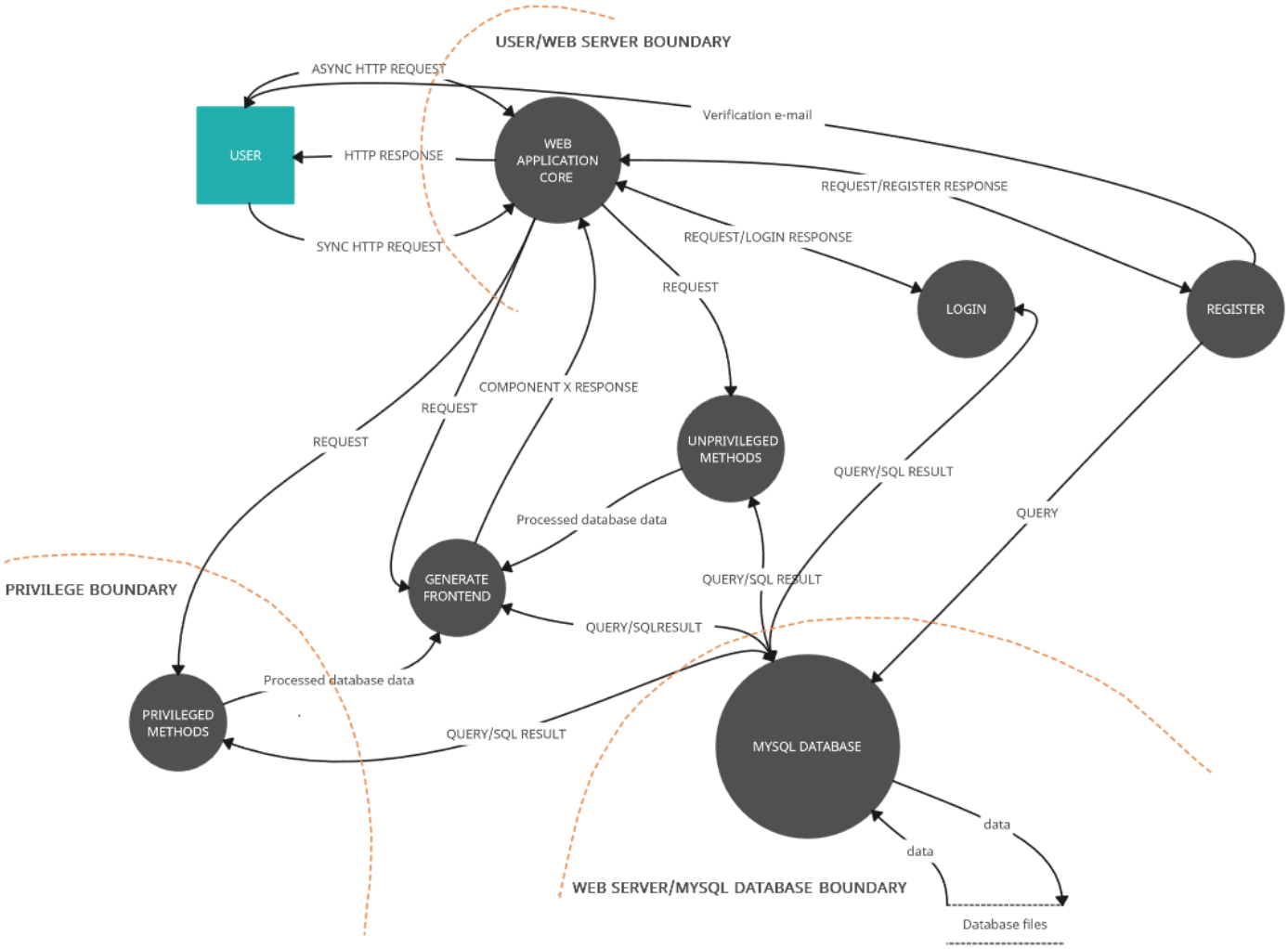
5.6 Dijagram toka podataka web aplikacije

Dijagram toka podataka predstavlja model odnosa između komponenti odnosno procesa sustava pri čemu se također označavaju ključni podatkovni tokovi. Navedeni dijagram je naročito koristan u svrhu označavanja granica povjerenja ("*trust boundaries*"). Prilikom provedbe analize sigurnosti web aplikacije potrebno je obratiti naročitu pozornost na mjesta gdje dolazi do promjene razine povjerenja. Razina povjerenja se dodjeljuje svakom pojedinom subjektu koji pristupa određenoj funkcionalnosti web aplikacije. Ona podrazumijeva spremnost subjekta koji je dodijelio odgovarajuću ovlast da postane u određenoj mjeri ranjiv u odnosu na subjekt kojemu je dodijeljena navedena ovlast odnosno razina povjerenja. Razina povjerenja omogućuje subjektu pristup određenim funkcionalnostima sustava odnosno odgovarajućim objektima. Pojedini dijelovi sustava podrazumijevaju prihvaćanje jednakih razina povjerenja odnosno postoji utvrđeno povjerenje između pojedinih komponenti unutar navedenog dijela sustava. Prema tome se rezultati relevantnih provjera u sklopu prve komponentne prihvaćaju u sklopu svih preostalih komponenti određenog dijela sustava. Ovdje je moguće povući paralelu sa situacijom gdje osoba mora iskoristiti odgovarajući ključ prilikom ulaska u kuću međutim ne mora ga nužno ponovno koristiti prilikom ulaska u svaku pojedinu sobu.

Na pojedinim točkama sustava u sklopu kojih dolazi do promjene razine povjerenja potrebno je provesti odgovarajuću analizu u svrhu održavanja sigurnosti promatranog sustava. Dakle potrebno je provjeriti jesu li na svim pojedinim granicama povjerenja ugrađene odgovarajuće

provjere vezane uz autentifikaciju i autorizaciju korisnika odnosno odgovarajući mehanizmi za provjeru, filtriranje i/ili transformaciju potencijalno malicioznih ulaznih podataka.

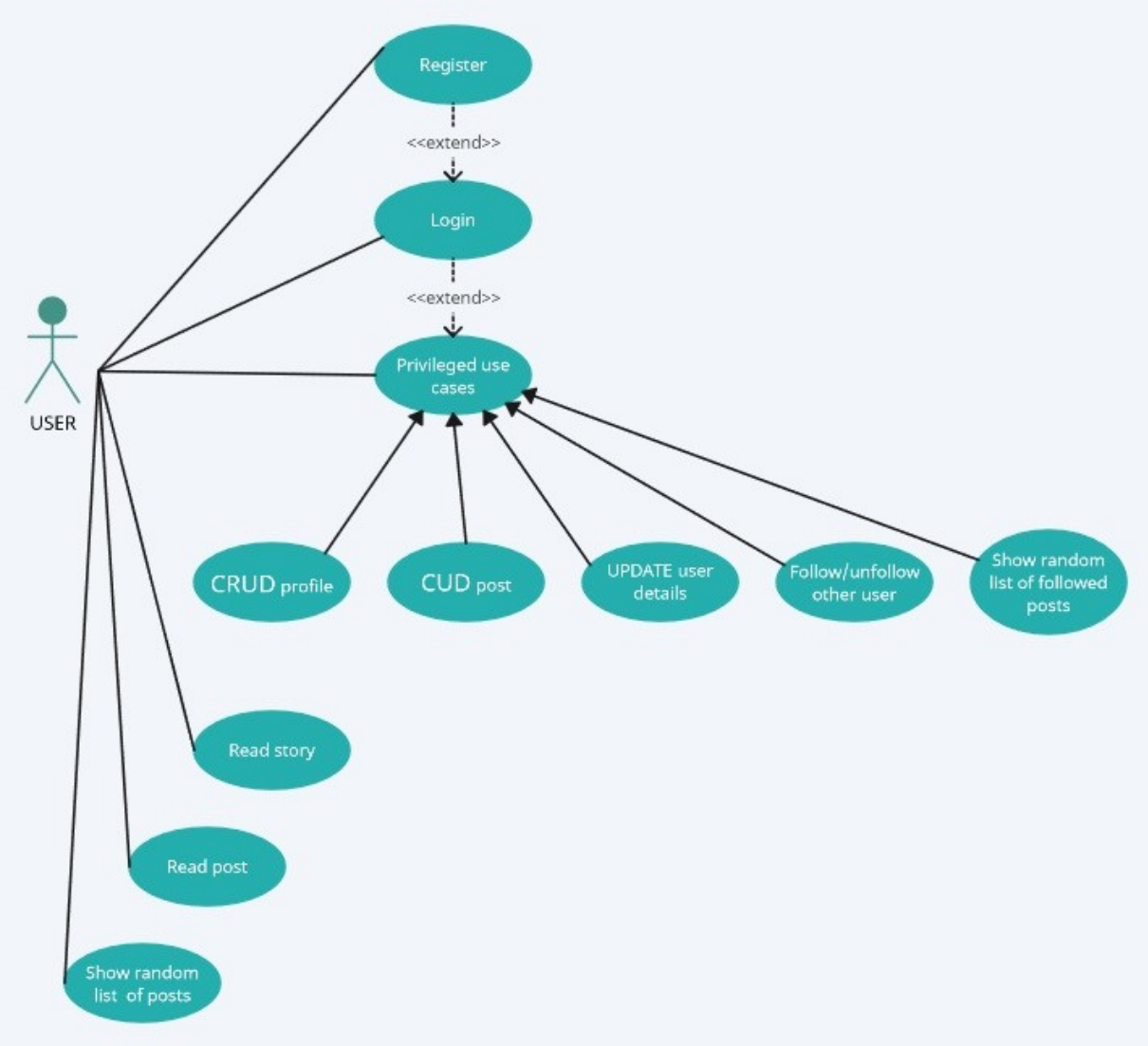
Prilikom analize konkretnog dijagrama toka može se primijetiti mogućnost primjene principa „obrane u dubini“ („*defence in depth*“). U sklopu granice povjerenja između web poslužitelja i baze podataka je moguće ugraditi dodatne provjere vezane uz validaciju podataka vezanih uz korisnički zahtjev. Navedena kontrola primjerice omogućuje dodatnu zaštitu u slučaju upotrebe „*backdoora*“ (zaobilaženje validacije korisničkog zahtjeva) koji mogu biti sadržani u sklopu analizirane web aplikacije. Granica povjerenja između web poslužitelja i baze podataka također podrazumijeva implementaciju odgovarajućih provjera vezanih uz autentifikaciju te autorizaciju iz perspektive web poslužitelja.



Slika 5.2. Dijagram toka podataka web aplikacije

5.7 Use case dijagram

Use case dijagram olakšava shvaćanje cjelokupne funkcionalnosti web aplikacije iz perspektive krajnjeg korisnika. Također olakšava uočavanje prijetnji iz perspektive malicioznog korisnika a time i iz perspektive analitičara sigurnosti web aplikacije.



Slika 5.3. „Use case dijagram“

5.8 Tehnologije

1. PHP 8.2
2. Apache HTTP Server 2.4.55
3. MySql server 8.0.32
4. Dell PowerEdge 2950
5. Javascript i Vue.js 3
6. Tailwind CSS
7. HTTP protokol
8. Laravel 10

5.9 Specifikacija početnih zahtjeva vezanih uz pojedine uloge

Napomene

1. Lanci stranica su grupirani prema zahtjevima prvog reda/korisničkim zahtjevima (grupe stranica implementiraju zahtjeve prvog reda). Stranice su dakle grupirane na osnovu funkcionalnosti koje je potrebno implementirati uz pomoć određenog skupa stranica.
2. Ulazna točka predstavlja osnovnu stranicu grupe stranica vezanih uz određeni zahtjev prvog reda.
3. Detalji na razini zahtjeva sustava i specifikacije programske potpore nisu definirani u cjelini. Prikazana je jedna od iteracija procesa specifikacije zahtjeva.

1. Početna stranica

   ANONIMNI KORISNIK

   1.1. Prikaži objave svih korisnika u nasumičnom poretku.

      1.1.1. „*Pagination sustav*“.

   1.2. Poveznice na ulazne točke grupa funkcionalnosti „Upravljanje registracijom“ i „Upravljanje korisničkom prijavom“.

PRIJAVLJENI KORISNIK

1.1. Prikaži objave svih korisnika u nasumičnom poretku.

1.1.1. „*Pagination sustav*“.

1.2. Poveznica na ulaznu točku grupe „Upravljanje korisničkim objavama“.

2. Upravljanje registracijom

ANONIMNI KORISNIK

2.1. Registracija korisnika.

2.1.1. Korištenje parametara: „*user, username, email, password, confirm password*“.

2.1.2. „*Parametri password i confirm password*“ moraju biti jednaki.

2.1.3. „*Parametri username i email*“ moraju biti unikatni.

2.1.4. Svi parametri su obavezni.

2.2. Automatska izrada (ispunjavanje) početnog korisničkog profila nakon uspješne registracije.

2.3. Funkcija oporavka lozinke.

2.4. Nakon uspješne registracije preusmjeriti korisnika na ulaznu točku grupe „Upravljanje korisničkim objavama“.

2.5. Verifikacija korisničkog emaila.

2.6. Poveznica na ulaznu točku korisničke prijave (iz ulazne točke registracije korisnika).

PRIJAVLJENI KORISNIK

2.1. Ako korisnik nije verificirao email i u slučaju pokušaja korisničkog pristupa ulaznoj točci grupe „Upravljanje korisničkim objavama“, prikaži odgovarajuću poruku (zahtjev za verifikacijom emaila).

2.2. Uređivanje privatnih korisničkih podataka vezanih uz registraciju.

3. Upravljanje korisničkom prijavom

ANONIMNI KORISNIK

3.1.Autentifikacija na osnovu korisničkog imena i lozinke.

3.2.Nakon uspješne prijave preusmjeri korisnika na ulaznu točku grupe „Upravljanje korisničkim objavama“. U slučaju ako je korisnik prethodno pokušao pristupiti stranici koja zahtijeva autentifikaciju, preusmjeri korisnika na tu stranicu.

3.3.„*Remember me funkcionalnost*“.

PRIJAVLJENI KORISNIK

3.1.U slučaju ako je korisnik prijavljen a ponovno pokuša pristupiti stranici koja sadrži formu za autentifikaciju, korisnika se preusmjeruje na ulaznu točku grupe „Upravljanje korisničkim objavama“.

3.2.Održanje sjednice prijavljenog korisnika.

3.3.Odjava korisnika.

4. Upravljanje korisničkim objavama

PRIJAVLJENI KORISNIK

4.1.Prezentacija liste svih korisničkih objava

4.1.1. „*Pagination sustav*“.

4.2.Prezentacija liste korisničkih objava praćenih od strane prijavljenog korisnika.

4.2.1. „*Pagination sustav*“.

4.3.Prezentacija pojedinih korisničkih objava.

4.4.Praćenje/otpraćanje korisnika od strane prijavljenog korisnika bez osvježavanja web preglednika.

4.5.Prezentacija pojedinog korisničkog profila.

4.5.1. „*Pagination sustav*“.

5. Upravljanje objavama prijavljenog korisnika

PRIJAVLJENI KORISNIK

5.1.Prezentacija liste objava prijavljenog korisnika.

5.2.Dodavanje korisničke objave.

5.2.1. Korištenje „*Intervention image paketa*“ za obradu i pohranjivanje korisničkih slika.

5.3.Brisanje korisničke objave.

5.4.Uređivanje korisničke objave.

5.4.1. Korištenje „*Intervention image paketa*“ za obradu i pohranjivanje korisničkih slika.

5.5.Uređivanje dodatnih javno dohvatljivih podataka vezanih uz profil prijavljenog korisnika.

Zajedničke funkcije svih stranica iz „Upravljanje korisničkim objavama“ i „Upravljanje objavama prijavljenog korisnika“ grupa funkcionalnosti:

1.Poveznice na stranice koje implementiraju funkcije „Odjava korisnika“ i „Uređivanje korisničkih podataka vezanih uz registraciju“.

2.Poveznice na ulazne točke „Upravljanje korisničkim objavama“ i „Upravljanje objavama prijavljenog korisnika“ grupa funkcionalnosti.

5.10Prijetnje

Validacija korisničkog unosa

1. „*Cross-site scripting*“ [35]
2. „*SQL injekcija*“
3. Manipuliranje s kolačićima (klijentska strana)
4. „*Manipulacija HTTP zaglavljem*“

[35] https://websitesecuritystore.com/blog/real-world-cross-site-scripting-examples/

Autentifikacija

1. Prisluškivanje mreže
2. Napadi grubom silom
3. „*Dictionary attack*“
4. Krađa vjerodajnica korištenjem sekundarnih kanala

Autorizacija

1. „*Luring attacks*“ (npr. „*CSRF napad*“)
2. „*Elevation of privilege*“ [36](općeniti napadi s ciljem pribavljanja više razine privilegija)

Upravljanje konfiguracijom

1. Korištenje raznih „*backdoor-a*“.

Upravljanje sjednicom

1. „*Session fixation napad*“
2. Predviđanje identifikatora sesije.
3. Prisluškivanje mreže (npr. ako je šifrirana samo razmjena podataka u sklopu autentifikacije)
4. „*Man in the browser napad* [37]“ (podrazumijeva instalaciju zlonamjernog koda koji presreće podatke između klijentskog dijela aplikacije (koja se izvršava u sklopu web preglednika) i određenih sigurnosnih mehanizama)
5. „*Man-in-the-middle napad*“
6. „*Replay napad*“ (uključuje „*session replay napad*“ i napad ponavljanjem cjelokupne poruke)

[36] https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained
[37] https://myappsecurity.blogspot.com/2007/01/ajax-sniffer-prrof-of-concept.html

Upravljanje iznimkama

1. Pribavljanje informacija o aplikacijskim detaljima i općenito o korištenim tehnologijama u sklopu aplikacijskog okruženja.
2. „*Denial of service napadi*“ (primjerice iskorištenje sigurnosnog mehanizma vezanog uz zaključavanja korisničkog računa nakon određenog broja neuspjelih pokušaja prijave)

Revizija i evidentiranje

1. Napadač izvršava napad na aplikaciju bez ostavljanja tragova (brisanje tragova).

5.11 Ranjivosti

Validacija korisničkog unosa[38]

1. Nepotpuna validacija
2. Validacija se izvršava samo na klijentu (bez validacije na poslužitelju).
3. Aplikacija ne vrši filtriranje korisničkog unosa.
4. „*Http-Only atribut*“ nije postavljen u sklopu kolačića.
5. Nije uspostavljen sustav za generiranje i ugradnju „*CSRF tokena*“ u sklopu formi.
6. Aplikacija dopušta učitavanje bilo koje vrste datoteka na mjestu gdje je očekivan neki od slikovnih formata datoteka. Ovdje je riječ o naročito opasnoj instanci nepotpune validacije.
7. „*Directory traversal ranjivost*“ odnosno odsustvo kanonizacije putanje datoteke, definirane na osnovu korisničkog unosa.

[38] „Same-origin policy“ dopušta slanje podataka na bilo koju domenu (izvođenjem malicioznog *„Javascript koda“* u sklopu web preglednika).

Autentifikacija

1. Prijenos nešifriranih vjerodajnica preko mreže uz korištenje „*HTTP protokola*“.
2. Aplikacija ne vodi evidenciju o uzastopnim pokušajima prijave.
3. Odsustvo odgovarajućih mehanizama za zaštitu od „*brute force napada*“ na korisničku prijavu (primjerice „*throttle*“ i „*reCaptcha mehanizmi*“).
4. Nekorištenje „*salt-a*“ u kontekstu sigurne pohrane lozinka unutar baze podataka.
5. Dopušteno je korištenje riječi iz rječnika prilikom definiranja lozinke.
6. Neodgovarajuća kompleksnost lozinke.
7. Istovjetne vjerodajnice se koriste u sklopu različitih sustava
8. Nesigurna pohrana vjerodajnica na raznim mjestima izvan okvira sustava u sklopu kojeg se koriste navedene vjerodajnice
9. Odsustvo mehanizama višefaktorske („*out-of-band*“) autentifikacije (također zaštita od „*MiTB napada*“)

Autorizacija

1. Autorizacijom nisu zaštićeni svi relevantni resursi.

Upravljanje konfiguracijom

1. Prisustvo raznih „ *backdoors*“ u sklopu aplikacijskog sustava.
2. Nesigurna konfiguracija programske podrške
3. Aplikacija ne primjenjuje sigurnosne preporuke vezano uz vraćena „*HTTP zaglavlja*“.

Osjetljivi podaci

1. Nad podacima pohranjenima unutar baze podataka nije primijenjena pretvorba korištenjem odgovarajuće „*hash funkcije*“. Također se ne koristi „*salt*“.
2. Prenošenje osjetljivih podataka preko nesigurnog komunikacijskog kanala.

3. Nepotpuna ili neodgovarajuća primjena kriptografskih mehanizama.

Upravljanje sesijom

1. Prenošenje identifikatora sesije unutar „*URL-a*“ (ranjivost na „*session-fixation napad*“).[39]
2. Sesija se održava aktivnom predugo vremena. U određenim slučajevima se preporuča jednokratno korištenje identifikatora sesije. Također se preporučuje regeneracija identifikatora sesije nakon svake prijave u sustav. Na ovaj način se izbjegava pretvorba fiksirane sesije (poznate napadaču) u autoriziranu sesiju.
3. Odsustvo mehanizma za praćenje istovremenog korištenja identičnih identifikatora sesije od strane različitih korisnika. Odsustvo bilježenja podataka vezanih uz okruženje korisničkog zahtjeva („*IP adresa*“, vrsta web preglednika i slično) te povezivanja navedenih podataka s odgovarajućim identifikatorom sjednice.
4. Odsustvo „*timestamp mehanizma*“ (označavanje pojedinih razmijenjenih poruka u svrhu osiguravanja od „*replay napada*“).
5. Odsustvo primjene kriptografskih mehanizama za očuvanje povjerljivosti i integriteta identifikatora korisničke sjednice.
6. „*Secure zastavica*“ nije postavljena u sklopu kolačića („*MiTM napad*“).
7. Odsustvo „*HSTS mehanizma*“ („*MiTM napad*“)
8. Nedovoljna kompleksnost mehanizma za generiranje identifikatora korisničke sesije.

Upravljanje iznimkama

1. Otkrivanje previše podataka vezanih uz aplikaciju i njezino okruženje u sklopu obavijesti vezanih uz pojavu pojedinih oblika pogrešaka (curenje podataka).

[39] https://security.stackexchange.com/questions/14093/why-is-passing-the-session-id-as-url-parameter-insecure

1. Neadekvatan sustav praćenja neuspješnih pokušaja prijave.
2. Evidencijski sustav nije primjereno osiguran od pokušaja neovlaštene manipulacije.

5.12 Rangiranje prijetnji (DREAD metoda)

Prijetnje se rangiraju na osnovu procjene rizika vezanog uz svaku pojedinu prijetnju. Pritom je moguće koristiti razne metode procjene rizika. Rangiranje prijetnji olakšava određivanje prioritetnih područja za ulaganje sredstava namijenjenih unaprjeđivanju odnosno održavanju sigurnosti web aplikacije.

Procjena rizika, vezanog uz svaku pojedinu prijetnju, se temelji na korištenju „*DREAD metode*“. Ova metoda predstavlja subjektivni model za kvalitativnu procjenu rizika. Model podrazumijeva izračun zbroja vrijednosti dodijeljenih svakom od sljedećih parametara:

1. **Damage:** procijenjena razina štete kao posljedica uspješne realizacije prijetnje.
2. **Reproducibility:** lakoća reprodukcije napada.
3. **Exploitability:** potrebna razina znanja, truda i ustrajnosti vezana uz izvor prijetnje („sposobnost prijetnje“) odnosno razina ranjivosti sustava u odnosu na analiziranu prijetnju.
4. **Affected users:** postotak korisnika sustava koji će osjetiti negativni utjecaj uspješnog izvršenja napada.
5. **Discoverability:** lakoća otkrivanja ranjivosti koje omogućuju uspješno izvršenje napada.

Različite implementacije „*DREAD metode*“ podrazumijevaju korištenje različitih intervala dopuštenih vrijednosti navedenih parametara te različitih pragova koji klasificiraju konačni rezultat primjene navedene metode. U kontekstu provedbe „*DREAD metode*“ donesena je

odluka o korištenju triju mogućih brojčanih vrijednosti za svaki od njezinih parametara (vrijednosti 1,2 ili 3).

Izračun zbroja vrijednosti dodijeljenih navedenim parametrima predstavlja razinu rizika vezanog uz pojedinu analiziranu prijetnju. Viša vrijednost konačnog rezultata podrazumijeva višu razinu procijenjenog rizika. U nastavku su prikazani pragovi koji se koriste u svrhu klasifikacije konačnog rezultata procjene rizika:

Tablica 5.5. Pragovi rizika

Razina rizika	**Rezultat**
Visoka	12 – 15
Srednja	8 -11
Niska	5 – 7

U nastavku je prikazana prioritetna lista relevantnih detektiranih prijetnji temeljena na korištenju navedene implementacije „*DREAD metode*“. Za svaku prijetnju se dodatno navodi skup relevantnih ranjivosti detektiranih u sklopu analizirane web aplikacije.

Tablica 5.6. Prioritetna lista relevantnih detektiranih prijetnji

Prijetnja	Relevantne ranjivosti	D	R	E	A	D	Rezultat
„*Cross-site scripting*“	1. Nepotpuna validacija. 2. Validacija se izvršava samo na klijentu (bez validacije na poslužitelju). 3. Aplikacija ne vrši filtriranje korisničkog unosa. 4. „*Http-Only atribut*“ nije postavljen u sklopu kolačića.	3	3	3	3	3	15
SQL injekcija	1. Nepotpuna validacija.	3	3	3	3	3	15

	2. Validacija se izvršava samo na klijentu (bez validacije na poslužitelju). 3. Aplikacija ne vrši filtriranje korisničkog unosa.						
„*Session fixation napad*“	1. Prenošenje identifikatora sesije unutar „*URL-a*“ 2. Sesija se održava aktivnom predugo vremena (u određenim slučajevima se preporuča jednokratno korištenje identifikatora sesije). Odsustvo mehanizma za praćenje istovremenog korištenja identičnih identifikatora sesije od strane različitih korisnika.	3	2	3	1	3	12
„*Canonicalization napad*“	1. „*Directory traversal ranjivost*“.	2	3	3	2	2	12
„*Man-in-the-middle napad*“	1. „*Secure zastavica*“ nije postavljena u sklopu kolačića. 2. Odsustvo „*HSTS mehanizma*“. Prenošenje osjetljivih podataka preko nesigurnog komunikacijskog kanala.	3	3	3	1	2	12

„*Replay napad*“	1. Sesija se održava aktivnom predugo vremena. 2. Odsustvo mehanizma za praćenje istovremenog korištenja identičnih identifikatora sesije od strane različitih korisnika. Odsustvo „*timestamp mehanizma*“	3	3	3	1	2	12
„*Luring attacks*“	Nije uspostavljen sustav za generiranje i ugradnju „*CSRF tokena*“ u sklopu formi.	3	3	2	1	3	12
Prisluškivanje mreže	1. Prenošenje osjetljivih podataka preko nesigurnog komunikacijskog kanala. Prenošenje identifikatora sesije unutar „*URL-a*“.	3	1	3	1	3	11
Manipulacija „*HTTP zaglavljem*“	Nije detektirana ranjivost. Web aplikacija ne injektira podatke vezane uz „*HTTP zaglavlja*“.	3	3	1	3	1	11
„*Elevation of privilege*“	1. Autorizacijom nisu zaštićeni svi relevantni resursi.	3	3	2	1	2	11
Korištenje raznih „*backdoor-a*“.	2. Nije detektirana ranjivost. Nije pronađen niti jedan relevantan „*backdoor*“. To ne znači da ranjivost nije prisutna.	3	3	1	3	1	11

Napad grubom silom na korisničku prijavu	1. Odsustvo standardnih mehanizama za zaštitu od napada grubom silom na korisničku prijavu (primjerice „*throttle*“ i „*reCaptcha mehanizmi*“). 2. Aplikacija ne vodi evidenciju o uzastopnim pokušajima prijave. 3. Neodgovarajuća kompleksnost lozinke.	3	1	3	1	3	11
Brisanje tragova vezanih uz izvršenje napada	1. Odsustvo primjerene zaštite integriteta sistemskih i operativnih zapisa	2	3	2	3	1	11
Predviđanje identifikatora sesije	1. Odsustvo primjene kriptografskih mehanizama za očuvanje povjerljivosti i integriteta identifikatora korisničke sjednice. 2. Nedovoljna kompleksnost mehanizma za generiranje identifikatora korisničke sesije. 2. Sesija se održava aktivnom predugo vremena.	3	1	2	1	3	10
„*Dictionary attack*“	3. Lozinke nisu zaštićene na odgovarajući način. Prilikom pohrane lozinke	3	1	2	3	1	10

	ne koristi se odgovarajući „*salt*“. 4. Dopušteno je korištenje riječi iz rječnika prilikom definiranja lozinke. 5. Neodgovarajuća kompleksnost lozinke.						
Krađa vjerodajnica korištenjem sekundarnih kanala	1. Istovjetne vjerodajnice se koriste u sklopu različitih sustava. 2. Nesigurna pohrana vjerodajnica na raznim mjestima izvan okvira sustava u sklopu kojeg se koriste navedene vjerodajnice.	3	1	2	1	1	9
Pribavljanje informacija o aplikacijskim detaljima i općenito o korištenim tehnologijama u sklopu aplikacijskog okruženja.	1. Curenje podataka	1	1	2	3	2	9
„*Man in the browser napad*“	1. Odsustvo mehanizama višefaktorske („*Out-of-Band*“) autentifikacije	3	1	1	1	1	7

5.13 Unaprjeđenje sigurnosti analizirane web aplikacije

Na osnovu izvršenog rangiranja prijetnji se donosi odluka o primjeni odgovarajućeg skupa kontrola odnosno sigurnosnih zahtjeva. Donesena je odluka da će prijetnje biti adresirane po njihovom prioritetu uz poštivanje proračunskih ograničenja vezanih uz sigurnost web aplikacija.

1. Sigurnosni zahtjevi vezani uz više grupa funkcionalnosti
   1.1. Preporučuje se korištenje „*Cloud komponenti (Azure)*“ .Cilj je zadržati odgovornost vezanu uz sigurnost aplikacijskog sloja uz delegaciju odgovornosti za sigurnost ostalih slojeva.
   1.2. Korištenje „*Https*“ umjesto „*Http protokola*“.
   1.3. Korištenje ugrađenih sigurnosnih mehanizama „*Laravel okvira*“.
   1.4. Obrana od „*SQL injection napada*“ korištenjem „*Eloquent ORM-a*“.
   1.5. Ispravna konfiguracija za produkciju u sklopu „*env datoteke*“.
   1.6. Obrana od „*Cross site scripting napada*“ korištenjem „*{{ }} notacije*“ u sklopu „*blade datoteka*“ (u pozadini se koristi „*htmlspecialchars PHP funkcija*“)
   1.7. Obrana od „*CSRF napada*“ korištenjem „*CSRF tokena*“.
   1.8. „HTML i PHP validacija forme“.
   1.9. Validacija korisničkog unosa i na razini baze podataka („*Defence in depth*“). Dodatna validacija štiti od iskorištenja eventualnog „*backdoora*“ vezanog uz bazu podataka.
   1.10. Izbjegavanje „*mass assingment*“ ranjivosti korištenjem „*fillable niza*“ u sklopu modela i također korištenjem „*validated() metode*“ nad korisničkim zahtjevom.
   1.11. „*ReCaptcha v3 funkcionalnost*“ (osiguranje formi od automatskog skeniranja i „*DOS napada*“).
   1.12. Enkripcija i zaštita integriteta kolačića.
   1.13. Vraćanje generičkih informacija o greškama u sklopu osjetljivih funkcionalnosti.
   1.14. Konfiguracija log mehanizma („*timezone: Europe/Zagreb*“)

1.15. Izbjegavanje bilježenja osjetljivih podataka poput lozinki u sklopu evidencijskog sustava.

1.16. Kvalitetno upravljanje s iznimkama bez otkrivanja suvišnih/osjetljivih informacija.

1.17. Administracija web aplikacije je omogućena isključivo na lokalnoj razini (izvan javne mreže).

2. Upravljanje registracijom

2.1. Jednaka zahtijevana razina snage lozinke u svim kontrolerima vezanim uz upravljanje s lozinkama.

2.2. Izvršenje „*hash funkcije*“ nad instancama lozinka, prije njihovog spremanja u bazu podataka.

2.3. „*Trim strings middleware*“ se ne primjenjuje nad „*password*“, „*current password*“, „*password_confirmation parametrima*“.

2.4. Oporavak lozinke korištenjem emaila (validacija linka).

3. Upravljanje korisničkom prijavom

3.1. Provjera broja zahtjeva u minuti i privremeno blokiranje pristupa u slučaju prekoračenja maksimalne vrijednosti („*rate limiting/throttle*“). Ova funkcionalnost predstavlja obranu od eventualnih „*DOS napada*“ i automatskog skeniranja web aplikacije.

3.2. Uništavanje sesije nakon odjave, regeneracija tokena (potrebno je promijeniti vrijednost „*session_id*“ i „*CSRF kolačića*“ u svrhu obrane od „*session fixation napada*“).

3.3. Skratiti vremenski interval potreban za deaktivaciju trenutne sesije.

3.4. Definirati autorizacijska pravila za modele korištenjem „*Laravel policy*“.

3.5. Dodjeljivanje minimalnih potrebnih ovlaštenja za svaku pojedinu ulogu.

3.6. „*Trim strings middleware*“ se ne primjenjuje nad „*password parametru*“.

3.7. Minimiziranje emitiranih informacija u slučaju neuspjele prijave.

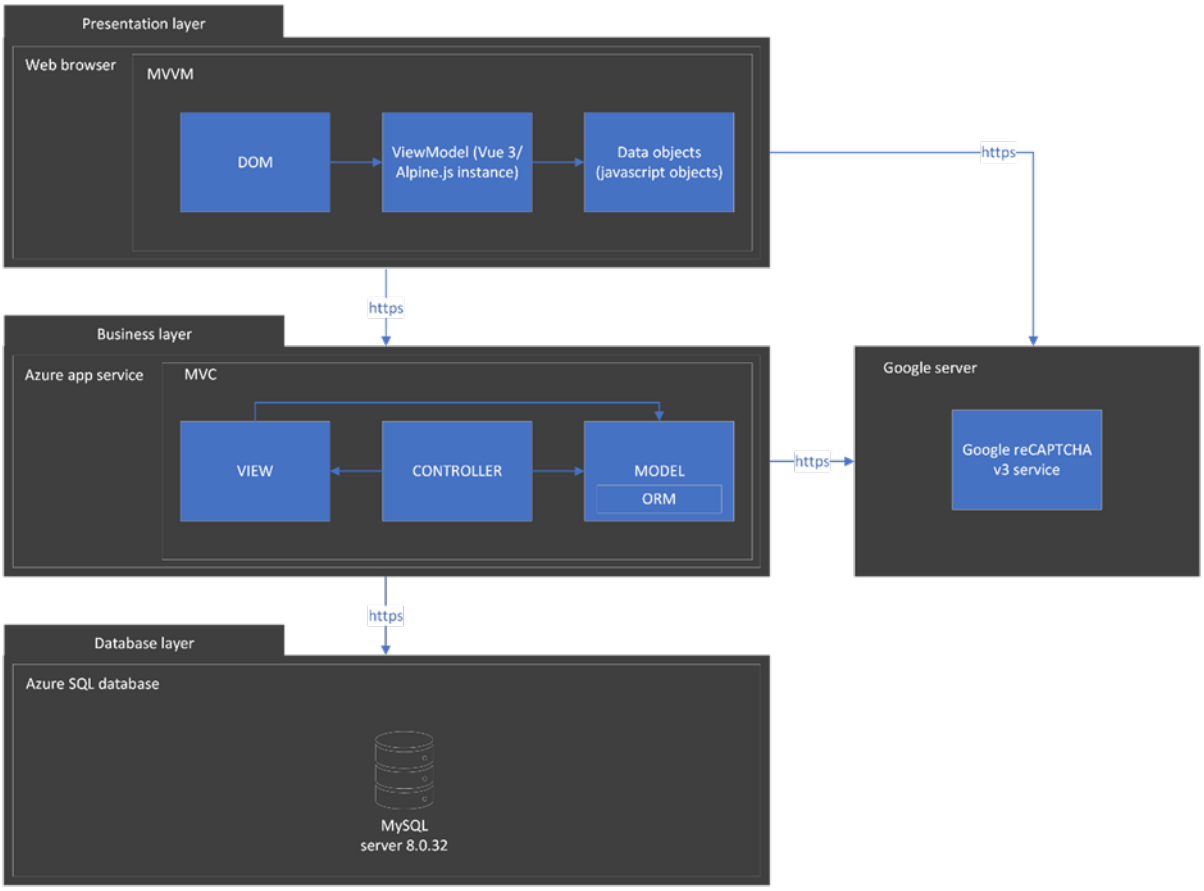
4. Upravljanje objavama prijavljenog korisnika

4.1. Brisanje svih povezanih modela i resursa nakon brisanja „*User modela*“. Brisanje korespondirajuće „*Image datoteke*“ nakon brisanja „*Post modela*“.

4.2. Validacija veličine i ekstenzije pojedinih datoteka koje se planiraju pohraniti na poslužitelju.

4.3. Izbjegavati korištenje „*getClientOriginalName()*“ i „*getClientOriginalExtension()*“ „*Laravel“ metoda*“ (vezano uz obradu korisničkih datoteka). Navedene metode nisu sigurne, s obzirom da klijent kontrolira naziv i ekstenziju datoteka koje učitava na poslužitelj. Preporučuje se koristiti „*hashName()*“ i „*extension() Laravel metode*“.

5.14Konačna arhitektura web aplikacije



Slika 5.4. Konačna arhitektura web aplikacije

6 IMPLEMENTACIJA UNAPRIJEĐENE VERZIJE WEB APLIKACIJE

U sklopu ovog poglavlja su opisani ključni koncepti vezani uz općenitu strukturu web aplikacije iz perspektive korištenih tehnologija. Također su opisani detalji vezani uz implementaciju izdvojenih komponenti web aplikacije. Prilikom analize pojedinih komponenti aplikacije osnovni cilj jest prezentirati i pojasniti pojedine elemente skupa ugrađenih sigurnosnih mehanizama. Pritom se prvenstveno analiziraju tehnologije sadržane u sklopu „*Laravel okvira*“. [40]U svrhu implementacije klijentskog dijela web aplikacije se koristi „*Vue.js 3 okvir*“ i „*Tailwind CSS okvir*“. Navedene tehnologije su pritom na odgovarajući način integrirane u sklopu „*Laravel okvira*“. Prilikom razvoja web aplikacije je također korišten „*Vite sustav*[41]“ i „*MailHog lažni SMTP poslužitelj*[42]“.

Analiza se vrši iz perspektive unaprjeđene verzije web aplikacije. Prilikom razvoja web aplikacije se, uz dokumente vezane uz dizajn sustava, koriste i odgovarajuće smjernice vezane uz sigurnu primjenu „*Laravel okvira*“.

Proces analize unaprjeđene verzije web aplikacije se također može interpretirati kao jednostavan primjer sigurnosnog pregleda koda.

6.1 Izdvojeni arhitektonski koncepti

6.1.1 Dependency Injection

„*Dependency Injection*“ predstavlja arhitektonski obrazac koji implementira „*Inversion Of Control (IOC) princip*“. Ovaj obrazac unaprjeđuje ponovnu iskoristivost, omogućuje jednostavniju refaktorizaciju odnosno održavanje koda te olakšava proces njegovog testiranja. Koncept podrazumijeva postojanje sustava za rezoluciju sučelja. Sustav, na osnovu zahtjeva

[40] Temeljeno na sljedećem online izvoru: https://laravel.com/docs/10.x/releases

[41] „Vite sustav“ predstavlja kombinaciju lokalnog razvojnog poslužitelja i „*bundler alata*“.

[42] Navedeni alat se koristi u svrhu simuliranja korisničkih email računa odnosno „*SMTP poslužitelja*“.

za injekcijom, injektira instance u ovisnosti o konkretnoj implementaciji sučelja. Implementacija „*Dependency Injection mehanizma Laravel okvira*“ temelji se na korištenju „*Service Container-a*“ i odgovarajućih mehanizama za rezoluciju klasa. U sklopu „*Service Container-a*“ se registriraju implementacije sučelja. Ovdje je dakle riječ o sučeljima koja se referenciraju u sklopu zahtjeva za izvršenjem injekcije („*type hint*“).

„*Dependency Injection*“ se razlikuje od arhitektonskog obrasca baziranog na korištenju „*Factory klasa*“. „*Dependency injection mehanizam*“ omogućuje centraliziranu promjenu načina konstrukcije injektirane klase. Ovdje je dakle riječ o mogućnosti centralizirane promjene broja i vrste parametara vezanih uz konstruktor instancirane klase. Na ovaj način se postiže viša razina „*decoupling-a*“ u odnosu na korištenje „*Factory klase*“. Dakle prilikom korištenja „*Dependency Injection-a*“ nije potrebno inicijalizirati korištenu uslugu, odnosno definirati parametre u sklopu njezinog konstruktora. Inicijalizacija korištene usluge se izvršava na jednom mjestu, u sklopu „*Service Container-a*“.

Implementacija „*Dependency injection mehanizma*“, ugrađenog u sklopu „*Laravel okvira*“ se bazira na primjeni „*PHP reflection API-a*“.

Ovdje je važno naglasiti da „*Laravel okvir*“ također omogućuje automatsku rezoluciju pojedinih klasa, bez potrebe za izvršenjem odgovarajućih registracija u sklopu „*Service Container-a*“. Ovdje je riječ o klasama koje ne koriste parametre u sklopu svog konstruktora ili za čije parametre je nakon rekurzivne analize utvrđeno da ovise isključivo o navedenoj vrsti klasa.

6.1.2 Laravel Facade

„*Facade*“ predstavlja „*Static Proxy*“ korišten u sklopu „*Laravel okvira*“. Ovaj koncept predstavlja alternativni pristup korištenju „*Dependency Injection arhitektonskog obrasca*“. „*Facade*“ podrazumijeva korištenje sučelja, u principu nedefiniranih, statičkih metoda na osnovu kojih se, kroz odgovarajući proces rezolucije, referencira odgovarajuća klasa, prethodno registrirana unutar „*Service Container-a*“.

6.1.3 Service provider Laravel okvira

„*Service Provider-i*“ predstavljaju centralno mjesto za registraciju programskih komponenti, koje se koriste u sklopu obrade korisničkog zahtjeva. Tu se prvenstveno misli na registraciju „*Event Listener-a*“, „*Middleware-a*“, ruta i poveznica („*binding*“ u sklopu „*Service Container-a*“). Riječ je dakle o skupu klasa koje se učitavaju prilikom „*bootstrap procesa Laravel okvira*“. Navedene klase je slobodno modificirati u ovisnosti o potrebama projekta. Programski kod vezan uz obradu korisničkih zahtjeva, direktno ili indirektno, referencira komponente registrirane u sklopu „*Service Provider-a*“. On dakle ovisi o okruženju, odnosno stanju sustava, generiranom izvršavanjem „*Service Provider-a*“. Prethodno učitavanje navedenih klasa i izvršavanje odgovarajućih metoda unutar navedenih klasa je ključno za ispravno funkcioniranje sustava. „*Service Provider klasa*“ sadrži dvije osnovne metode, „*register*“ i „*boot metodu*“. „*Register metoda*“ se koristi za registriranje komponenti u sklopu „*Service Container-a*“. „*Boot metoda*“ se izvršava nakon izvršavanja „*register metoda*“ svih registriranih „*Service Provider-a*“. Ova metoda se koristi za dodatne registracije koje mogu ovisiti o izvršenju „*register metoda*“ ostalih „*Service Provider-a*“. Registracija, primjerice „*Event Listenera*“ u sklopu „*register metode*“, može rezultirati s pogreškom. Do pogreške dolazi u slučaju ako se, u sklopu definicije „*Event Listener-a*“, koristi sučelje za koje je tek potrebno registrirati implementaciju (u sklopu „*Service Container-a*“), korištenjem nekog od preostalih neizvršenih „*Service Provider-a*“.

6.1.4 Trait

„*Trait*“ predstavlja kolekciju metoda, koje nije potrebno enkapsulirati odnosno instancirati posredstvom neke klase i koje je moguće integrirati unutar bilo koje klase. „*Trait*“ se može koristiti u svrhu zaobilaženja ograničenja svojstvenih programskim jezicima temeljenim na jednostrukom nasljeđivanju („*PHP*“).

6.1.5 Laravel Middleware

„*Middleware*“ omogućuje filtriranje (obradu) korisničkih zahtjeva i odgovora, generiranih od strane pojedinih komponenti „*Laravel okvira*“. Obrada se izvršava u točkama između pojedinih „*Laravel komponenti*“. „*Middleware*“ omogućuje grupiranje funkcionalnosti, koje bi inače morale biti definirane u sklopu svake pojedine komponente povezane s „*middleware-om*“. Na ovaj način se olakšava održavanje koda obzirom na smanjenje redundancije koda. „*Middleware*“ također može biti povezan samo s jednom komponentom „*Laravel okvira*“. Prema tome, „*middleware*“ povećava i modularnost odnosno mogućnost ponovnog iskorištavanja koda. Navedeno je moguće postići i odgovarajućim korištenjem „*trait-a*“ unutar određene komponente. Razlika je u tome što „*middleware*“ omogućuje koncizniju odnosno pregledniju definiciju pojedine funkcionalnosti (obrada ulaza ili izlaza iz komponente).

Unutar „*Laravel okvira*“ postoje tri kategorije „*middleware-a*“:

1. „*Global middleware*“ (odnosi se na sve korisničke zahtjeve)
2. „*Route Group middleware*“ (odnosi se na obradu ulaza i izlaza iz grupe ruta)
3. „*Route middleware*“ (odnosi se na obradu ulaza i izlaza iz pojedine rute)

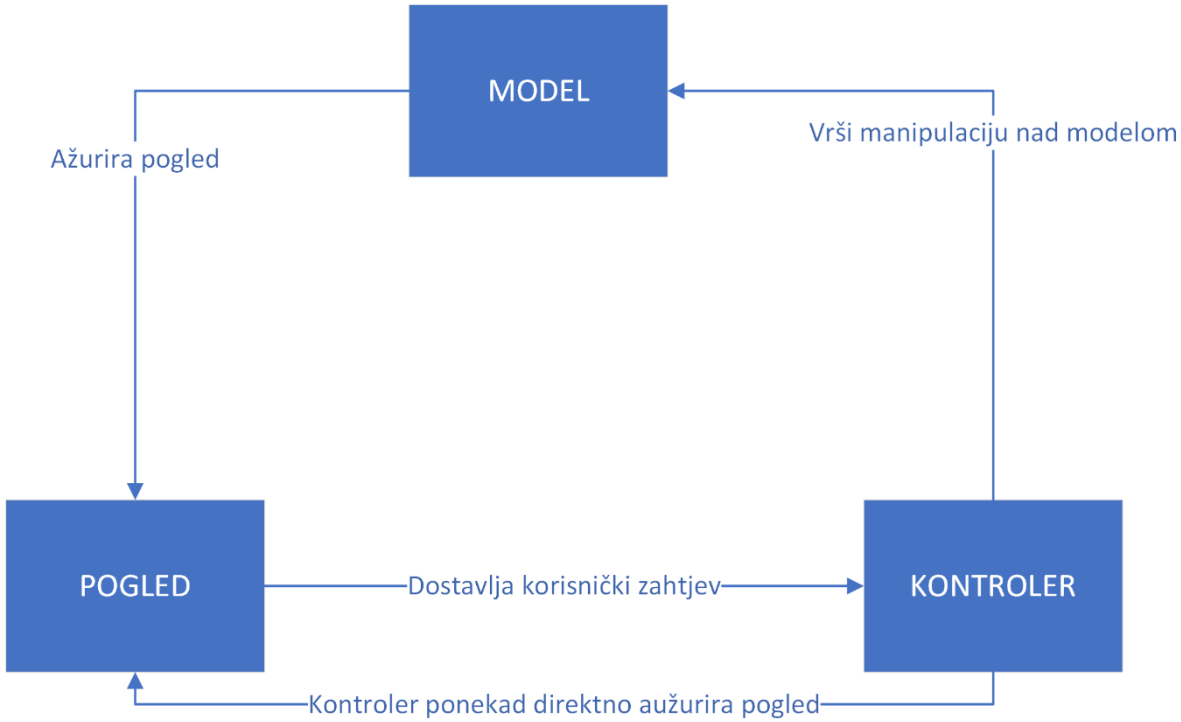
6.1.6 Laravel auth middleware

„*Auth middleware*“ omogućuje autentifikaciju odnosno pozivanje odgovarajućih metoda „*Laravel autentifikacijskog mehanizma*“ za određenu grupu ruta. Izbjegava se ponavljanje koda odnosno ugradnja navedenih poziva metoda unutar svake rute koja izvršava autentifikaciju korisnika. U slučaju ako korisnik nije prethodno autentificiran (ne održava se autentificirana sesija korisnika), „*middleware*“ preusmjerava korisnika na, prethodno imenovanu, „*login rutu*“. Prilikom pridruživanja „*middleware-a*“ određenoj grupi ruta, moguće je specificirati „*Guard komponentu*“ ili niz „*Guard“ komponenti*“. „*Guard komponente*“ predstavljaju osnovu autentifikacijskog mehanizma „*Laravel okvira*“ te su detaljno dokumentirane u nastavku ovog Specijalističkog rada.

6.1.7 MVC

„*MVC (model-view-controller) arhitektonski obrazac*“ podrazumijeva organizaciju koda web aplikacije korištenjem triju povezanih segmenta koda. Ovdje je zapravo riječ o obliku razdvajanja zaduženja („*separation of concerns*“). U nastavku su navedena zaduženja vezana uz pojedine dijelove „*MVC arhitektonskog obrasca*“:

- **Model** sadrži poslovnu logiku te je zadužen za manipulaciju s podacima iz perspektive baze podataka.
- **Pogled** je zadužen za generiranje korisničkog sučelja, na osnovu podataka zaprimljenih od strane modela i kontrolera.
- **Kontroler** predstavlja glavnu poveznicu između preostalih komponenti sustava te inicira odgovarajuće pozive prema pogledu i modelu a na osnovu korisničkog zahtjeva. Kontroler dakle sadrži upravljačku logiku web aplikacije.



Slika 6.1. MVC model (bazirano na sljedećem online izvoru: https://developer.mozilla.org/en-US/docs/Glossary/MVC)

Primjena „*MVC arhitektonskog obrasca*“ omogućuje efikasnu podjelu zaduženja među razvojnim programerima odnosno paralelni razvoj komponenti sustava. Navedeni obrazac također omogućuje ponovnu iskoristivost i bolju organizaciju koda, olakšava održavanje koda te unaprjeđuje njegovu preglednost.

6.2 Životni ciklus korisničkog zahtjeva

U nastavku je opisan okvirni proces obrade korisničkog zahtjeva iz perspektive „*Laravel okvira*“.

Prvi korak životnog ciklusa korisničkog zahtjeva započinje s njegovom dostavom web poslužitelju. Web poslužitelj vrši odgovarajuću obradu te prosljeđuje zahtjev prema „*public/index.php datoteci*“. Navedena „*php datoteka*“ zatim kreira „instancu aplikacije“ odnosno instancu „*Application klase*“, koja predstavlja ekstenziju „Container klase“ odnosno „Service Container-a“. Uz navedeno se također izvršava registracija implementacija vezanih uz pojedina „*Kernel sučelja*“ te se kreira nova instanca navedenog sučelja.

Sljedeći korak podrazumijeva prosljeđivanje korisničkog zahtjeva prema instanci koja proširuje „*ConsoleKernel*“ ili „*HttpKernel klasu*“. U nastavku analize je opisan proces obrade korisničkog zahtjeva koji podrazumijeva korištenje ekstenzije „*HttpKernel klase*“. Korištenjem navedene instance se izvršava „*interna Laravel konfiguracija*“, koja primjerice podrazumijeva detekciju okruženja te konfiguriranje sustava za upravljanje pogreškama i zapisima sustava. Unutar navedene instance se također definira skup korištenih „*middleware-a*“ te se izvršavaju pojedini „*Service provider-i*“ predmetne web aplikacije. Kompletna daljnja obrada korisničkog zahtjeva se zapravo izvršava unutar navedene instance „*Kernel klase*“.

„*App\Providers\RouteServiceProvider klasa*“ predstavlja jedan od najvažnijih „*Service provider-a*“. U sklopu navedene klase se definira „*HOME varijabla*“ odnosno putanja osnovne stranice web aplikacije. Također se vrši registracija datoteka u sklopu kojih se definiraju pojedine rute web aplikacije. Pojedina grupa ruta se pritom povezuje s nekom od prethodno registriranih „*grupa middleware-a*“. Unutar „*routes/web.php datoteke*“ su definirane pojedine „*web rute*“ web aplikacije, odnosno ključne rute vezane uz obradu korisničkih zahtjeva. U sklopu „*routes/api.php datoteke*“ se definiraju eventualne rute vezane

uz „*API predmetne web aplikacije*“. Rute se koriste u svrhu mapiranja korisničkog zahtjeva s odgovarajućim „*callback-om*“ odnosno s registriranom metodom određenog kontrolera.

Nakon registracije svih potrebnih komponenti vrši se pozivanje prethodno definirane instance „*Router klase*“ odnosno rutera. Ruter u pravilu prosljeđuje korisnički zahtjev prema registriranoj metodi kontrolera pojedine rute te također poziva registrirani skup „*middleware-a*“, vezanih uz pojedinu rutu.

Nakon izvršenja obrade iz perspektive metode kontrolera, korisnički zahtjev ponovno prolazi kroz skup „*middleware-a*“, vezanih uz predmetnu rutu. U konačnici „*handle metoda*“ instance „*Kernel klase*“ vraća „*response objekt*“. Sadržaj navedenog objekta se zatim prosljeđuje korisniku web aplikacije u obliku „*HTTP/S odgovora*“.

Važno je naglasiti da se nad svim rutama, vezanim uz izvršenje privilegiranih akcija web aplikacije, primjenjuje „*auth middleware*“. Pri tom se koristi autentifikacijski mehanizam, integriran u sklopu samog „*Laravel okvira*“. Navedeni sustav je detaljno analiziran u nastavku Specijalističkog rada.

6.3 Implementacija modela

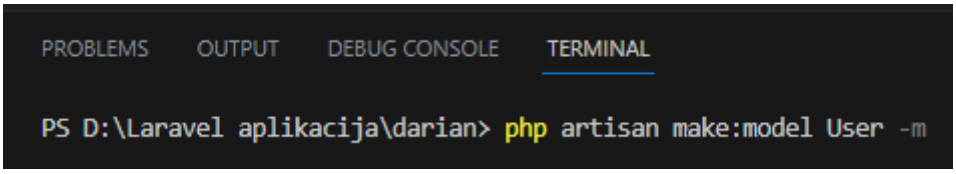
„*Laravel okvir*“ sadrži skup različitih sustava koji olakšavaju upravljanje s bazom podataka. Ovdje je prvenstveno riječ o „*Eloquent ORM sustavu*“ i „*Laravel migration sustavu*“.

„*Eloquent ORM sustav*“ se bazira na korištenju „*Model klase*“, koja sadrži metode za jednostavno oblikovanje „*SQl upita*“ te pohranu odgovarajućih dinamičkih karakteristika. Sveobuhvatan skup ekstenzija ove klase predstavlja model u kontekstu „*MVC arhitekture Laravel okvira*“. Pritom se pojedinačne ekstenzije navedene klase također imenuju kao modeli, vezani uz određenu tablicu trenutno implementirane baze podataka.

Svaki model je dakle vezan uz pojedinu tablicu baze podataka odnosno omogućuje izvođenje „*SQL upita*“ iz perspektive vezane tablice. Prilikom oblikovanja „*SQL upita*“ koriste se podaci pohranjeni u sklopu trenutne instance modela te se rezultati navedenog upita također pohranjuju unutar trenutnog modela. Važno je naglasiti da svaka ekstenzija „*Model klase*“ ima mogućnost pohrane proizvoljnih varijabli, koje ne moraju biti prethodno deklarirane

unutar predmetnog modela. Ovdje je riječ o korištenju mehanizma za pohranu dinamičkih karakteristika. Navedeni sustav se bazira na korištenju „*_get*“ i „*_set php magic metoda*“.

Strukturu novog modela je najjednostavnije generirati korištenjem odgovarajuće „*artisan naredbe*“. Tijekom generiranja modela se također definira početna struktura vezane migracije. U tu svrhu se koristi „*-m opcija*“ navedene naredbe:



Slika 6.2. Generiranje strukture novog modela

Migracije predstavljaju ekstenzije „*Migration klase*“ te omogućuju jednostavno upravljanje strukturom povezane baze podataka te jednostavnu reprodukciju navedene konfiguracije. U sklopu migracija se, uz posredovanje apstraktnog sloja, definiraju odgovarajuće „*SQL naredb*e“. Izvršavanjem navedenih naredbi se stvaraju ili brišu pojedine tablice te se omogućuje modifikacija njihove strukture. Svaka pojedina migracija dakle predstavlja datoteku pohranjenu unutar trenutne web aplikacije te ju je moguće jednostavno reproducirati. Korištenjem migracija se izbjegava nužnost ponavljanja manualne definicije strukture baze podataka te se olakšava suradnja između razvojnih programera[43].

6.3.1 Izdvojene komponente „User modela“

U nastavku je izdvojen skup sigurnosnih mehanizama, primjenjivih u kontekstu oblikovanja „*Model klase*“. Pri tome se kao primjer koristi konkretna implementacija „*User modela*“ web aplikacije. Struktura navedenog modela je bazirana na rezultatu izvršenja prethodno demonstrirane „*artisan naredbe*“.

[43] Alternativa podrazumijeva izvršenje manualne definicije struktura tablica, prilikom svakog pojedinog uvođenja u rad predmetne web aplikacije odnosno prilikom izvođenja aplikacije u sklopu novog razvojnog okruženja.

6.3.1.1 Fillable polje

```
protected $fillable = [
    'name',
    'username',
    'email',
    'password',
];
```

Slika 6.3. „Fillable polje“

„*User model*“ sadrži definiciju „*fillable polja*“. Navedeno polje omogućuje zaštitu od propuštanja neočekivanih podataka prilikom primjene masovne asignacije korisničkih podataka.

Masovnu asignaciju korisničkih podataka je prikladno primijeniti prilikom korištenja modela u svrhu kreiranja novih redaka tablice baze podataka. To podrazumijeva automatsku ugradnju skupa korisnički definiranih atributa unutar predmetnog modela. Važno je primijetiti da je zlonamjerni korisnik u mogućnosti dostaviti proizvoljne argumente „*users tablice*“, unutar posebno oblikovanog korisničkog zahtjeva. S obzirom na navedenu činjenicu potrebno je definirati odgovarajuća ograničenja, vezana uz izvođenje masovne asignacije korisničkih podataka.

„*Fillable polje*“ sadrži skup atributa krajnje referencirane tablice baze podataka, koje je dopušteno definirati u sklopu modela a na osnovu korisničkih podataka.

Također je moguće koristiti „*guarded polje*“, u svrhu definiranja svih atributa modela koje nije dopušteno definirati na osnovu korisničkog zahtjeva.

U kontekstu implementirane web aplikacije, ovdje je zapravo riječ o dodatnom sigurnosnom mehanizmu. U sklopu samih kontrolera je također definirana odgovarajuća provjera, koja se primjenjuje prilikom obrade skupa korisničkih podataka. Navedena provjera se postiže primjenom „*validated() metode*“ nad „*Request klasom*“.

6.3.1.2 Hidden polje

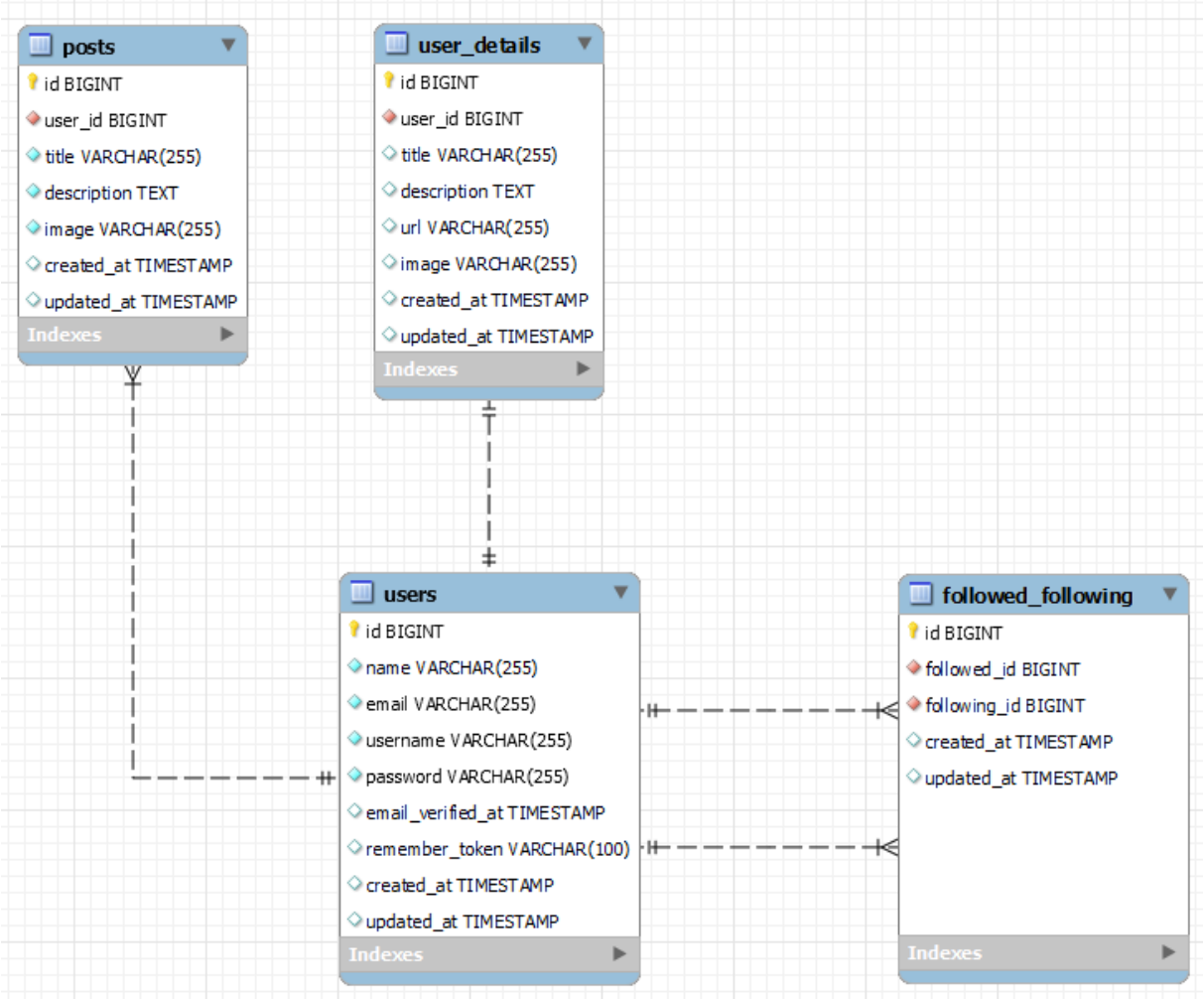
```
protected $hidden = [
    'password',
    'remember_token',
];
```

Slika 6.4. "Hidden polje"

„*Hidden polje*“ se koristi u svrhu registracije parametara koje je potrebno izostaviti izvan „*JSON*“ ili „*array reprezentacije*“ trenutnog modela. Navedeni sigurnosni mehanizam dakle osigurava proces serijalizacije predmetnog modela.

6.3.2 ER dijagram

U nastavku je izložen ER dijagram, vezan uz skup ključnih tablica implementirane web aplikacije.



Slika 6.5. ER dijagram

Definiranjem „dvostruke veze“ na relaciji između „*users*“ i „*followed_following pivot tablice*“ se nastoji prezentirati „*many-to-many relacija*“ u kontekstu pojedinih redaka „*users tablice*“. Ovdje je zapravo riječ o rekurzivnoj relaciji „*users tablice*“. Pojedini redak „*followed_following pivot tablice*“ se praktički vezuje uz dva različita retka „*users tablice*“.

Korisničke sesije se ne pohranjuju u sklopu baze podataka. Umjesto baze podataka se koristi skup odgovarajućih datoteka, pohranjenih u sklopu „*storage/framework/sessions direktorija*“.

6.4 Implementacija temeljnih kontrolera web aplikacije

U nastavku su opisani ključni kontroleri web aplikacije. Pritom se analiziraju pojedini elementi „*Laravel okvira*“, s posebnim fokusom na skup primijenjenih sigurnosnih mehanizama.

6.4.1 Post kontroler („PostController klasa“)

„*PostController*“ sadrži logiku vezanu uz upravljanje korisničkim objavama. Ovdje je riječ o ključnoj funkcionalnosti web aplikacije iz perspektive korisnika. U nastavku su izdvojene pojedine metode navedenog kontrolera. Pritom je opisana njihova osnovna funkcionalnost te eventualno ugrađeni sigurnosni mehanizmi.

6.4.1.1 Create metoda

```
public function create()
    {
        return view('post.create');
    }
```

Slika 6.6. "Post kontroler create metoda"

„*Create metoda*“ predstavlja najjednostavniji mogući oblik metode kontrolera. Njezina jedina svrha jest pozivanje „*post.create pogleda*“, bez prosljeđivanja atributa. Navedeni pogled generira formu koja omogućuje definiciju nove korisničke objave te dostavu navedenih podataka prema „*store metodi*“ sadržanoj u sklopu „*Post kontrolera*“.

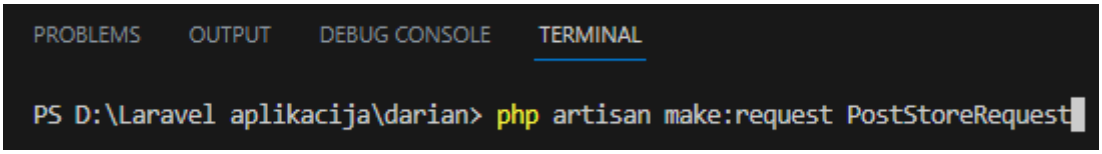
6.4.1.2 Store metoda

```
   public function store(PostStoreRequest $request)
    {
        $imagePath = "/storage/".$request->file('image')->store('uploads',
'public');

        $image = Image::make(public_path("{$imagePath}"))->fit(1200,1200);
        $image->save();

        $request->user()->posts()->create([
            'title' => $request->input('title'),
            'description' => $request->input('description'),
            'image' => $imagePath,
        ]);

        return redirect('/story/' . $request->user()->id);
    }
```

Slika 6.7. "Post kontroler store metoda"

Za početak možemo primijetiti da trenutna metoda podrazumijeva korištenje „*dependency injection mehanizma*“. Ovdje je konkretno riječ o injekciji „*PostStoreRequest klase*“ korištenjem automatske rezolucije. Navedena klasa predstavlja ekstenziju „*FormRequest klase*“, koja pak predstavlja proširenje „*Request klase*“. „*Request klasa*“ sadrži skup osnovnih metoda koje omogućuju obradu i pristup pojedinim elementima korisničkog zahtjeva, bez potrebe za direktnim referenciranjem „*superglobal PHP varijabli*“.

Proširenjem „*FormRequest klase*“, korisnik je u mogućnosti definirati pravila vezana uz validaciju i autorizaciju korisničkog zahtjeva. Navedena pravila se pritom automatski izvršavaju prilikom injektiranja odnosno generiranja instance navedene klase. Korištenjem ovog pristupa se ostvaruje bolja preglednost koda definiranog u sklopu pojedine metode kontrolera odnosno omogućuje se izdvajanje dijelova koda, koji nisu direktno vezani uz osnovnu logiku web aplikacije. Strukturu klase, koja se temelji na proširenju „*FormRequest klase*“, najlakše je definirati korištenjem odgovarajuće „*Artisan CLI naredbe*“:



Slika 6.8. Generiranje strukture ekstenzije "FormRequest klase"

Na osnovu navedene strukture se oblikuje konačna definicija „*PostStoreRequest klase*“:

```
class PostStoreRequest extends FormRequest
{
    public function rules()
    {
        return[
            'title' => ['string', 'max:255','required'],
            'description' => ['string', 'max:500','required'],
            'image' => ['required', File::image()->max(12 * 1024)],
        ];
    }
}
```

Slika 6.9. "PostStoreRequest klasa"

U sklopu „*rules metode*“ se definiraju pravila vezana uz validaciju korisničkog zahtjeva. U konkretnom slučaju se definiraju ograničenja vezana uz tip podataka i maksimalnu veličinu pojedinih parametara te se definira zahtjev za dostavom svih navedenih parametara korisničkog zahtjeva. Na sličan način je moguće definirati „*authorize metodu*“ te time omogućiti autorizaciju korisnika u sklopu „*PostStoreRequest klase*“. U sklopu analizirane web aplikacije, mehanizam autorizacije korisničkog zahtjeva je ipak implementiran korištenjem drugačijeg pristupa a koji će biti opisan u nastavku analize.

Neuspjela validacija korisničkog zahtjeva rezultira s preusmjeravanjem korisnika na prethodno posjećenu lokaciju. Podaci vezani uz relevantne pogreške se pritom pohranjuju u sklopu trenutne sesije, te se brišu nakon sljedećeg korisničkog zahtjeva („*flash data*“). U slučaju uspješne validacije, nastavlja se s izvođenjem koda, definiranog u sklopu analizirane metode.

U nastavku izvršenja metode kontrolera, dostavljena slika korisničke objave se preuzima iz datoteke za privremenu pohranu te se pohranjuje unutar odgovarajućeg direktorija.

„*File metoda Request klase*“ vraća instancu „*UploadedFile klase*“. Nad navedenom klasom je definiran skup metoda koje je moguće koristiti u svrhu interakcije s datotekom dostavljenom u sklopu korisničkog zahtjeva. Konkretan poziv „*store metode*“, implementirane nad „*UploadedFile objektom*“, omogućuje pohranu datoteke u sklopu “*app/storage/public/uploads direktorija*“. Navedena metoda vraća sufiks putanje koja je iskorištena prilikom pohrane slike korisničke objave: „*./public/uploads/**naziv datoteke***“. Ovoj putanji se zatim dodaje prefiks „*./storage/*“. Na ovaj način je definiran sufiks konačne putanje koja omogućuje pristup pohranjenoj slici iz perspektive krajnjeg korisnika.

„*Laravel okvir*“, temeljeno na izvornim postavkama, omogućuje eksterni pristup isključivo „*public direktoriju*“. Navedeni sigurnosni mehanizam se koristi u svrhu obrane od neovlaštenog direktnog pristupa pojedinim datotekama sustava. S druge strane, pojedine komponente „*Laravel okvira*“ u pravilu pohranjuju resurse unutar „*./app/storage/ direktorija*“. Ako želimo omogućiti direktan pristup „*app/storage direktoriju*“, potrebno je definirati „*symlink*“ na relaciji između „*public direktorija*“ i prethodno navedene mape. Dakle prethodno demonstriran sufiks putanje korisničke objave je oblikovan na način da omogući ostvarenje indirektnog pristupa resursu, korištenjem simboličnog linka. Gledajući iz perspektive konkretne specifikacije web aplikacije, omogućavanje direktnog pristupa slikama korisničke objave je prihvatljivo s obzirom da je riječ o javno dohvatljivim resursima. Eventualna enumeracija navedenih resursa ne predstavlja direktni sigurnosni rizik. Navedena praksa ipak otvara prostor za uvođenje ranjivosti prilikom eventualne nadogradnje web aplikacije.

Originalno pohranjena slika korisničke objave se zatim modificira korištenjem „*Intervention Image paketa*“ odnosno pohranjuje se slika veličine 1200x1200 piksela.

Nakon toga slijedi oblikovanje odgovarajućeg upita prema bazi podataka, korištenjem „*Eloquent ORM sustava*“. Navedeni upit omogućuje pohranu pojedinih parametara korisničkog zahtjeva u sklopu „*posts tablice*“. Sufiks putanje u sklopu koje je pohranjena slika korisničke objave, se pritom koristi kao vrijednost „*image atributa*“ navedene tablice.

S obzirom da se oblikovanje prethodno navedenog „*SQL upita*“ temelji na referenciranju „*users tablice*“ trenutno prijavljenog korisnika, nije potrebna ugradnja dodatnih mehanizama za autorizaciju korisničkog zahtjeva. Svi novostvoreni reci „*post tablice*“ mogu bit povezani jedino s identifikacijskim brojem trenutnog korisnika, definiranim u sklopu „*users tablice*“. Autorizacija je dakle implicitno izvršena na osnovu referenciranja trenutno prijavljenog korisnika prilikom oblikovanja odgovarajućeg „*SQL upita*“.

Metoda završava preusmjeravanjem korisnika na njegovu „story stranicu“.

6.4.1.3 Show metoda

```
    public function show(Post $post, Request $request)
    {
        $follows = ($request->user()) ? $request->user()->following-
>contains($post->user->id) : false;
        return view('post.show', compact('post', 'follows'));
    }
```

Slika 6.10. "Post kontroler show metoda"

Show metoda koristi „*Route Model Binding mehanizam*“ s ciljem automatskog generiranja instance odnosno populacije „*Post modela*“ i to na osnovu „*route parametra*“, definiranog u sklopu korisničkog zahtjeva. Ovaj mehanizam zahtjeva odgovarajuće imenovanje pojedinih „*route parametra*“. Inicijalna konfiguracija „*Laravel okvira*“ podrazumijeva istovjetnost naziva navedenih parametra s nazivima varijabli koje se koristi u svrhu referenciranja predmetnog modela.

Show metoda prema tome implicitno generira „*Post model*“, koji korespondira identifikatoru definiranom u sklopu rute, te također zaprima injektiranu instancu „*Request klase*“. Korištenjem navedenih instanci se provjerava prati li trenutni korisnik objave korisničkog profila, koji je vezan uz trenutno referenciranu korisničku objavu. Rezultat navedene provjere se pohranjuje u sklopu „*$follows varijable*“. Navedena varijabla se zatim u kombinaciji s generiranim „*Post modelom*“ prosljeđuje „*post.show pogledu*“, unutar kojeg se prezentira sadržaj ugrađenog modela.

6.4.1.4 ListAll metoda

```
public function listAll()
{
    $posts = Post::inRandomOrder()->with('user')->paginate(6);

    return view('post.list-all', compact('posts'));
}
```

Slika 6.11. "Post kontroler listAll metoda"

Ova metoda omogućuje generiranje skupa modela svih korisničkih objava te prosljeđivanje navedenog skupa podataka, pohranjenog u sklopu instance „*Collection klase*“, prema „*post.list-all pogledu*“. Navedeni modeli se generiraju u nasumičnom poretku te također sadrže definiciju vezanog „*User modela*“. „*User model*“ odnosno redak „*users tablice*“, koji je vezan uz pojedinu korisničku objavu, se dakle također ugrađuje u sklopu svakog pojedinog „*Post modela*“.

Ovdje je riječ o primjeni „*eager loading koncepta*“ odnosno proaktivnom izvršenju „*SQl upita*“, koji se koristi prilikom definicije pojedinih „*User modela*“. Navedeno rješenje je primijenjeno s obzirom na činjenicu da pogled, kojem se prosljeđuju obrađeni podaci, podrazumijeva referenciranje „*User modela*“ iz perspektive korištenja svake pojedine instance „*Post modela*“. Proaktivnim izvršenjem dvaju „*SQL upita*“ se u značajnoj mjeri smanjuje konačan broj izvršenih „*SQl upita*“. Dakle umjesto „N+1 upita“, gdje je „N“ broj preuzetih „*Post modela*“, izvršavaju se samo dva „*SQL upita*“. U svrhu implementacije ovog rješenja primjenjuje se „*with() metoda Eloquent ORM sustava*“. U konkretnom primjeru navedena metoda omogućuje ugradnju vezanog „*User modela*“ u sklopu svake generirane instance „*Post modela*“. „*User model*“ se ugrađuje kao dodatni atribut odnosno dodatna dinamička karakteristika („*dynamic property*“) „*Post modela*“.

U sklopu ove metode se također primjenjuje mogućnost automatskog generiranja „*pagination sustava“,* svojstvena „*Laravel okviru*“. „*Pagination sustav*“ omogućuje prikaz rezultata korisničkog zahtjeva unutar skupa povezanih stranica. Na ovaj način se ubrzava učitavanje pojedine stranice te se ostvaruje bolja preglednost u kontekstu korisničkog sučelja.

Primjena navedenog sustava, iz perspektive razvojnog programera, se temelji na korištenju dvaju komponenti. Ovdje je riječ o „*paginate() naredbi*“, definiranoj u kontekstu instance „*query builder-a*“, te „*links() metodi*“ koja se primjenjuje prilikom oblikovanja klijentske strane web aplikacije. „*Paginate() metoda*“ vraća dio rezultata vezanih uz „*SQL upit*“, oblikovan korištenjem „*Eloquent ORM sustava*“. Razvojni programer pritom definira veličinu pojedinog podskupa rezultata predmetnog upita. Vrijednost „***page** query varijable*“ se koristi u svrhu određivanja odmaka od početnog elementa skupa rezultata. „*Paginate() metoda*“ također dinamički ugrađuje generiranu „*link() metodu*“ u sklopu „*Eloquent modela*“, nad kojim se izvršava trenutni „*SQL upit*“. „*Links() metoda*“ omogućuje automatsko generiranje klijentskog dijela „pagination sustava“, vezanog uz određeni model odnosno metodu kontrolera. Ova metoda se automatski oblikuje u ovisnosti o svakom pojedinom slučaju korištenja „*paginate() metode*“. „*Links() metoda*“ dakle omogućuje definiranje strukture i dizajna skupa navigacijskih linkova. Navedeni linkovi vode prema predmetnoj metodi kontrolera te sadrže odgovarajuće vrijednosti „***page** query parametra*“.

6.4.1.5 ListFollowed metoda

```
    public function listFollowed(Request $request)
    {
        $users = $request->user()->following()->pluck('following_id');
        $posts = Post::whereIn('user_id', $users)->with('user')-
>inRandomOrder()->paginate(6);

        return view('post.list-followed', compact('posts'));
    }
```

Slika 6.12. "Post kontroler listFollowed metoda"

„*ListFollowed metoda*“ podrazumijeva generiranje skupa modela svih korisničkih objava vezanih uz korisnički profil, koji je praćen od strane trenutnog korisnika. U tu svrhu se prvo izdvaja skup identifikatora praćenih korisničkih profila[44], vezanog uz trenutnog korisnika. Pritom se koristi „*pluck() metoda*“, definirana nad „*Laravel Collection klasom*“. Nakon toga

[44] Navedeni rezultati se spremaju u sklopu „*Collection klase*“ te se zatim obrađuju korištenjem „*pluck metode*“.

slijedi oblikovanje „*SQl upita*“, koji rezultira s generiranjem skupa modela svih korisničkih objava vezanih uz prethodno definiranu listu identifikatora pojedinih korisnika. U sklopu svakog generiranog modela korisničke objave se ugrađuje model vezanog korisnika. Pojedini modeli korisničkih objava se ugrađuju u nasumičnom poretku unutar „*Collection klase*“, te se također ponovno koristi „*paginate() metoda*“, u svrhu automatskog generiranja „*pagination sustava*“.

S obzirom da je krajnji cilj metode generiranje slučajnog niza korisničkih objava, pristupa se oblikovanju „*SQL upita*“ iz perspektive „*Post modela*“. Zbog toga se u konkretnom primjeru koristi „*whereIn naredba*“ nad „*Post modelom*“. Alternativni i manje efikasni pristup podrazumijeva preuzimanje skupa „*User modela*“ u nasumičnom poretku te zatim generiranje nasumične liste svih korisničkih objava vezanih uz pojedinog korisnika, korištenjem „*post relacijske metode*“.

6.4.1.6 Delete metoda

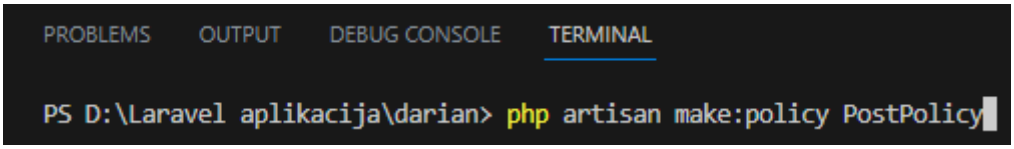
```
public function delete(Post $post)
{
    $this->authorize('delete', $post);

    File::delete(public_path($post->image));
    Post::where('id', $post->id)->delete();


    return redirect('/story/'.$post->user->id);
}
```

Slika 6.13. "Post kontroler delete metoda"

U sklopu ove metode se susrećemo sa standardnim načinom autorizacije korisničkog zahtjeva, gledano iz perspektive „*Laravel okvira*“. Navedeni pristup podrazumijeva korištenje „*Policy klasa*“ te referenciranje njihovih pojedinih metoda u sklopu konstruktora kontrolera ili iz perspektive pojedine metode kontrolera.

„*Policy klase*“ omogućuju organizaciju autorizacijske logike nad određenim modelom. To podrazumijeva definiciju pravila pristupa vezanih uz izvršenje svake pojedine akcije nad predmetnim modelom.

Strukturu nove „*Policy klase*“, vezane uz „*Post model*“, je najjednostavnije definirati korištenjem sljedeće naredbe:



Slika 6.14. Generiranje strukture ekstenzije "Policy klase"

Strukture novih „*Policy klasa*“ se, korištenjem navedene naredbe, automatski pohranjuju u sklopu „*app/Policies direktorija*[45]“.

U nastavku je prezentirana konačna definicija „*PostPolicy klase*“:

```
class PostPolicy
{
    use HandlesAuthorization;

    public function update(User $user, Post $post)
    {
        return $user->id == $post->user_id;

    public function delete(User $user, Post $post)
    {
        return $user->id == $post->user_id;
    }
}
```

Slika 6.15. "PostPolicy klasa"

Dakle svaka prikazana metoda „*PostPolicy klase*“ omogućuje autorizaciju izvršenja određene akcije nad „*Post modelom*“. Metode „*PostPolicy klase*“ sadrže dva argumenta. Ovdje je riječ

[45] U slučaju ako navedeni direktorij nije definiran u kontekstu trenutne aplikacije, on se također generira tijekom izvođenja navedene *„artisan naredbe“*.

o modelu trenutnog korisnika te modelu korisničke objave, kojem navedeni korisnik pokušava pristupiti. Rezultat izvršenja pojedine metode može predstavljati „*true*“ ili „*false vrijednost*“. Navedeni rezultati se koriste u sklopu raznih „autorizacijskih metoda“, koje se pozivaju iz perspektive konstruktora kontrolera ili pojedine metode kontrolera.

Prije korištenja navedenih metoda potrebno je izvršiti registraciju „*Policy klase*“. Važno je naglasiti da „*Laravel okvir*“ omogućuje implicitno izvršenje registracije odnosno automatsko povezivanje navedene klase s prikladno imenovanim modelom. Pritom se koristi konvencija koja podrazumijeva istovjetnost naziva početnog dijela naziva „*Policy klase*“ i naziva njezinog referenciranog modela. Uz navedeno, naziv „*Policy klase*“ također mora sadržavati sufiks „*Policy*“.

Ovu poveznicu je naravno moguće definirati i eksplicitno, u slučaju ako ne želimo koristiti prethodno navedenu konvenciju. U nastavku je izložen primjer registracije „*Policy klasa*“ u sklopu odgovarajuće „*Provider klase*“. Pritom su demonstrirana dva načina registracije i to iz perspektive „*Post*“ i „*UserDetails modela*“.

```
class AuthServiceProvider extends ServiceProvider
{
    protected $policies = [
        Post::class => PostPolicy::class,
        'App\Models\UserDetails' => 'App\Policies\UserDetailsPolicy',
    ];

    public function boot()
    {
        $this->registerPolicies();
    }
}
```

Slika 6.16. Registracija "Policy klasa"

„*Delete metoda Post kontrolera*“, prilikom autorizacije korisničkog zahtjeva, referencira „*delete metodu PostPolicy klase*“. Pritom se koristi „*authorize metoda*“. Navedena metoda se poziva nad instancom „*PostController klase*“ te u slučaju neuspjele autorizacije automatski generira „*HTTP/S odgovor*“ s „403 statusnim kodom“. U slučaju ako je referencirana „*delete*

metoda PostPolicy klase" vratila „*true vrijednost*", nastavlja se s izvođenjem koda „*delete metode predmetnog kontrolera"*.

Ključna funkcionalnost navedene metode podrazumijeva brisanje odgovarajućeg retka unutar „*posts tablice*", pohranjene u sklopu baze podataka. U tu svrhu je moguće koristiti „*delete metodu"* koja je definirana nad općenitim „*Eloquent modelom*". Prije brisanja same korisničke objave vrši se brisanje lokalno pohranjene datoteke odnosno slike vezane uz navedenu korisničku objavu.

Metoda u konačnici preusmjerava korisnika prema „story stranici", vezanoj uz njegov korisnički profil.

6.4.1.7 Edit metoda

```
public function edit(Post $post){

    $this->authorize('update', $post);

    return view('post.edit', compact('post'));
}
```

Slika 6.17. "Post kontroler edit metoda"

Ova metoda se koristi u svrhu generiranja „*post.edit pogleda*" odnosno korisničkog sučelja u sklopu kojeg je moguće modificirati podatke vezane uz trenutno referenciranu korisničku objavu. Pritom se navedenom pogledu prosljeđuje predmetni „*Post model*"[46] .

Autorizacija korisnika, u sklopu same „*edit metode*", se izvršava u svrhu osiguranja kvalitetnog korisničkog iskustva[47]. Pritom se referencira „*update metoda*" definirana u sklopu „*PostPolicy klase*".

[46] „*Post.edit pogled*" referencira „*Post model*" u svrhu prezentacije trenutne vrijednosti pojedinih parametara vezanih uz korisničku objavu.

[47] Korisnik će u suprotnom slučaju biti naveden na ispunjavanje relevantne forme, u sklopu klijentske strane web aplikacije, te će biti obaviješten o eventualnoj zabrani pristupa tek nakon ispostave navedene forme.

6.4.1.8 Update metoda

```
    public function update(PostUpdateRequest $request, Post $post)
    {

        $this->authorize('update', $post);

        if ($request->safe()->only(['image'])) {
            $imagePath = "/storage/".$request->file('image')->store('uploads',
'public');

            $image = Image::make(public_path("{$imagePath}"))->fit(1200,1200);
            $image->save();

            $post->update(array_merge(
                $request->validated(),
                ['image' => $imagePath]
            ));
        }

        else {
            $post->update($request->validated());
        }

        return redirect('/post/' . $post->id);
    }
```

Slika 6.18. "Post kontroler update metoda"

„*Update metoda*“ omogućuje ažuriranje pojedinih parametara referiranog „*Post modela*“, a na osnovu podataka sadržanih u sklopu korisničkog zahtjeva. Podaci vezani uz korisnički zahtjev su ugrađeni u sklopu injektirane instance „*PostUpdateRequest klase*“.

Ovdje je važno primijetiti da se validacijska logika, sadržana u sklopu „*PostUpdateRequest klase*“, razlikuje od validacije definirane u sklopu „*PostStoreRequest klase*“. U ovom slučaju korisnik nije dužan definirati novu sliku korisničke objave u sklopu korisničkog zahtjeva upućenog prema trenutnoj metodi.

U sklopu analizirane metode se prvo vrši autorizacija korisničkog zahtjeva odnosno provjerava se pravo izvršenja „*update akcije*“ nad „*Post modelom*“ iz perspektive trenutnog

korisnika. Nakon toga se provjerava je li u sklopu korisničkog zahtjeva dostavljena nova slika korisničke objave.

Ako je navedena datoteka dostavljena, ona se pohranjuje u prilagođenom obliku unutar web poslužitelja. Preostali parametri korisničke objave se zatim, u kombinaciji s konačnom putanjom slikovne datoteke, ugrađuju unutar odgovarajućeg retka „*posts tablice*“. U slučaju ako korisnik nije definirao novu sliku korisničke objave, pohranjuju se isključivo preostali parametri relevantnog modela.

Metoda u konačnici preusmjerava korisnika na „*show metodu*“ trenutnog kontrolera, uz referenciranje novodefinirane korisničke objave.

6.4.2 UserDetailsController

Ovaj kontroler grupira metode koje pružaju podršku prilikom konfiguracije dodatnih javno dohvatljivih podataka vezanih uz korisnički profil. Ovdje je dakle riječ o javno dohvatljivom opisu korisničkog profila. U nastavku su ukratko opisane metode navedenog kontrolera.

6.4.2.1 Edit metoda

```
public function edit(UserDetails $user_details){

        $this->authorize('update', $user_details);
        return view('user-details.edit', compact('user_details'));
    }
```

Slika 6.19. "UserDetails kontroler edit metoda"

„*Edit metoda*“ izvršava autorizaciju korisnika u kontekstu „*update metode UserDetails klase*“. U slučaju uspješne autorizacije poziva se „*user-details.edit pogled*“, uz prosljeđivanje prethodno generiranog „*UserDetails modela*“. Navedeni pogled generira formu koja omogućuje izmjenu javno dohvatljivih podataka, vezanih uz korisnički profil trenutnog korisnika.

6.4.2.2 Update metoda

```
 public function update(UserDetailsEditRequest $request, UserDetails
$user_details)
    {

        $this->authorize('update', $user_details);

        if ($request->safe()->only(['image'])) {
            $imagePath = "/storage/".$request->file('image')-
>store('userDetails', 'public');

            $image = Image::make(public_path("{$imagePath}"))->fit(1000,1000);
            $image->save();

            $request->user()->user_details->update(array_merge(
                $request->validated(),
                ['image' => $imagePath]
            ));
        }

        else {
            $request->user()->user_details->update($request->validated());
        }

        return redirect('/story/' . $user_details->user->id);
    }
```

Slika 6.20. "UserDetails kontroler update metoda"

„*Update metoda*“ omogućuje pohranu modificiranih podataka vezanih uz „*UserDetails model*“. Validacijska logika, ugrađena u sklopu „*UserDetailsEditRequest klase*“ ne zahtjeva nužnu definiciju nove slikovne datoteke u sklopu korisničkog zahtjeva. Ovdje je riječ o sličnom načinu obrade kao i u sklopu „*update metode Post kontrolera*“. Nakon pohrane navedenih podataka, korisnika se preusmjeruje na „story stranicu“, vezanu uz njegov korisnički profil.

6.5 Sigurnosni mehanizmi klijentske strane web aplikacije

U nastavku su izdvojeni sigurnosni mehanizmi, implementirani u sklopu klijentske strane web aplikacije.

6.5.1 Obrana od CSRF napada

„*Laravel okvir*“ sadrži ugrađeni sigurnosni mehanizam koji omogućuje zaštitu od uspješnog izvođenja „*CSRF napada*“.

Mehanizam se bazira na generiranju unikatnog tokena za svakog korisnika („*CSRF token*“)[48]. „CSRF token“ se pritom pohranjuje u sklopu svake pojedine korisničke sesije. Razvojni programer je dužan ugraditi navedeni token unutar svake generirane korisničke forme klijentske strane web aplikacije. U tu svrhu je moguće koristiti „*@csrf Blade direktivu*[49]“.

„*VerifyCsrfToken middleware*“ omogućuje automatsku provjeru istovjetnosti dostavljenog „*CSRF tokena*“ s relevantnom vrijednošću, pohranjenom u sklopu korisničke sesije. Navedeni „*middleware*“ je definiran u sklopu „*web middleware grupe*“ te se automatski dodjeljuje svim rutama definiranim u sklopu „*routes/web.php datoteke*“.

6.5.2 Obrana od XSS napada

„*Blade echo naredba*“, koja podrazumijeva korištenje „*{{ }} notacije*“, izvršava automatsko filtriranje njezinog injektiranog sadržaja. Navedeni sigurnosni mehanizam se bazira na primjeni „*php htmlspecialchars funkcije*“.

[48] „*CSRF token*“ se također dostavlja korisniku u sklopu „*XSRF-TOKEN kolačića*“. Pri tome se vrijednost „*SameSite atributa kolačića*“ definira kao „*Lax*“.

[49] „*Blade*“ predstavlja „*template processor*“, ugrađen u sklopu „*Laravel okvira*“.

6.5.3 Obrana od DOS napada

„*RECAPTCHA v3*“ predstavlja sigurnosni mehanizam koji omogućuje ocjenu svakog pojedinog korisničkog zahtjeva s ciljem detekcije automatizirane interakcije s predmetnom web aplikacijom. Za razliku od „*RECAPTCHA v2 sustava*“, ovaj mehanizam ne zahtijeva rješavanje izazova od strane korisnika. Primjenom ovog alternativnog rješenja se poboljšava kvaliteta korisničkog iskustva uz istovremenu implementaciju odgovarajuće zaštite. Detalji vezani uz funkcioniranje navedenog sustava su još uvijek tajni, s obzirom da je riječ o novoj tehnologiji implementiranoj od strane Google kompanije. Korištenjem „*RECAPTCHA v3*“ mehanizma je moguće u određenoj mjeri ublažiti rizik vezan uz potencijalno izvršenje „*DOS napada*“.

6.5.3.1 Integracija RECAPTCHA v3 sustava

Klijentski dio „*recaptcha v3 funkcionalnosti*“ je ugrađen u sklopu odgovarajuće „*Vue.js 3 komponente*“ web aplikacije. Navedena komponenta omogućuje presretanje korisničkog zahtjeva, definiranog u sklopu dostavljene forme. Komponenta, nakon presretanja navedenog zahtjeva, izvršava poziv prema „*Google API-u*“ te ugrađuje zaprimljeni „*recaptcha token*“ u sklopu odgovarajućeg skrivenog polja forme. Nakon toga se inicira dostava modificiranog korisničkog zahtjeva na odgovarajuću adresu, definiranu u sklopu predmetne forme.

U svrhu integracije „*recaptcha v3 mehanizma*“ je također definirana odgovarajuća ekstenzija „*Rules klase*“ odnosno „*ReCaptcha klasa*“. U sklopu navedene klase se izvršava verifikacija „*recaptcha tokena*“. Ova verifikacija se bazira na rezultatu odgovarajućeg poziva prema „*Google API-u*“. Navedena klasa se ugrađuje u sklopu validacijskog mehanizma, vezanog uz štićenu rutu. „*ReCaptcha klasa*“ se pritom deklarira kao i svako drugo pravilo, korišteno u sklopu definicije validacijskog mehanizma.

6.6 Analiza sustava za autentifikaciju integriranog u sklopu Laravel okvira

Autentifikacija podrazumijeva proces verifikacije identiteta. Proces se temelji na analizi prezentiranih komponenti potencijalnog identiteta i dokazivanju veze između entiteta, koji prezentira navedene podatke, s određenim poznatim identitetom. Dokazivanje veze između entiteta i određenog identiteta se provodi na osnovu usporedbe komponenti prethodno registriranog i prezentiranog identiteta. Identitet se može opisati kao skup podataka vezanih uz određeni entitet koji omogućuju isključivo referenciranje odnosno razlikovanje navedenog entiteta. Dakle autentifikaciji prethodi identifikacija gdje određeni entitet izlaže potencijalne dokaze o svom identitetu. Kroz proces autentifikacije se navedeni potencijalni dokazi o identitetu prihvaćaju ili odbacuju.

Prilikom analize procesa autentifikacije korisno je, uz obrazloženi proces identifikacije, pojasniti i proces autorizacije. U slučaju uspješne autentifikacije korisnik u pravilu dobiva pristup određenom skupu resursa. Međutim pristup pojedinim resursima unutar sustava može zahtijevati dodatnu verifikaciju. Autorizacija podrazumijeva verifikaciju prava korisnika u kontekstu pristupa određenim štićenim resursima. U slučaju uspješne autorizacije korisnik je u mogućnosti iskoristiti svoja prava pristupa odnosno prava korištenja određenog resursa. Prava korisnika se inicijalno dodjeljuju tijekom procesa registracije korisnika. Navedena prava je naknadno moguće modificirati ovisno o promjeni funkcionalnih i sigurnosnih zahtjeva sustava.

Važno je naglasiti da autorizaciju nije moguće provesti bez prethodne autentifikacije. Autentifikacijom se dokazuje identitet korisnika. Na osnovu navedenog identiteta se zatim provjerava zadovoljavajuća razina vezanih prava koja, u slučaju uspješne autorizacije, omogućuju pristup određenim resursima unutar sustava. Dakle, autentifikacija podrazumijeva verifikaciju identiteta korisnika dok autorizacija podrazumijeva verifikaciju prava pristupa korisnika određenom resursu.

„*Laravel okvir*“ sadrži predefinirane sustave za registraciju i autentifikaciju korisnika. Navedeni mehanizmi su kvalitetno implementirani te ih je moguće koristiti kao osnovu prilikom razvoja vlastitog rješenja za autentifikaciju korisnika. U svrhu boljeg razumijevanja

konkretnog načina implementacije autentifikacijskog mehanizma, korisno je provesti analizu navedenih predložaka. Pri tome se naročita pažnja pridaje analizi ugrađenih sigurnosnih mehanizma.

„*Laravel Breeze*“ predstavlja elegantnu minimalnu implementaciju autentifikacijskih funkcionalnosti, ugrađenih u sklopu „*Laravel okvira*“. S obzirom na navedeno ovaj predložak predstavlja idealan edukacijski model. Sustav koristi „*MVC arhitektonski obrazac*“ i sadrži skupove predefiniranih „*Model*“, „*View*“ i „*Controller klasa*“, uz odgovarajuće predefinirane rute.

6.6.1 Laravel autentifikacijski mehanizam (u užem smislu riječi)

„*Laravel okvir*“ sadrži fleksibilni autentifikacijski mehanizam (u užem smislu riječi). Mehanizam otvara mogućnosti za detaljno prilagođavanje i ekstenziju njegove osnovne funkcionalnosti. Sustav se sastoji od „*Guard komponenti*“ koje sadrže dvije podkomponente, a koje je moguće konfigurirati u sklopu „*guards niza*“ „*auth.php konfiguracijske datoteke*“. Riječ je o „*Guard Driver-u*“ (ponekad se za „*Guard Driver*“ također koristi naziv „*Guard*“) i “*Provider-u*“:

```
'guards' => [
    'web' => [
        'driver' => 'session',     //GUARD ODNOSNO GUARD DRIVER
        'provider' => 'users',     //PROVIDER
    ],
],
'providers' => [
    'users' => [
        'driver' => 'eloquent',              //PROVIDER DRIVER
        'model' => App\Models\User::class,   //PROVIDER MODEL
    ],
```

Slika 6.21. "auth.php" isječak

Logika potrebna za instanciranje odgovarajućih kombinacija „*Guard Driver-a*“ i „*Provider komponenti*“, na osnovu korisničke konfiguracije, nalazi se u sklopu „*AuthManager klase*“.

„*Guard Driver*“ definira procedure vezane uz proces autentifikacije korisnika te pritom koristi usluge „*Provider komponente*“. U sklopu ove komponente su definirani detalji vezani uz proces inicijalne autentifikacije i koraci koje je potrebno provesti u slučaju eventualnog poklapanja korisničkih vjerodajnica s onima pohranjenima u memoriji. To podrazumijeva definiciju načina na koji se korisnici autentificiraju u sklopu svakog sljedećeg zahtjeva. „*Guard Driver komponenta*“ predstavlja implementaciju „*Guard sučelja*“, odnosno podrazumijeva implementaciju „*check*“, „*guest*“, „*user*“, „*id*“, „*validate*“, „*hasUser*“ i “*setUser metoda*“. „*SessionGuard klasa*“ (implementacija „*Session Guard Driver-a*“) implementira „*StatefulGuard sučelje*“, koje predstavlja ekstenziju „*Guard sučelja*“ i uključuje dodatne metode („*attempt*“, „*login*“, „*logout*“ i slično). Detalje procesa autentifikacije korisnika je moguće dodatno modificirati u sklopu modela nad kojim se izvršava proces autentifikacije. U sklopu modela je moguće definirati alternativne implementacije „*AuthenticatableContract sučelja*“. Navedeno sučelje definira „*signature*“ raznih metoda korištenih u svrhu upravljanja autentifikacijom korisnika. „*Laravel okvir*“ sadrži dva predefinirana „*Guard Driver-a*“. Riječ je o „*Session*“ i „*Token guard-u*“. „*Session guard*“ održava stanje autentifikacije korisnika, koristeći kolačiće u sklopu web preglednika. „*Token guard“* za istu svrhu koristi tokene integrirane u sklopu klijentske strane web aplikacije. Ovaj „*Guard Driver*“ se obično koristi u svrhu autentifikacije drugih aplikacija (na osnovu njihovog zahtjeva upućenog prema „*API-u trenutne aplikacije*“). Pozivi između aplikacija se izvršavaju izvan okruženja web preglednika (bez kolačića). Zbog toga je u ovom slučaju prikladno koristiti tokene u svrhu održavanja stanja prijave.

„*Provider komponenta*“ implementira metode koje se koriste prilikom interakcije autentifikacijskog sustava s određenim sustavom za pohranu podataka. „*Provider konfiguracija*“ također sadrži definiciju modela odnosno tablice koja se koristi za pohranu autentifikacijskih podataka. „*Provider driver*“ podrazumijeva klasu koja implementira „*UserProvider sučelje*“. „*Laravel okvir*“ sadrži dvije osnovne verzije „*Provider Driver-a*“: „*Eloquent*“ i „*Database*“. Prva podrazumijeva pristup bazi podataka posredstvom „*Eloquent ORM mehanizma*“. Druga podrazumijeva pristup bazi podataka korištenjem „*database query builder-a*“.

U nastavku slijedi analiza autentifikacijskog mehanizma aplikacije, baziranog na „*Laravel Breeze predlošku*“ (uz manje modifikacije). Autentifikacijski sustav se temelji na korištenju

lozinke prilikom inicijalne autentifikacije i tokena u svrhu održavanja sesije. Prilikom analize fokus je usmjeren na poslužiteljsku stranu odnosno na analizu odgovarajućeg skupa kontrolera, bez specifikacije okruženja svake pojedine klase („*Namespaces*“, „*Aliases*“ i slično).

6.6.2 RegisteredUserController

```php
class RegisteredUserController extends Controller
{
    public function create()
    {
        return view('auth.register');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => ['required', 'string', 'max:255'],
            'email' => ['required', 'string', 'email', 'max:255', 'unique:'.User::class],
            'username' => ['required', 'string', 'max:255', 'unique:'.User::class],
            'password' => [
                'required',
                'confirmed',
                Rules\Password::min(8)
                    ->mixedCase()
                    ->numbers()
                    ->symbols()
                    ->uncompromised()],
        ]);

        $user = User::create([
            'name' => $request->name,
            'username' => $request->username,
            'email' => $request->email,
            'password' => Hash::make($request->password),
        ]);

        event(new Registered($user));

        Auth::login($user);

        return redirect(RouteServiceProvider::HOME);
    }
}
```

Slika 6.22. "RegisteredUserController" isječak

„*RegisteredUserController*“ sadrži dvije metode, imenovane u skladu s „*Laravel konvencijom za imenovanje CRUD operacija*“.

„*Create metoda*“ vraća registracijsku formu posredstvom odgovarajuće klase iz prezentacijskog sloja.

„*Store metoda*“ obrađuje „*POST zahtjev*“ za registraciju korisnika (zahtjev generiran korištenjem prethodno prezentirane forme za registraciju korisnika). Obrada korisničkog zahtjeva se temelji na korištenju „*Request klase*“. Ovdje nailazimo na primjer injekcije instance klase na osnovu deklariranog sučelja („*type hint*“). Navedena injekcija predstavlja jedan od mogućih načina generiranja instance „*Request klase*“, temeljen na korištenju „*Dependency Injection mehanizma*“.

„*Request klasa*“ sadrži odgovarajuće metode koje omogućuju pristup podacima zaprimljenog korisničkog zahtjeva (tu se prvenstveno misli na tijelo forme) te njihovu manipulaciju. U svrhu validacije korisničkog zahtjeva, poziva se „*validate metoda*“ nad „*Request klasom*“. U sklopu validacije korisničkog zahtjeva za registracijom, prvenstveno je potrebno provjeriti zadovoljava li deklarirana lozinka skup prethodno definiranih ograničenja.

U slučaju ako korisnički zahtjev zadovoljava sva pravila definirana u sklopu validacijske metode, pristupa se pohrani registriranog korisnika u bazu podataka. U svrhu postizanja više razine sigurnosti, pohranjuje se isključivo sažetak lozinke. Pri tome se koristi odgovarajuća „*hash funkcionalnost*“, referencirana korištenjem „*make metode*“ „*Hash Facade-a*“.

Slijedi emitiranje „*Registered event-a*“. Korištenjem parametra navedenog „*event-a*“, prenosi se model registriranog korisnika. Mapiranje između „*Registered event-a*“ i odgovarajućeg „*listener-a*“ je definirano u sklopu „*EventServiceProvider-a*“ (jedan od deklariranih „*Service Provider-a*“):

```
class EventServiceProvider extends ServiceProvider
{
    protected $listen = [
        Registered::class => [
            SendEmailVerificationNotification::class,
        ],
    ];
…
```

Slika 6.23. "EventServiceProvider" isječak

„*SendEmailVerificationNotification listener*“ provjerava implementira li prosljeđeni objekt („*User model*“) „*MustVerifyEmail sučelje (contract)*“. „*User model*“ nasljeđuje „*MustVerifyEmail trait*“, u sklopu kojeg je definirana implementacija „*MustVerifyEmail sučelja*“:

```
class SendEmailVerificationNotification
{
    public function handle(Registered $event)
    {
        if ($event->user instanceof MustVerifyEmail && ! $event->user-
>hasVerifiedEmail()) {
            $event->user->sendEmailVerificationNotification();
        }
    }
}
```

Slika 6.24. "SendEmailVerificationNotification" isječak

Ovdje je riječ o jednoj od komponenti sustava za verifikaciju korisnika, koji će biti detaljnije dokumentiran u nastavku ovog Specijalističkog rada.

U slučaju ako „*User model*“ implementira „*MustVerifyEmail sučelje*“ i u slučaju ako email adresa već nije verificirana, korisniku se dostavlja email notifikacija s odgovarajućim linkom za verifikaciju email adrese.

„*Login() metoda*“ „*Auth fasade*“ zatim referencira mehanizam, koji omogućuje direktno dodjeljivanje autentifikacijske sesije netom registriranom korisniku. S obzirom da je u sklopu postojeće sesije već izvršena registracija korisnika, nema potrebe za provjerom lozinke.

Kontroler u konačnici preusmjeruje korisnika na rutu definiranu u sklopu „*HOME konstante*“, registrirane u sklopu „*RouteServiceProvider-a*“.

6.6.3 AuthenticatedSessionController

```php
class AuthenticatedSessionController extends Controller
{
    public function create()
    {
        return view('auth.login');
    }

    public function store(LoginRequest $request)
    {
        $request->authenticate();

        $request->session()->regenerate();

        return redirect()->intended(RouteServiceProvider::HOME);
    }

    public function destroy(Request $request)
    {
        Auth::guard('web')->logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect('/');
    }
}
```

Slika 6.25. "AuthenticatedSessionController" isječak

Ovaj kontroler sadrži metode koje se koriste za prijavu odnosno odjavu korisnika iz sustava te se referenciraju korištenjem „*login rute*“.

6.6.3.1 Create metoda

„*Create metoda*“ vraća pogled s formom za prijavu korisnika. Forma očekuje unos „*password*“ i „“*email parametra*“ uz opcionalni unos „*remember_me parametra*“ (na klijentskoj strani se također vrši validacija korisničkog unosa). Aktivacijom forme se upućuje zahtjev prema „*store metodi*“ istog kontrolera.

6.6.3.2 Store metoda

„*Store metoda*“ injektira instancu „*LoginRequest klase*“. Navedena klasa predstavlja proširenje „*Request klase*“. Unutar „*LoginRequest klase*“ se definiraju validacijska pravila (nisu prikazana u isječku) i metoda za autentifikaciju korisnika.

„*Authenticate metoda*“ koristi „*ensureIsNotRateLimited metodu*“ u svrhu provjere stanja mehanizma, koji prati odnosno ograničava broj pokušaja korisničke prijave unutar zadanog vremenskog intervala („*rate limiting*“). Navedeni „*rate limiting mehanizam*“ se bazira na korištenju odgovarajućih implementacija s registriranom vezom prema „*RateLimiter Facade-u*“. On omogućuje privremeno blokiranje korisničke prijave (u slučaju prekoračenja broja dozvoljenih pokušaja prijave u određenom vremenskom intervalu). Mehanizam također emitira informaciju o preostalom vremenu do reaktivacije funkcionalnosti korisničke prijave. Neuspjeli pokušaji korisničke prijave se registriraju korištenjem ključa baziranog na podacima o korisničkoj „*email*“ i „*IP adresi*“.

Nakon navedene provjere slijedi pokušaj autentifikacije korisnika. Pritom se koristi „*attempt metoda*“ „*Auth Facade-a*“. Navedena metoda kao parametre očekuje niz vjerodajnica korisnika i „*boolean vrijednost*“. U slučaju primitka pozitivne „*boolean vrijednosti*“, pristupa se strukturiranju komponenti „*remember me mehanizma*“. To podrazumijeva generiranje „*remember me kolačića*“ u sklopu relevantnog „*HTTP/S odgovora*“ te pohranu odgovarajuće vrijednosti unutar „*remember_token polja*“ korisničke tablice. U sklopu vrijednosti „*Remember me kolačića*“ je pohranjen skup šifriranih podataka. Ovdje je riječ o identifikatoru korisnika, kriptografskom sažetku korisničke lozinke i vrijednosti prethodno pohranjenog „*remember_token polja*“ korisničke tablice. Vrijeme trajanja navedenog kolačića se definira u

iznosu od 5 godina[50]. Korištenjem „*Remember me kolačića*“ se omogućuje autentifikacija korisnika čak i u slučaju kada se izbrišu podaci vezani uz trenutnu korisničku sesiju[51] ili u slučaju zatvaranja i ponovnog otvaranja web preglednika od strane krajnjeg korisnika.

„*Remember_token polje*“ korisničke tablice predstavlja komponentu sigurnosnog mehanizama, uvedenog u sklopu „*Laravel okvira verzije 4.1.26*[52]“. Korištenjem navedenog stupca se uvodi mogućnost poništavanja „*remember me kolačića*“, prilikom odjave korisnika. U sklopu svake sljedeće korisničke prijave se definira nova vrijednost „*remember_token atributa*“. Prethodne verzije „*remember me tokena*“ postaju nevažeće s obzirom na nepoklapanje ugrađene vrijednosti „*remember_token atributa*“ s njegovom aktualno pohranjenom vrijednošću u sklopu baze podataka web aplikacije. Na ovaj način se implementira jedan oblik obrane od eventualnih „*Remember me cookie hijacking napada*“.

[50] Ovdje je zanimljivo primijetiti da novije verzije *„Google Chrome web preglednika“* (nakon verzije M104) dozvoljavaju maksimalno vrijeme trajanja kolačića u iznosu od 400 dana odnosno 34560000 sekundi.

[51] Prema izvornim postavkama *„Laravel okvira“*, podaci vezani uz pojedinu sesiju se brišu u roku od 120 minuta nakon njezinog stvaranja.

[52] Temeljeno na sljedećem online resursu: https://laravel.com/docs/4.2/upgrade#upgrade-4.1.29

```php
class LoginRequest extends FormRequest
{
    …
    public function authenticate()
    {
        $this->ensureIsNotRateLimited();

        if (! Auth::attempt($this->only('email', 'password'), $this->boolean('remember'))) {
            RateLimiter::hit($this->throttleKey());

            throw ValidationException::withMessages([
                'email' => trans('auth.failed'),
            ]);
        }

        RateLimiter::clear($this->throttleKey());
    }
    public function ensureIsNotRateLimited()
    {
        if (! RateLimiter::tooManyAttempts($this->throttleKey(), 5)) {
            return;
        }

        event(new Lockout($this));

        $seconds = RateLimiter::availableIn($this->throttleKey());

        throw ValidationException::withMessages([
            'email' => trans('auth.throttle', [
                'seconds' => $seconds,
                'minutes' => ceil($seconds / 60),
            ]),
        ]);
    }
    public function throttleKey()
    {
        return Str::transliterate(Str::lower($this->input('email')).'|'.$this->ip());
    }
}
```

Slika 6.26. "LoginRequest" isječak

Implementacija referencirane „*attempt metode*“ ovisi i o konfiguraciji odabrane „*Guard komponente*“. „*Web guard komponenta*“ podrazumijeva korištenje „*SessionGuard klase (Guard Driver)*“ i „*EloquentUserProvider klase (Provider)*“. Korištena implementacija „*attempt metode*“ (definirana u sklopu „*SessionGuard-a*“) prvo emitira „*Attempting event*“. Nakon toga se generira nova instanca „*user modela*“ na osnovu „*retrieveByCredentials metode*“ (definirane u sklopu „*EloquentUserProvider-a*“). Navedena instanca modela se koristi u sklopu „*hasValidCredentials metode*“ (također definirane u sklopu „*EloquentUserProvider-a*“). Navedena metoda vrši konkretnu autentifikaciju korisnika odnosno provjerava vezu između dostavljenih podataka i odgovarajućih podataka pohranjenih u sklopu baze podataka. Ovdje je važno naglasiti da sustav, u sklopu niza korisničkih vjerodajnica (parametara „*attempt metode*“), očekuje definiranu vrijednost „*parametra password*“. Preostali parametri služe isključivo u svrhu generiranja instance odgovarajućeg modela korisnika („*Eloquent ORM*“).

U slučaju uspješne autentifikacije, slijedi izmjena identifikatora sesije (trenutno autentificiranog korisnika):

```
…
$request->session()->regenerate();
…
```

Slika 6.27. Poziv metode za regeneraciju identifikatora sesije

Regeneracija korisničke sesije predstavlja sigurnosni mehanizam, uz pomoć kojeg se smanjuje vjerojatnost uspješnog izvršenja napada, temeljenih na fiksaciji korisničke sesije („*session fixation attacks*“).

6.6.3.3 Destroy metoda

Ova metoda omogućuje odjavu korisnika. Pri tom se referencira jednaka „*Guard komponenta*“ kao u slučaju prijave korisnika. Nakon odjave korisnika potrebno je izbrisati podatke vezane uz trenutnu korisničku sesiju, regenerirati identifikator korisničke sesije i

također izvršiti regeneraciju „*CSRF tokena*“. „*Invalidate metoda*“ omogućuje istovremeno brisanje podataka sesije i regeneraciju njezinog identifikatora. „*RegenerateToken metoda*“ omogućuje regeneraciju „*CSRF tokena*“. Riječ je o tokenu koji se ispostavlja u sklopu formi prezentiranih krajnjem korisniku. Ovaj token omogućuje implementaciju dodatne veze između korisničkog zahtjeva i poslužiteljske strane web aplikacije, neovisne o korisničkoj sesiji. Time se osigurava dodatni sloj obrane (korišten u kontekstu obrane od „*CSRF napada*“). Regeneracija „*CSRF tokena*“ se obično izvršava u sklopu procesa odjave korisnika. U suprotnom može doći do određenih funkcionalnih problema. Prijevremena regeneracija „*CSRF tokena*“ može dovesti do nepoklapanja između očekivanih vrijednosti „*CSRF tokena*“ i vrijednosti pohranjenih unutar stranica, sadržanih unutar memorije web preglednika (u slučaju prebacivanja na prethodno otvorene stranice navedenog web mjesta unutar drugih dijelova preglednika).

6.6.4 Sustav za verifikaciju emaila

„*Laravel okvir*“ sadrži „*Verified middleware*“. „*Middleware*“ provjerava je li korisnik potvrdio svoju email adresu, definiranu u sklopu korisničke registracije. Korištenje navedenog sustava podrazumijeva implementaciju triju kontrolera, referenciranih korištenjem triju ruta imenovanih u skladu s prethodno definiranim pravilima. Riječ je o „*verification.notice*“, „*verification.verify*“ i „*verification.send rutama*“. Pristup svim navedenim rutama je zaštićen primjenom „*Auth middleware-a*“. Dakle pristup je dozvoljen isključivo autentificiranim korisnicima. U nastavku je pojašnjena funkcionalnost triju kontrolera koji implementiraju odgovarajuće metode referencirane od strane navedenih imenovanih ruta.

6.6.4.1 EmailVerificationPromptController

```php
class EmailVerificationPromptController extends Controller
{
    public function __invoke(Request $request)
    {
        return $request->user()->hasVerifiedEmail()
                    ? redirect()->intended(RouteServiceProvider::HOME)
                    : view('auth.verify-email');
    }
}
```

Slika 6.28. "EmailVerificationPromptController" isječak

U slučaju ako korisnik nije verificirao email, a pokuša pristupiti ruti nad kojom je registriran „*Verified middleware*“, bit će preusmjeren na „*verification.notice rutu*“. Navedena ruta referencira „*__invoke“ metodu „EmailVerificationPromptController-a*“. Navedena metoda kontrolera provjerava je li trenutni korisnik verificirao email adresu. Ako je korisnik verificirao email adresu, preusmjeruje ga se na rutu s koje je prvobitno preusmjeren. U suprotnom se vraća odgovarajući pogled odnosno generira se stranica koja prenosi korisniku odgovarajuće informacije. U sklopu „*auth.verify-email pogleda*“ korisnika se obavještava o nužnosti verifikacije emaila. Verifikacijski email se dostavlja na registriranu email adresu korisnika (u sklopu „*RegisteredUserController-a*“). Korisnik je također u mogućnosti pokrenuti proces ponovnog slanja verifikacijskog emaila.

6.6.4.2 VerifyEmailController

```
class VerifyEmailController extends Controller
{
    public function __invoke(EmailVerificationRequest $request)
    {
        if ($request->user()->hasVerifiedEmail()) {
            return redirect()-
>intended(RouteServiceProvider::HOME.'?verified=1');
        }

        if ($request->user()->markEmailAsVerified()) {
            event(new Verified($request->user()));
        }

        return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
    }
}
```

Slika 6.29. "VerifyEmailController" isječak

U slučaju ako je korisnik kliknuo na poveznicu unutar verifikacijskog emaila, bit će preusmjeren na „*verification.verify rutu*“. Navedenoj ruti je dodijeljen „*auth*“ i „*signed middleware*“. Prilikom registracije korisnika, generira se „*signed URL*“ (u sklopu „*sendEmailVerificationNotification() metode*“). Navedeni „*signed URL*“ sadrži dva osnovna parametra. Riječ je o identitetu korisnika i „*hash verziji korisničkog emaila*“. „*Signed URL*“ sadrži dodatni parametar koji predstavlja digitalni potpis „*hash verzije navedenog URL-a*“. Na ovaj način se osigurava integritet linka odnosno omogućuje se detekcija eventualne modifikacije linka. „*Signed middleware*“ izvršava provjeru integriteta poveznice koja je dovela do aktivacije njemu pridružene rute.

„*VerifyEmailController*“ u sklopu svoje „*__invoke metode*“ automatski injektira instancu klase „*EmailVerificationRequest*“ (bez manualne definicije u sklopu „*Service Container-a*“). Riječ je ekstenziji „*FormRequest klase*“ koja predstavlja ekstenziju „*Request klase*“.

```
class EmailVerificationRequest extends FormRequest
{
    public function authorize()
    {
        if (! hash_equals((string) $this->route('id'),
                          (string) $this->user()->getKey())) {
            return false;
        }

        if (! hash_equals((string) $this->route('hash'),
                          sha1($this->user()->getEmailForVerification()))) {
            return false;
        }

        return true;
    }
…
```

Slika 6.30. "EmailVerificationRequest" isječak

U sklopu „*EmailVerificationRequest klase*“ se izvršava autorizacija zahtjeva („*authorize metoda*“ se poziva automatski u sklopu konstruktora klase) odnosno validacija parametara ugrađenih unutar zaprimljenog zahtjeva (poveznica korištena za verifikaciju emaila). Provjerava se jednakost zaprimljenog identifikatora korisnika i odgovarajuće vrijednosti pohranjene unutar baze podataka. Također se uspoređuje vrijednost „*hash parametra*“ s „*hash verzijom korisničkog emaila*“, pohranjenoj unutar baze podataka. Na ovaj način se može pouzdano utvrditi da je trenutni korisnik zaprimio i iskoristio verifikacijski email, koristeći mogućnost pristupa prethodno registriranoj email adresi. Verifikacija emaila se (iz sigurnosne perspektive) koristi u svrhu obrane od automatske registracije korisnika (registracija takozvanih „*bot računa*“) odnosno registracije lažnih korisnika te u svrhu dodatne verifikacije identiteta korisnika. „*EmailVerificationRequest klasa*“ koristi još jedan dodatni sigurnosni mehanizam. U svrhu usporedbe zaprimljenih parametara s odgovarajućim vrijednostima iz baze podataka, koristi se „*hash_equals metoda*“. Ova metoda vraća rezultat uvijek u isto vrijeme. Alternativne (optimizirane) metode, korištene za usporedbu jednakosti vrijednosti parametara, vraćaju rezultat odmah nakon detekcije prvog razlikovnog bita. Na ovaj način, sustav emitira osjetljive podatke vezane uz strukturu parametara nad kojima se izvršava navedena operacija. Na osnovu ovih informacija, napadač može izvesti informirani napad

grubom silom odnosno drastično smanjiti broj zahtijeva koje je potrebno generirati u svrhu pribavljanja osjetljivih podataka. Ovdje je riječ o mogućnosti izvršenja „*timing napada*“.

Nakon izvršene autorizacije zahtjeva, u sklopu „*EmailVerificationRequest klase*“, slijedi pohrana podatka o izvršenoj verifikaciji korisničke email adrese. U svrhu izvršenja navedene registracije, koristi se „*email_verified_at stupac*“ unutar „*users tablice*“. Nakon pohrane navedenog podatka, emitira se „*Verified event*“, uz prijenos trenutnog korisnika.

6.6.4.3 EmailVerificationNotificationController

Ponekad se dogodi slučaj da korisnik unese krivu email adresu tijekom procesa registracije. Također je moguće da korisnik greškom izbriše registriranu email adresu. Kako bi se izbjegli ovi i slični problemi, koristi se „*EmailVerificationNotificationController*“. Funkcionalnost navedenog kontrolera omogućuje ponovno slanje verifikacijskog emaila na osnovu korisničkog zahtjeva (iniciranog korištenjem odgovarajuće poveznice prezentirane u sklopu korisničkog sučelja web aplikacije).

```
class EmailVerificationNotificationController extends Controller
{
    public function store(Request $request)
    {
        if ($request->user()->hasVerifiedEmail()) {
            return redirect()->intended(RouteServiceProvider::HOME);
        }

        $request->user()->sendEmailVerificationNotification();

        return back()->with('status', 'verification-link-sent');
    }
}
```

Slika 6.31. "EmailVerificationNotificationController" isječak

6.6.5 PasswordController

```php
class PasswordController extends Controller
{
    public function update(Request $request)
    {
        $validated = $request->validateWithBag('updatePassword', [
            'current_password' => ['required', 'current_password'],
            'password' => [
                'required',
                'confirmed',

                Rules\Password::min(8)
                    ->mixedCase()
                    ->numbers()
                    ->symbols()
                    ->uncompromised()],
        ]);

        $request->user()->update([
            'password' => Hash::make($validated['password']),
        ]);

        return back()->with('status', 'password-updated');
    }
}
```

Slika 6.32. "PasswordController" isječak

Ovaj kontroler sadrži „*update metodu*“, koja omogućuje validaciju korisničkog zahtjeva i ažuriranje dostavljene korisničke lozinke. Korisnička lozinka se prije same pohrane, pretvara u sažeti oblik uz korištenje odgovarajuće „*hash funkcije*“. Ruta koja referencira ovu metodu je zaštićena s „*auth middleware-om*“.

6.6.6 Sustav za resetiranje lozinke

U svrhu oporavka izgubljene/zaboravljene lozinke korisnik može iskoristiti funkcionalnosti ugrađenog sustava za resetiranje lozinki. Sustav se sastoji od dva kontrolera, od kojih svaki

sadrži dvije metode. Korespondirajuće rute se izvode pod pretpostavkom da korisnik dostavlja zahtjeve koristeći ulogu gosta.

6.6.6.1 PasswordResetLinkController

„*Create metoda*“ „*PasswordResetLinkController-a*“ vraća odgovarajući pogled. U sklopu navedenog pogleda se definira forma koja sadrži polje za „*email parametar*“. Korištenjem navedene forme se inicira zahtjev prema „*store metodi*“, uz prijenos unesene email adrese.

„*Store metoda*“ vrši validaciju zaprimljene email adrese. Na osnovu zaprimljenog podatka o email adresi korisnika, preuzima se povezani model korisnika iz baze podataka. Pritom se koriste metode „*Provider-a*“, definiranog u sklopu „*password konfiguracijskog“* niza unutar „*auth.php konfiguracijske datoteke*“. Nad navedenim modelom se pozivaju odgovarajuće implementacije metoda „*CanResetPassword sučelja*“ u kombinaciji s metodama „*Notifiable trait-a*“. Sve navedeno se izvršava u sklopu mehanizma referenciranog korištenjem „*Password Facade-a*“ odnosno korištenjem implementacije „*PasswordBroker sučelja*“. Cilj navedenih akcija jest ispostavljanje poveznice za resetiranje korisnika, korištenjem email adrese zaprimljene u sklopu dostavljene forme. U sklopu poveznice se dostavlja email adresa korisnika i token koji je prethodno registrirao vezu s „*user modelom*“ (deduciranim na osnovu zaprimljene email adrese).

```php
class PasswordResetLinkController extends Controller
{
    public function create()
    {
        return view('auth.forgot-password');
    }

    public function store(Request $request)
    {
        $request->validate([
            'email' => ['required', 'email'],
        ]);

        $status = Password::sendResetLink(
            $request->only('email')
        );

        return $status == Password::RESET_LINK_SENT
                    ? back()->with('status', __($status))
                    : back()->withInput($request->only('email'))
                            ->withErrors(['email' => __($status)]);
    }
}
```

Slika 6.33. "PasswordResetLinkController" isječak

6.6.6.2 NewPasswordController

„*Create metoda*“ vraća pogled s formom za resetiranje lozinke. Pritom se prosljeđuje korisnički zahtjev odnosno parametri definirani u sklopu linka za resetiranje lozinke. Ovdje je važno napomenuti da se definicije ruta baziraju na provjeri ograničenog dijela „*URL-a*“. To znači da će „*URL*“, s prefiksom koji se poklapa s nekom definicijom ruta, biti proslijeđen istom kontroleru, neovisno o ostatku „*URL-a*“. To je pod pretpostavkom da ne postoji neka druga ruta koja referencira upravo taj „*URL*“. „*Auth.reset-password pogled*“ pohranjuje zaprimljeni token u sklopu „*hidden polja forme*“. Također definira vrijednost „*email polja forme*“, koristeći zaprimljeni „*email parametar*“. Korisnik upisuje željenu lozinku u sklopu „password“ polja i potvrđuje upisanu lozinku u sklopu „*password_confirmation polja*“. Aktiviranje forme rezultira slanjem zahtjeva prema „*store metodi NewPasswordController-a*“.

```php
class NewPasswordController extends Controller
{
    public function create(Request $request)
    {
        return view('auth.reset-password', ['request' => $request]);
    }

    public function store(Request $request)
    {
        $request->validate([
            'token' => ['required'],
            'email' => ['required', 'email'],
            'password' => ['required','confirmed', Rules\Password::min(8)-
>mixedCase()->numbers()->symbols()->uncompromised()],
        ]);

        $status = Password::reset(
            $request->only('email', 'password', 'password_confirmation',
'token'),
            function ($user) use ($request) {
                $user->forceFill([
                    'password' => Hash::make($request->password),
                    'remember_token' => Str::random(60),
                ])->save();

                event(new PasswordReset($user));
            }
        );

        return $status == Password::PASSWORD_RESET
                    ? redirect()->route('login')->with('status', __($status))
                    : back()->withInput($request->only('email'))
                            ->withErrors(['email' => __($status)]);
    }
}
```

Slika 6.34. "NewPasswordController" isječak

„*Store metoda*“ prvo vrši osnovnu validaciju korisničkog zahtjeva. Nakon toga se posredstvom „*Password Facade-a*“ aktivira „*reset metoda PasswordBroker klase*“. Navedena metoda podrazumijeva validaciju odnosa između zaprimljenih podataka. Ugrađeni token omogućuje potvrđivanje trenutnog korisnika kao vlasnika email adrese na koju je dostavljen link za resetiranje lozinke. U sklopu validacije korisničkih podataka provjerava se postojanje

veze između modela korisnika (deduciranog na osnovu dostavljene email adrese) i tokena. Također se provjerava vrijeme važenja tokena te jednakost unesenih vrijednosti „*password*“ i „*password_confirmation polja*“. U slučaju uspješne validacije pohranjuje se nova vrijednost lozinke (uz prethodnu pretvorbu korištenjem „*hash funkcije*“). Također se definira nova vrijednost „*parametra remember_token*“. S obzirom da „*parametar remember_token*“ nije deklariran unutar „*fillable niza user modela*“, potrebno je iskoristiti „*forceFill metodu*“ u svrhu pohrane navedenog skupa parametara.

6.6.7 Sustav za potvrdu lozinke

„*Laravel okvir*“ sadrži „*RequirePassword middleware*“, koji se koristi u svrhu implementacije dodatnog sloja zaštite u okviru pojedinih ruta. U svrhu odobrenja pristupa zaštićenoj ruti, pridruženi „*middleware*“ zahtijeva od korisnika potvrdu korisničke lozinke. Ovaj je „*middleware*“ u pravilu pridružen rutama koje pretpostavljaju autentificiranost trenutnog korisnika, odnosno kojima je ujedno pridružen „*auth middleware*“. U nastavku je prikazan kod „*handle metode*“ navedenog „*middleware-a*“.

```
  public function handle($request, Closure $next, $redirectToRoute = null,
$passwordTimeoutSeconds = null)
    {
        if ($this->shouldConfirmPassword($request, $passwordTimeoutSeconds)) {
            if ($request->expectsJson()) {
                return $this->responseFactory->json([
                    'message' => 'Password confirmation required.',
                ], 423);
            }

            return $this->responseFactory->redirectGuest(
                $this->urlGenerator->route($redirectToRoute ??
'password.confirm')
            );
        }

        return $next($request);
    }
```

Slika 6.35. "RequirePassword middleware" isječak

Navedeni „*middleware*“ koristi direktno ili indirektno metode definirane u sklopu „*ConfirmablePasswordController-a*“.

6.6.7.1 ConfirmablePasswordController

Prethodno navedeni „*middleware*“ preusmjerava korisnika, koji nije nedavno potvrdio lozinku, na „*password.confirm rutu*“, koja referencira „*show metoda „ConfirmablePasswordController-a*“.

„*Show metoda*“ vraća pogled koji implementira formu za potvrdu korisničke lozinke te inicira zahtjev prema „*store metodi*“ vezanoj uz isti kontroler.

„*Store metoda*“ referencira „*validate() metodu*“, prethodno konfigurirane „*web guard komponente*“, koristeći „*guard metodu Auth Facade-a*“. Izabrana implementacija „*validate() metode*“ (deklarirana u sklopu „*Guard sučelja*“ i sadržana u sklopu „*SessionGuard klase*“) očekuje skup vjerodajnica kao svoje parametre. Podrazumijeva se unos „*password parametra*“ i barem još jednog identifikatora korisnika. Na osnovu dodatnog identifikatora sustav kreira instancu trenutnog korisnika te se u nastavku procesa koriste „*Eloquent metode*“ definirane nad navedenim modelom. Ovaj korak izvršenja općenito ovisi o izabranom „*Provider-u*“. U navedenom primjeru se kao dodatni identifikator koristi „*email parametar*“ trenutno autentificiranog korisnika. „*Validate() metoda*“ potvrđuje autentifikaciju korisnika odnosno izvršava ponovnu provjeru usklađenosti korisnički dostavljene kombinacije lozinke i ostalih identifikatora s onima pohranjenima u sklopu baze podataka.

U slučaju uspješne potvrde korisničke lozinke, slijedi pohrana podatka o vremenu izvršenja navedene potvrde. Ovaj podatak se pohranjuje u sklopu korisničke sesije. Na ovaj način je omogućena razmjena podatka o vremenu izvršenja potvrde lozinke s „*RequirePassword middleware-om*“.

```php
class ConfirmablePasswordController extends Controller
{
    public function show()
    {
        return view('auth.confirm-password');
    }

    public function store(Request $request)
    {
        if (! Auth::guard('web')->validate([
            'email' => $request->user()->email,
            'password' => $request->password,
        ])) {
            throw ValidationException::withMessages([
                'password' => __('auth.password'),
            ]);
        }

        $request->session()->put('auth.password_confirmed_at', time());

        return redirect()->intended(RouteServiceProvider::HOME);
    }
}
```

Slika 6.36. "ConfirmablePasswordController" isječak

7 PENETRACIJSKO TESTIRANJE WEB APLIKACIJE

U nastavku Specijalističkog rada je prikazan relativno jednostavan primjer izvođenja napada u kontekstu „*white box penetracijskog testiranja*“. Navedeno testiranje se izvršava nad unaprijeđenom verzijom analizirane web aplikacije. Opseg manualnog testiranja je ograničen na analizu sigurnosti sustava za autentifikaciju korisnika te ranjivosti vezanih uz potencijalno izvršenje „*CSRF*“ i „*SQl injection napada*“. Prilikom izvođenja manualnog penetracijskog testiranja koriste se pojedine značajke „*Burp Suite okvira*“. Uz navedeno se također automatski izvršava široki skup raznih oblika napada na web aplikaciju. U tu svrhu se koristi alat za aktivno skeniranje ranjivosti web aplikacija, ugrađen u sklopu „*OWASP ZAP okvira*“.

7.1 Aktivno skeniranje ranjivosti web aplikacije

Aktivno skeniranje ranjivosti web aplikacije često predstavlja prvi korak u izvršenju procesa penetracijskog testiranja. Prije pokretanja aktivnog testiranja potrebno je izvršiti odgovarajuću konfiguraciju.

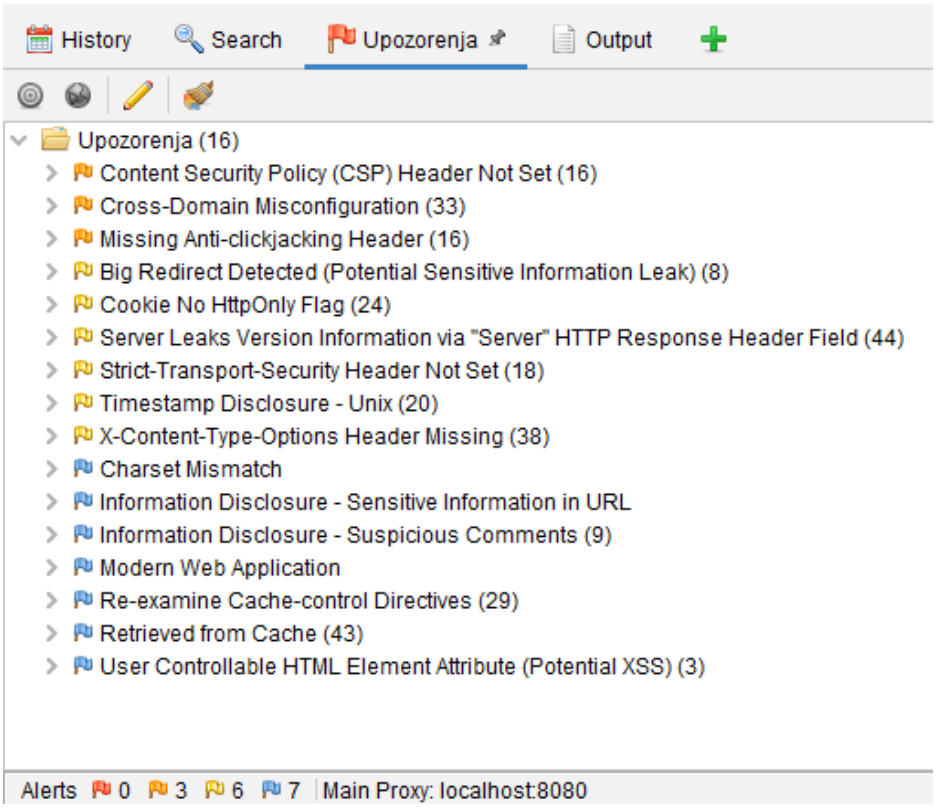
Testirana web aplikacija koristi „*CSRF token*“. Ova činjenica otežava automatizirano izvođenje penetracijskog testiranja. „*OWASP ZAP okvir*“ međutim implementira sustav automatskog regeneriranja „*CSRF tokena*“. Kako bi sustav bio u mogućnosti zaobići „*CSRF zaštitu*“, potrebno je navesti imena korištenih „*CSRF tokena*“ unutar „*Options Anti CRSF Tokens ekrana*“.

Također je potrebno konfigurirati „*OWASP ZAP funkcionalnost automatske prijave*“. Automatsku prijavu je najjednostavnije konfigurirati korištenjem „*POST zahtjeva*“ usmjerenog prema sustavu za prijavu, snimljenog u sklopu manualnog izvođenja prijave. Navedeni „*POST zahtjev*“ je potrebno prvo označiti kao kontekst i zatim izabrati „*Form-based Auth Login Request*“ iz padajućeg izbornika. Korištenjem padajućeg izbornika je potrebno označiti parametre navedenog zahtjeva koji se koriste kao korisničko ime („*username*“) i lozinka („*password*“). U konkretnom slučaju je riječ o „*email*“ i „*password*

parametrima". Unutar navedenih parametara će se, prilikom izvođenja automatske prijave, ugraditi odgovarajuće vrijednosti korisničkog imena i lozinke. Navedene vrijednosti su pohranjene u sklopu strukture podataka vezane uz izabranog „korisnika[53]", u čijem kontekstu se izvršava proces automatskog penetracijskog testiranja.

„*OWASP ZAP okvir*" sadrži mogućnost automatskog detektiranja stanja korisničke prijave. U svrhu aktiviranja ove funkcionalnosti potrebno je definirati odgovarajuće regularne izraze. Regularni izrazi se pritom koriste u svrhu referenciranja dijelova odgovora web aplikacije. Navedeni isječci koda su vezani isključivo uz stanje u sklopu kojeg je korisnik prijavljen odnosno odjavljen iz sustava.

Nakon odgovarajuće konfiguracije moguće je započeti s izvođenjem aktivnog skeniranja ranjivosti web aplikacije. U nastavku je prikazan rezultat izvršenja navedenog procesa.



Slika 7.1. Rezultat aktivnog skeniranja ranjivosti

[53] Ovdje je riječ o korisniku u kontekstu *„OWASP ZAP okvira"*.

Detektiran je skup ranjivosti relativno niskog rizika koje mahom predstavljaju lažno pozitivan rezultat. Valjane detektirane ranjivosti je moguće jednostavno izbjeći odgovarajućom konfiguracijom web aplikacije. Između ostalog se generira „*Cookie No HttpOnlyFlag upozorenje*“, vezano uz „*XSRF-TOKEN*“. Navedeni kolačić predstavlja kriptirani oblik „*CSRF tokena*“, dodijeljenog trenutnom korisniku. Definiranjem „*HttpOnly zastavice*“ u sklopu “*XSRF-TOKEN kolačića*“ se ne ostvaruje značajno unaprjeđenje razine sigurnosti sustava.

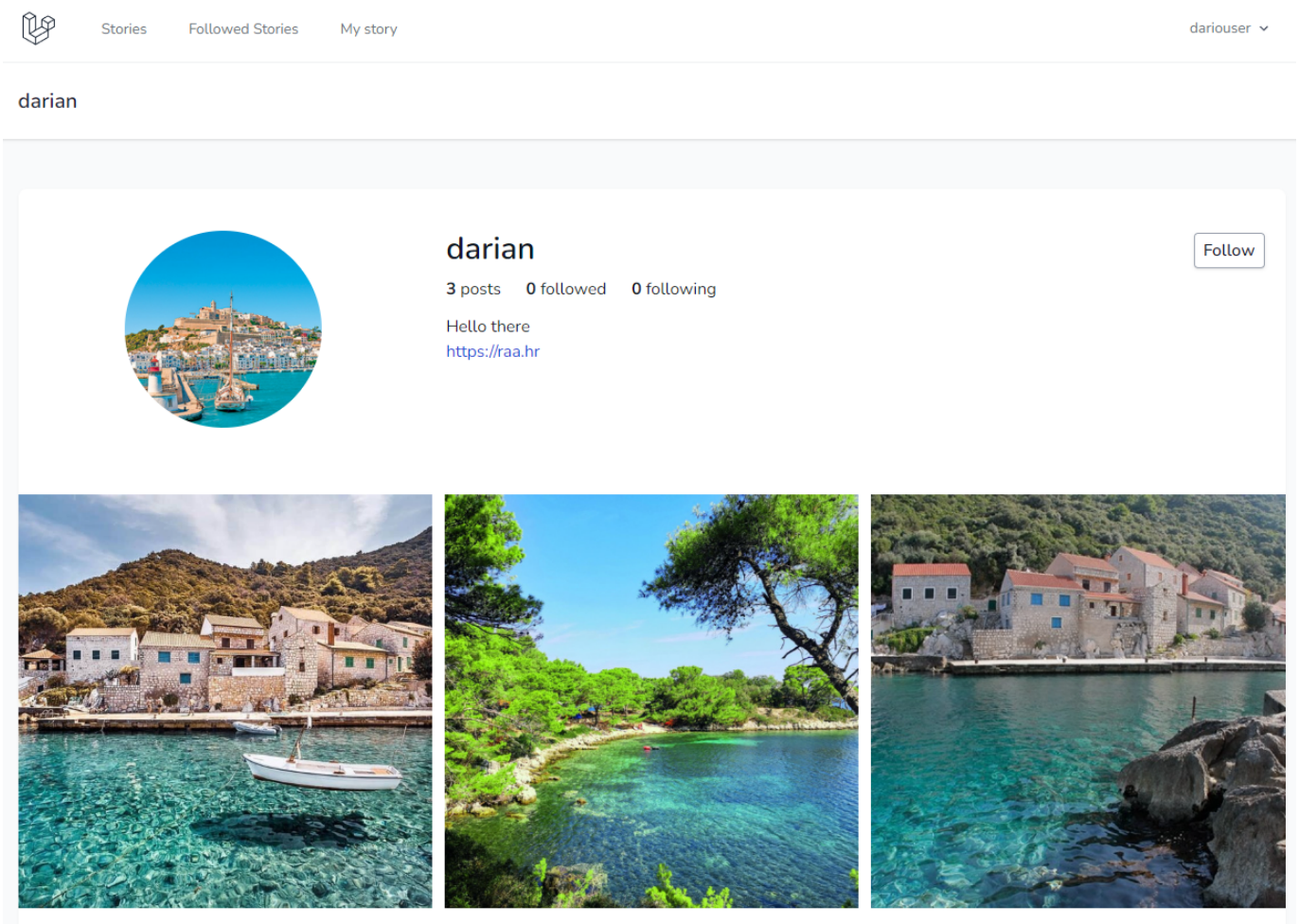
U slučaju ako je napadač u mogućnosti ugraditi zlonamjernu skriptu unutar određenog odgovora web aplikacije (te je ona uspješno pokrenuta u kontekstu određenog korisnika)[54], poznavanje „*CSRF tokena*“ mu ne predstavlja naročito korisnu informaciju u kontekstu daljnje eskalacije napada. Navedeno proizlazi iz činjenice što analizirana web aplikacija implementira redundantne sigurnosne mehanizme. Ovdje je prvenstveno riječ o primjeni ”CSP mehanizma“. Kolačić namijenjen pohrani podataka vezanih uz korisničku sesiju je također zaštićen odgovarajućom deklaracijom “HttpOnly” ali i “SameSite atributa”. Na ovaj način je onemogućeno slanje navedenog kolačića korištenjem injektirane skripte te je otežano izvođenje eventualnog “CSRF napada” (čak i u slučaju poznavanja odgovarajućeg “CSRF tokena”).

S druge strane, definiranje “HttpOnly zastavice” u sklopu “XSRF-TOKEN kolačića”, onemogućuje automatsko procesuiranje navedenog kolačića od strane odgovarajućih javascript biblioteka (npr “axios biblioteke”). U navedenom slučaju nije moguće definirati standardni oblik asinkronog poziva prema aplikaciji bez aktivacije pogreške, generirane od strane “VeriyCsrfToken middleware-a".

[54] Izvršavanje pojedinih skripti je moguće dodatno ograničiti definiranjem ”ContentSecurityPolicy HTTP zaglavlja” odnosno korištenjem ”CSP mehanizma”.

7.2 Testiranje autentifikacije

Prilikom testiranja autentifikacijskog sustava web aplikacije, korisno je izvršiti analizu javno dostupnih podataka vezanih uz pojedini korisnički profil. Na osnovu navedene analize je moguće generirati skup potencijalnih korisničkih imena vezanih uz pojedini korisnički profil. U nastavku je izložen primjer sučelja korisničkog profila web aplikacije i to iz perspektive posjetitelja navedenog profila.



Slika 7.2. Primjer sučelja korisničkog profila web aplikacije

Na osnovu podataka, prezentiranih u sklopu navedenog sučelja, generira se odgovarajuća lista potencijalnih korisničkih imena. Važno je naglasiti da web aplikacija kao korisničko ime, u kontekstu korisničke prijave, očekuje email adresu registriranog korisnika. Ovdje je također

riječ o obliku sigurnosnog mehanizma s obzirom da napadač nije u mogućnosti koristiti liste uobičajenih korisničkih imena. Svaka email adresa se u pravilu koristi od strane samo jednog korisnika.

1. *darian@gmail.com*
2. *raa@gmail.com*
3. *darian@raa.hr*

U nastavku je također izložen primjer skupa često korištenih lozinki:

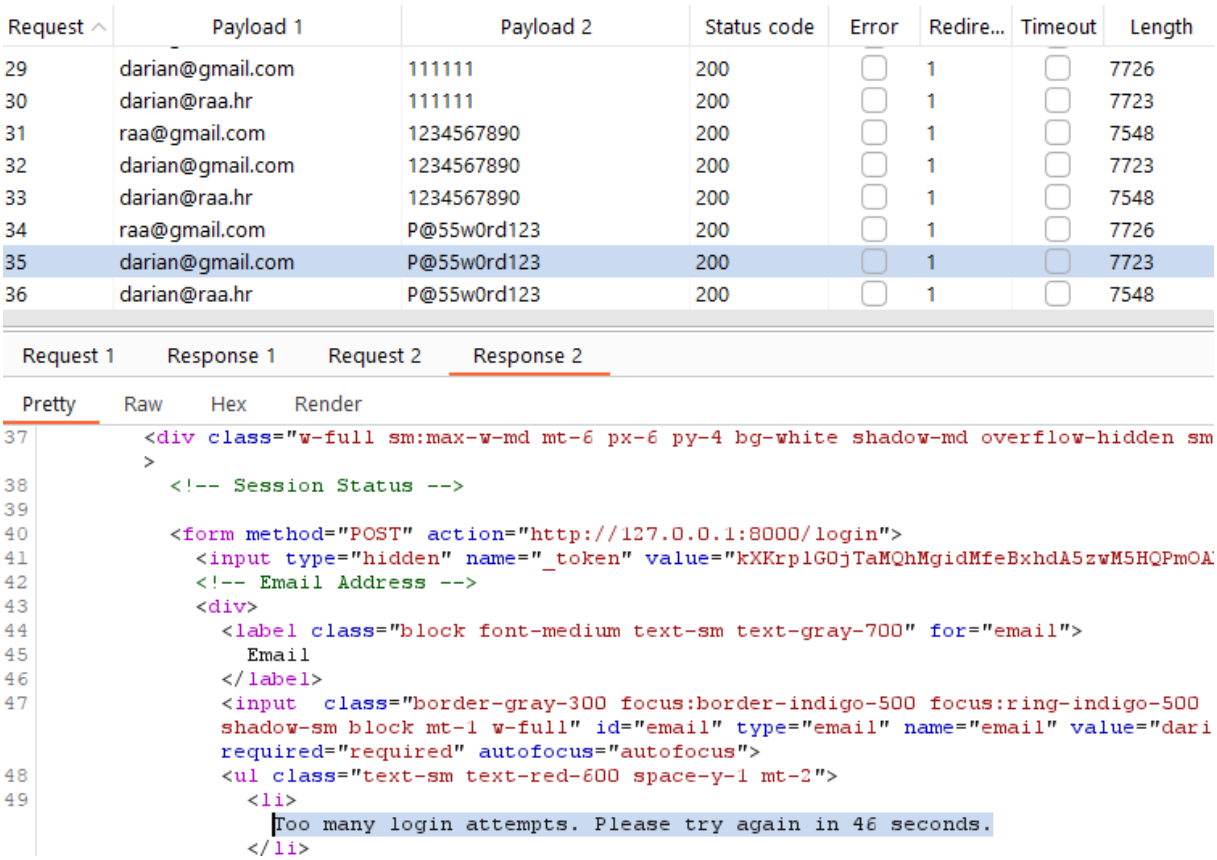
1. *123456*
2. *123456789*
3. *qwerty*
4. *password*
5. *12345*
6. *qwerty123*
7. *1q2w3e*
8. *12345678*
9. *111111*
10. *P@55w0rd123*

U svrhu izvršenja automatiziranog napada grubom silom na web aplikaciju, najprije je potrebno detektirati odgovarajući „*post zahtjev*“ koji referencira funkcionalnost korisničke prijave. Ovo se postiže prethodnim izvršenjem manualne prijave u sustav, uz preporučljivo navođenje krive lozinke (s obzirom na specifičnost implementirane web aplikacije)[55]. Navedena akcija se izvršava korištenjem integriranog web preglednika „*Burp Suite okvira*“ te

[55] Analizirana web aplikacija ne dopušta autentificiranom korisniku ponovni pristup stranici vezanoj uz korisničku prijavu. Autentificirani korisnik je dužan izvršiti odjavu s trenutnog korisničkog računa, prije pristupa funkcionalnosti korisničke prijave.

rezultira s pohranom relevantnog „*post zahtjeva*“, unutar mape stranica. Zahtjev će pritom biti spremljen u sklopu „*login direktorija*“, vezanog uz referencirano web mjesto. „*Post zahtjev*“ je zatim potrebno proslijediti prema „*Burp Intruder alatu*“.

Prije izvođenja samog napada potrebno je izvršiti odgovarajuću konfiguraciju unutar „*Burp Intruder alata*“. Za početak je potrebno konfigurirati alat na način da dopušta izvršenje preusmjeravanja na osnovu „*HTTP redirect odgovora*“, generiranih od strane web aplikacije. Nakon toga je potrebno odabrati „*cluster bomb tip napada*“ te definirati pozicije „*payload-a*“ unutar predmetnog „*post zahtjeva*“. Unutar prvog „*payload-a*“, vezanog uz vrijednost korisničkog imena se ugrađuje prethodno naveden skup pretpostavljenih korisničkih imena. U sklopu drugog „*payloada*“, vezanog uz korisničku lozinku, se ugrađuje prethodno definiran skup uobičajenih lozinki. Na osnovu ove konfiguracije se pokreće napad na web aplikaciju. U nastavku je prezentiran rezultat izvršenja navedenog napada:



Request	Payload 1	Payload 2	Status code	Error	Redire...	Timeout	Length
29	darian@gmail.com	111111	200	☐	1	☐	7726
30	darian@raa.hr	111111	200	☐	1	☐	7723
31	raa@gmail.com	1234567890	200	☐	1	☐	7548
32	darian@gmail.com	1234567890	200	☐	1	☐	7723
33	darian@raa.hr	1234567890	200	☐	1	☐	7548
34	raa@gmail.com	P@55w0rd123	200	☐	1	☐	7726
35	darian@gmail.com	P@55w0rd123	200	☐	1	☐	7723
36	darian@raa.hr	P@55w0rd123	200	☐	1	☐	7548

Request 1 | Response 1 | Request 2 | Response 2

Pretty | Raw | Hex | Render

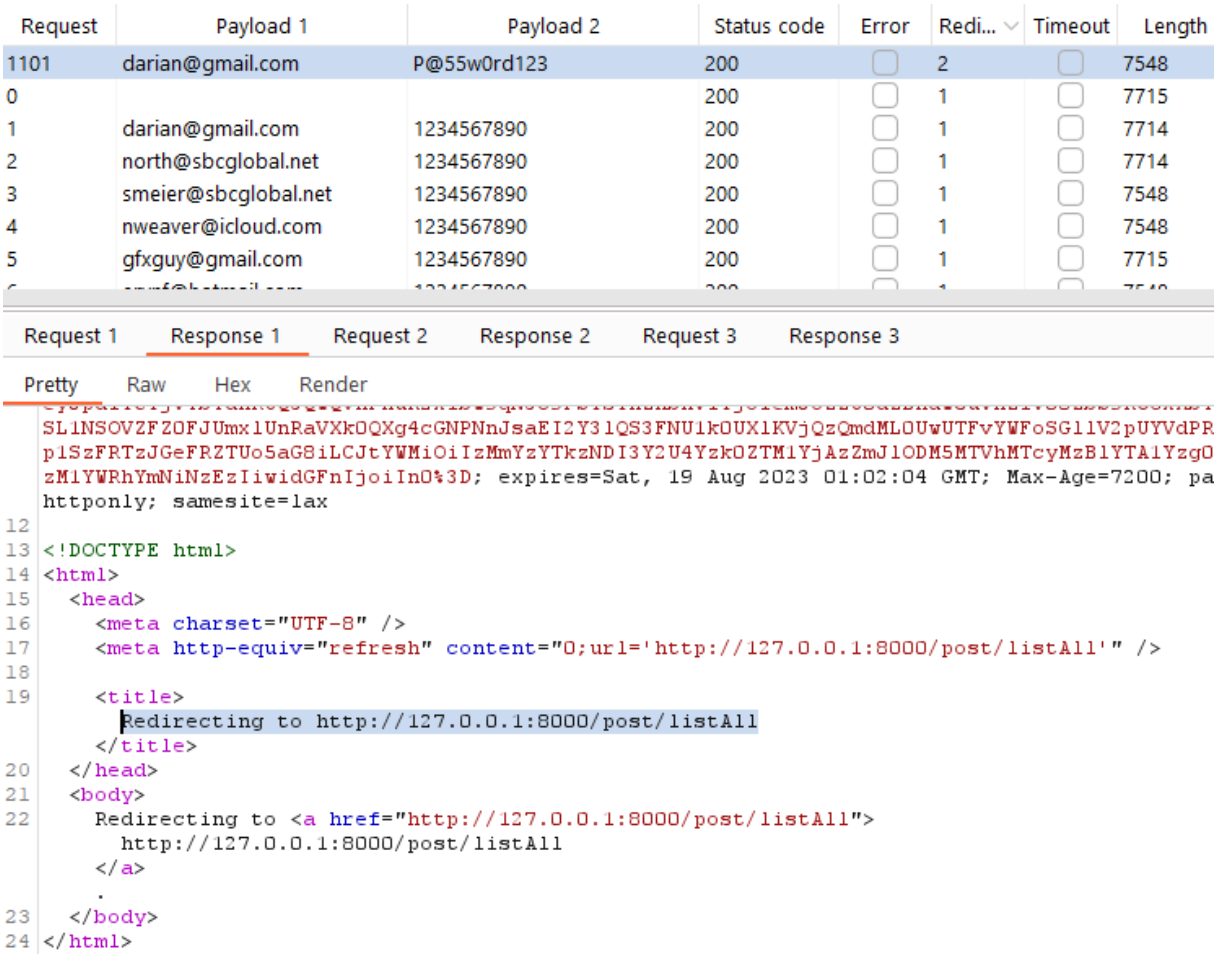
```
<div class="w-full sm:max-w-md mt-6 px-6 py-4 bg-white shadow-md overflow-hidden sm
>
  <!-- Session Status -->

  <form method="POST" action="http://127.0.0.1:8000/login">
    <input type="hidden" name="_token" value="kXKrplGOjTaMQhMgidMfeBxhdA5zwM5HQPmOA
    <!-- Email Address -->
    <div>
      <label class="block font-medium text-sm text-gray-700" for="email">
        Email
      </label>
      <input  class="border-gray-300 focus:border-indigo-500 focus:ring-indigo-500
      shadow-sm block mt-1 w-full" id="email" type="email" name="email" value="dari
      required="required" autofocus="autofocus">
      <ul class="text-sm text-red-600 space-y-1 mt-2">
        <li>
          Too many login attempts. Please try again in 46 seconds.
        </li>
```

Slika 7.3. Prvi pokušaj napada na sustav za autentifikaciju korisnika

Napad je bio bezuspješan odnosno nije rezultirao s probojem. Napadnuta web aplikacija naime sadrži ugrađeni „*throttle mehanizam*“. Navedeni sigurnosni mehanizam, u slučaju većeg broja neuspjelih uzastopnih pokušaja autentifikacije, privremeno onemogućuje izvršenje korisničke prijave. Pri tome se generira obavijest o preostalom vremenu do ponovne aktivacije funkcionalnosti korisničke prijave, vezane uz predmetno korisničko ime.

Detektirani sigurnosni mehanizam je moguće zaobići testiranjem većeg broja potencijalnih korisničkih imena, vezanih uz različite korisničke profile. Pri tome se koristi jednaki tip napada i redoslijed „*payload-a*“, kao u sklopu prethodnog pokušaja proboja web aplikacije[56]. U nastavku je prikazan rezultat izvršenja napada, temeljen na testiranju 100 različitih korisničkih imena uz korištenje prethodno definiranog skupa potencijalnih lozinki. Pri tome su pojedini zahtjevi sortirani u ovisnosti o „*redirect atributu*“.

[56] Napad je oblikovan na način da svaki pojedini generirani zahtjev dostavlja sljedeći element skupa korisničkih imena. Na ovaj način se omogućuje zaobilazak „*throttle sigurnosnog mehanizma*“. Vrijeme trajanja blokade funkcionalnosti korisničke prijave je naime kraće od vremena potrebnog za generiranje novog zahtjeva, vezanog uz isto korisničko ime.



Request	Payload 1	Payload 2	Status code	Error	Redi...	Timeout	Length
1101	darian@gmail.com	P@55w0rd123	200		2		7548
0			200		1		7715
1	darian@gmail.com	1234567890	200		1		7714
2	north@sbcglobal.net	1234567890	200		1		7714
3	smeier@sbcglobal.net	1234567890	200		1		7548
4	nweaver@icloud.com	1234567890	200		1		7548
5	gfxguy@gmail.com	1234567890	200		1		7715

Slika 7.4. Uspješan napad na sustav za autentifikaciju korisnika

Sortiranjem generiranih zahtjeva, na osnovu različitih kombinacija atributa tablice, se omogućuje uočavanje specifičnih oblika odgovora web aplikacije. Prilikom sortiranja, temeljenog na broju izvršenih akcija preusmjeravanja pojedinog zahtjeva, se uočava zanimljivi rezultat. Vrijednost „*redirect atributa*“, vezana uz 1101. zahtjev, se naime razlikuje od vrijednosti sadržane u sklopu svih ostalih elemenata analiziranog skupa.

Na osnovu daljnje analize 1101. zahtjeva se dolazi do zaključka da je izvršenje navedenog zahtjeva rezultiralo s uspješnom prijavom u testiranu web aplikaciju. Relevantan odgovor web aplikacije naime podrazumijeva preusmjeravanje prema „*./post/listAll ruti*“. Ovdje je riječ o odgovoru koji se generira isključivo u slučaju uspješne autentifikacije korisnika, gledano iz perspektive trenutno analizirane web aplikacije.

Web aplikacija u načelu primjenjuje dovoljno stroge uvjete, vezane uz generiranje lozinke iz perspektive pojedinog korisnika. U svrhu ostvarenja više razine sigurnosti, moguće je

primijeniti dodatni skup ograničenja. Ovdje je primjerice riječ o zabrani korištenja skupa najčešće korištenih lozinki, prilikom registriranja novog korisničkog profila. Također je moguće informirati korisnike o sigurnosnim rizicima. Korisnicima bi u tom kontekstu trebalo preporučiti da izbjegavaju javno prezentiranje podataka, na osnovu kojih je moguće pretpostaviti oblik njihove email adrese.

Web aplikacija u načelu generira jednake informacije prilikom svake pojedine instance neuspješne korisničke prijave. Na ovaj način je onemogućeno uspješno izvršenje enumeracije korisničkih imena, korištenjem klasičnog pristupa. „*Throttle sigurnosni mehanizam*“ je moguće dodatno konfigurirati, u ovisnosti o svakoj pojedinoj situaciji[57]. Pri tome je potrebno održati balans između sigurnosnih zahtjeva i kvalitete korisničkog iskustva.

7.3 CSRF napad

Testiranje mogućnosti izvođenja „*CSRF napada*“ se bazira na generiranju raznih oblika zahtjeva prema pojedinim resursima web aplikacije. Pri tome se navedeni zahtjevi upućuju iz perspektive web mjesta koje koristi drugačije izvorište u odnosu na izvorište testirane aplikacije. U svrhu izvođenja „*CSRF napada*“ je dakle potrebno definirati jednostavnu web stranicu, učitati ju u sklopu web poslužitelja te joj dodijeliti domenu koja se razlikuje od domene analizirane web aplikacije.

Prvi korak testiranja podrazumijeva testiranje mogućnosti izvođenja pojedinih „*post zahtjeva*“ [58]iz perspektive testne web stranice. U svrhu osiguravanja mogućnosti izvršenja navedenog oblika zahtjeva, gledajući iz perspektive krajnjeg korisnika, potrebno je generirati odgovarajuću formu. U nastavku je izložen primjer testne web stranice s ugrađenom formom:

[57] U tom kontekstu se izvršava konfiguracija vremenskog intervala, tijekom kojeg je korisniku onemogućeno izvršavanje daljnjih pokušaja prijave. Također je moguće smanjiti broj dozvoljenih pokušaja prijave odnosno broj pokušaja koji ne rezultiraju s privremenom blokadom funkcionalnosti prijave.

[58] Web aplikacija također koristi *„DELETE“* i *„PATCH metode“* prilikom definicije ruta u sklopu *„web.php datoteke“.* Zahtjevi vezani uz navedene rute se formalno klasificiraju kao *„post zahtjevi“*. Pritom se u sklopu navedenih zahtjeva, korištenjem skrivenog polja, dostavlja naziv konačno referencirane metode.

```html
<html>
    <body>
        <form action="http://ranjiva-aplikacija/resurs/" method="POST">
            <input type="hidden" name="title" value="injektirani_naziv"/>
            <input type="hidden" name="description" value="injektirani_opis"/>
        </form>
        <script>
            document.forms[0].submit();
        </script>
    </body>
</html>
```

Slika 7.5. Web stranica za testiranje s ugrađenom formom

Izložena web stranica dakle omogućuje automatsko generiranje „*post zahtjeva*“, bez potrebe za dodatnim izvršenjem korisničke akcije. Pri tome je naravno potrebno na određeni način potaknuti korisnika web aplikacije da pristupi web mjestu koje poslužuje navedeni „*html dokument*“. Kao pomoć pri generiranju testne forme, prikladno je koristiti odgovarajuće alate integrirane u sklopu „*Burp Suite*“ ili „*OWASP ZAP okvira*“[59].

Ovaj oblik napada nije moguće izvesti u kontekstu trenutno analizirane web aplikacije. To proizlazi iz činjenice što web aplikacija sadrži sigurnosni mehanizam, baziran na korištenju „*CSRF tokena*“ u sklopu svake pojedine forme.

Daljnjim izvršenjem manualne analize web aplikacije se uočava pogreška u implementaciji web aplikacije odnosno pogreška vezana uz imenovanje odgovarajuće rute. Navedeni propust je uočen u sklopu komponente vezane uz praćenje ili otpraćivanje korisničkog profila. Prilikom referenciranja navedene funkcionalnosti se umjesto preporučene „*POST metode*“ koristi „*GET metoda*“ odnosno ne poštuju se smjernice vezane uz imenovanje pojedinih ruta. Ova ranjivost omogućuje zaobilaženje sigurnosnog mehanizma web aplikacije, temeljenog na korištenju „*CSRF tokena*“. „*VerifyCsrfToken middleware*“ naime ne izvršava verifikaciju „*query CSRF parametra*“ u slučaju generiranja „*GET zahtjeva*“.

[59] Ovdje je riječ o „*CSRF PoC generatoru*“ sadržanom unutar „*Burp Suite okvira*“ i „*Generate Anti-CSRF Test FORM opciji*“, sadržanoj u sklopu „*OWASP ZAP okvira*“.

U svrhu testiranja mogućnosti eksploatacije navedenog oblika ranjivosti potrebno je konstruirati odgovarajući „*html dokument*“. Primjer navedenog dokumenta je prikazan u nastavku:

```
<html>
    <body>
        <img src="http://127.0.0.1:8000/follow/3" width="0" height="0">
        <a href="http://127.0.0.1:8000/follow/3">Website</a>
    </body>
</html>
```

Slika 7.6. Web stranica za testiranje s ugrađenim linkovima

Testna web stranica u ovom slučaju automatski (prilikom pokretanja) generira „*get zahtjev*“, vezan uz praćenje odnosno otpraćivanje korisničkog profila napadača. Navedeni zahtjev je definiran u sklopu slikovnog elementa odnosno „*img taga*“. Prilikom generiranja „*get zahtjeva*“, vezanog za preuzimanje sadržaja navedenog slikovnog elementa, web preglednik koristi vrijednost njegovog „*src atributa*“. U sklopu testne web stranice je također definiran link odnosno „*anchor tag*“ koji referencira ranjivu komponentu web aplikacije.

Korištenjem navedene testne web stranice se dolazi do zaključka da izvršenje automatiziranog generiranja zahtjeva, korištenjem slikovnog elementa, nije izvedivo. „*Laravel okvir*“ naime postavlja „*Lax vrijednost*“ u sklopu „*SameSite atributa*“, vezanog uz „*laravel_session*“ i „*XSRF-TOKEN kolačiće*“. Na ovaj način je onemogućeno slanje tokena trenutne korisničke sesije, u sklopu generiranog „*get zahtjeva*“. Važno je napomenuti da neki web preglednici, u slučaju ako vrijednost „*SameSite atributa kolačića*“ nije definirana, automatski dodjeljuju vrijednost „*Lax*“ navedenom atributu[60].

Napad je ipak izvediv u slučaju ako se korisnika web aplikacije uspije navesti da pristupi linku koji referencira ranjivu komponentu web aplikacije. Uspješno izvršenje napada rezultira s registracijom praćenja korisničkog profila napadača od strane napadnutog korisnika web aplikacije.

[60] Navedena funkcionalnost je primijenjena u sklopu novijih verzija *„Google Chrome web preglednika“*, međutim ne primjenjuje se (prema izvornim postavkama) u sklopu *„Firefox web preglednika“*.

Može se zaključiti da „*Laravel okvir*“ sadrži kvalitetne sigurnosne mehanizme koji pomažu u obrani od „*CSRF napada*“. Uočena ranjivost je vezana uz pogrešku u implementaciji predmetne web aplikacije[61]. Demonstrirano je kako čak i relativno mala pogreška, nastala tijekom razvoja sustava, može omogućiti zaobilazak ugrađenog sigurnosnog mehanizma.

7.4 SQL injection napad

Prethodnim pokretanjem automatskog penetracijskog testiranja je izvršen skup inicijalnih provjera vezanih uz potencijalne „*SQL injection ranjivosti*“. Nad izdvojenim skupom ulaznih točaka je zatim moguće provesti dodatni skup fokusiranih provjera, korištenjem opširnijih odnosno specijaliziranih „*fuzzing lista*“[62].

Dodatno testiranje se izvršava nad „*post.update rutom*“ web aplikacije. Navedena ruta omogućuje ažuriranje pojedine korisničke objave, na osnovu dostavljenih korisničkih podataka. U svrhu izvođenja napada se koristi „*Burp intruder alat*“.

Prvi korak vezan uz oblikovanje napada podrazumijeva prijavu u sustav te posjećivanje „*post.update rute*“, korištenjem integriranog web preglednika „*Burp Suite okvira*“. Pohranjeni zahtjev je zatim potrebno proslijediti prema „*Burp Intruder alatu*“[63]. Nakon toga je potrebno izvršiti odgovarajuću konfiguraciju navedenog alata. Analizirana web aplikacija generira „*redirect odgovor*“ nakon uspješnog spremanja novih podataka u sklopu pojedine korisničke objave. U skladu s tim je potrebno dopustiti izvršavanje preusmjeravanja, iz perspektive „*Burp Intruder alata*“. Također je potrebno odabrati „*sniper tip napada*“ te označiti pozicije

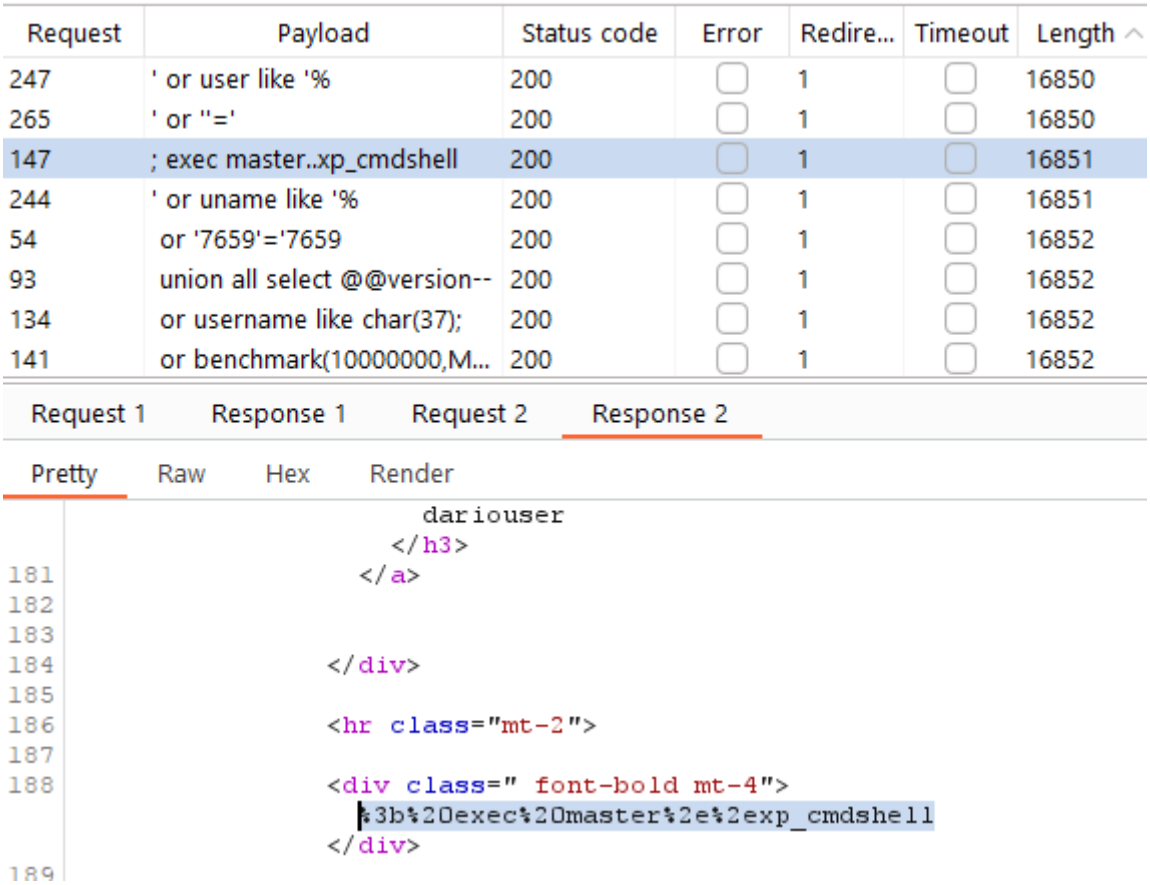
[61] U svrhu rješenja navedenog problema, potrebno je registrirati pravilnu metodu rute („*POST metoda*“), vezane uz funkcionalnost praćenja ili otpraćivanja korisnika. Također je potrebno modificirati „*Vue.js 3 komponentu*“, koja omogućuje asinkrono izvođenje zahtjeva prema navedenoj ruti. Dakle umjesto „*axios.get naredbe*“ je potrebno koristiti „*axios.post naredbu*“.

[62] Kao primjer specijalizirane liste se može navesti lista koja omogućuje testiranje odnosno zaobilaženje autentifikacijske funkcionalnosti web aplikacije.

[63] Također je moguće direktno proslijediti željenu poziciju ugradnje zlonamjernih „*SQL upita*“ prema „*Burp Intruder alatu“.* Označena vrijednost parametra analiziranog zahtjeva, nad kojim je pozvana „*Send to Intruder naredba*“, se u tom slučaju automatski registrira kao „*payload pozicija*“.

„*payload-a*“ u sklopu analiziranog „*post zahtjeva*“. U konkretnom slučaju se označava vrijednost „*title parametra*“.

U nastavku je demonstriran rezultat izvršenja „*SQl injection napada*“ prema „*post.update ruti*“ web aplikacije. Pri tome se koristi lista vezana uz izvođenje generičkih „*SQL injection napada*“[64].



Slika 7.7. Rezultat izvršenja "SQL injection napada"

Napad nije rezultirao s detekcijom eventualnog izlaska izvan konteksta referencirane „*SQl naredbe*“, koja se izvršava iz perspektive napadnute web aplikacije.

Web aplikacija sadrži robusni sigurnosni mehanizam, koji omogućuje obranu od „*SQl injection napada*“. Ovdje je riječ o parametrizaciji „*SQL upita*“ odnosno korištenju „*PDO sustava*“. Navedeni sustav predstavlja bazu izvršenja većine metoda „*Laravel okvira*“, vezanih uz upravljanje s bazom podataka. Pri tome je važno naglasiti da se u sklopu

[64] Navedena lista je preuzeta sa sljedećeg web mjesta: https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/SQLi/Generic-SQLi.txt

konkretne implementacije web aplikacije, prilikom pristupa bazi podataka, koriste isključivo sigurne metode „*Laravel okvira*“.

8 ZAKLJUČAK

„*Laravel okvir*“ predstavlja kvalitetno rješenje koje je prikladno koristiti prilikom razvoja sigurne web aplikacije. Navedeni zaključak je temeljen na analizi izloženog primjera implementacije web aplikacije te na izvođenju procesa penetracijskog testiranja nad navedenom implementacijom.

Razumijevanje ključnih sigurnosnih mehanizama, ugrađenih u sklopu „*Laravel okvira*“, predstavlja preduvjet za sigurnu implementaciju web aplikacije, temeljene na korištenju navedenog okvira. Pokazuje se da čak i male pogreške, a koje nastaju kao rezultat nepotpunog razumijevanja korištenih alata, mogu rezultirati pojavom značajnih ranjivosti sustava. Navedene ranjivosti u određenim situacijama mogu omogućiti zaobilazak ključnih sigurnosnih mehanizama implementirane web aplikacije.

U sklopu Specijalističkog rada je izložen opis ključnih sigurnosnih mehanizama koji pružaju podršku u kontekstu sigurnog razvoj weba aplikacije. Pri tome je poseban fokus usmjeren na analizu implementacije sustava za autentifikaciju korisnika, ugrađenog u sklopu „*Laravel okvira*“.

Uz navedeno su također dokumentirani detalji vezani uz korištenje dvaju najpopularnijih alata za penetracijsko testiranje web aplikacija te je izložen uvid u cjelokupan proces sigurnog razvoja softvera i penetracijskog testiranja.

9 LITERATURA

[1] Gartner: Top 5 Markets In $4.6 Trillion Tech Industry for 2023, 2024., dostupno na: https://nwncarousel.com/news-room/news/gartner-top-5-markets-in-4-6-trillion-tech-industry-for-2023-2024/, pristupljeno u kolovozu 2023.

[2] Gartner's 8 Cybersecurity Predictions for 2023-2025, dostupno na: https://krontech.com/gartners-8-cybersecurity-predictions-for-2023-2025, pristupljeno u kolovozu 2023.

[3] Perri L., Top Strategic Cybersecurity Trends for 2023, dostupno na: https://www.gartner.com/en/articles/top-strategic-cybersecurity-trends-for-2023, pristupljeno u kolovozu 2023.

[4] What Is SDLC (Software Development Lifecycle)?, dostupno na https://aws.amazon.com/what-is/sdlc/, pristupljeno u kolovozu 2023.

[5] Juneja K., What is SDLC? Understanding Top Fundamentals, Phases, and Methodologies, dostupno na: https://www.kellton.com/kellton-tech-blog/what-is-software-development-life-cyclem, pristupljeno u kolovozu 2023.

[6] Secure Software Development Lifecycle (SSDLC), dostupno na: https://snyk.io/learn/secure-sdlc/, pristupljeno u kolovozu 2023.

[7] Dawson M., Norman Burrell D., Rahim E., Brewster S., Integrating Software Assurance into the Software Development Life Cycle (SDLC), dostupno na: https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC, pristupljeno u kolovozu 2023.

[8] OWASP Secure Coding Practices-Quick Reference Guide, dostupno na: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/, pristupljeno u kolovozu 2023.

[9] SEI CERT Coding Standards, dostupno na: https://wiki.sei.cmu.edu/confluence/display/seccode, pristupljeno u kolovozu 2023.

[10] O'Shea B., SCA vs SAST: what are they and which one is right for you?, dostupno na: https://github.blog/2022-09-09-sca-vs-sast-what-are-they-and-which-one-is-right-for-you/, pristupljeno u kolovozu 2023.

[11] OWASP Web Security Testing Guide, dostupno na: https://owasp.org/www-project-web-security-testing-guide/, pristupljeno u kolovozu 2023.

[12] Bullard R., The Maintenance Phase in the Software Life Cycle, dostupno na: https://www.techwalla.com/articles/the-maintenance-phase-in-the-software-life-cycle, pristupljeno u kolovozu 2023.

[13] Security Information And Event Management (SIEM), dostupno na: https://netacea.com/glossary/security-information-and-event-management-siem/, pristupljeno u kolovozu 2023.

[14] PTES Technical Guidelines, dostupno na: http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines, pristupljeno u kolovozu 2023.

[15] PortSwigger Web Security Academy, dostupno na: https://portswigger.net/web-security, pristupljeno u kolovozu 2023.

[16] Stuattard D., Pinto M. (2011). The Web Application Hacker's Handbook - Finding and Exploiting Security Flaws. Izdavač: Wiley Publishing, Inc. (2011).

[17] Cross-Origin Resource Sharing (CORS), dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#simple_requests, pristupljeno u kolovozu 2023.

[18] Burp Suite documentation, dostupno na: https://portswigger.net/burp/documentation, pristupljeno u kolovozu 2023.

[19] What Is Certificate Pinning?, dostupno na: https://www.sectigo.com/resource-library/what-is-certificate-pinning, pristupljeno u kolovozu 2023.

[20] Kensler J, Statistical Hypothesis Testing, 2018.

[21] 5 Real-World Cross Site Scripting Examples, dostupno na: https://websitesecuritystore.com/blog/real-world-cross-site-scripting-examples/, pristupljeno u kolovozu 2023.

[22] Privilege Escalation Attack & Defense Explained, dostupno na: https://www.beyondtrust.com/blog/entry/privilege-escalation-attack-defense-explained, pristupljeno u kolovozu 2023.

[23] Agarwal A., Ajax Sniffer - Prrof of concept, dostupno na: https://myappsecurity.blogspot.com/2007/01/ajax-sniffer-prrof-of-concept.html, pristupljeno u kolovozu 2023.

[24] Laravel dokumentacija 10 verzija, dostupna na: https://laravel.com/docs/10.x/releases, pristupljeno u kolovozu 2023.

[25] Laravel dokumentacija 4.2 verzija, dostupno na: https://laravel.com/docs/4.2/upgrade#upgrade-4.1.29, pristupljeno u kolovozu 2023.

Prilog A

Primjer sigurne web aplikacije, razvijene korištenjem „Laravel okvira“.

Životopis

Darian Žuvić je rođen 15.9.1988. u Splitu.

2014. godine završava Sveučilišni preddiplomski studij računarstva, smjer Računarska znanost na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu.

U razdoblju od 2016. godine do 2017.godine radi u firmi Raafourty d.d. (OIB: 17742557834) kao voditelj održavanja IT sustava.

2017. godine završava Sveučilišni preddiplomski studij poslovne ekonomije na Ekonomskom fakultetu Sveučilišta u Zagrebu.

U razdoblju od 2017. godine do 2019. godine je zaposlen u firmi Rafourty d.d. (OIB: 17742557834) u svojstvu direktora.

2019. godine završava Sveučilišni diplomski studij poslovne ekonomije, smjer Menadžerska informatika na Ekonomskom fakultetu Sveučilišta u Zagrebu.

U razdoblju od 2019. godine do 2020. godine je zaposlen u firmi Banovina Raafourty Holding d.o.o. (OIB: 29138652181) u svojstvu direktora.

U razdoblju od 2020. godine do danas je zaposlen u firmi Raafourty Holding d.o.o. (OIB: 66376790716) u svojstvu direktora.

Curriculum Vitae

Darian Žuvić was born on September 15, 1988. in Split.

In 2014, he completed his undergraduate studies in computer science, majoring in Computer Science at the Faculty of Electrical Engineering and Computing, University of Zagreb.

In the period from 2016 to 2017, he worked in the company Raafourty d.d. (OIB: 17742557834) as IT system maintenance manager.

In 2017, he completed his undergraduate studies in business economics at the Faculty of Economics of the University of Zagreb.

In the period from 2017 to 2019, he was employed at the company Rafourty d.d. (OIB: 17742557834) in the capacity of director.

In 2019, he completed his university graduate studies in business economics, majoring in Managerial Informatics at the Faculty of Economics of the University of Zagreb.

In the period from 2019 to 2020, he was employed at Banovina Raafourty Holding d.o.o. (OIB: 29138652181) in the capacity of director.

In the period from 2020 until today, he was employed at Raafourty Holding d.o.o. (OIB: 66376790716) in the capacity of director.