Istraživanje primjenjivosti alata otvorenog koda u svrhu unaprjeđenja sigurnosti web aplikacija

Sekondo, Mario

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Transport and Traffic Sciences / Sveučilište u Zagrebu, Fakultet prometnih znanosti**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:119:636762

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-07**

Repository / Repozitorij:

Faculty of Transport and Traffic Sciences - Institutional Repository

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI

Mario Sekondo

ISTRAŽIVANJE PRIMJENJIVOSTI ALATA OTVORENOG KODA U SVRHU UNAPRJEĐENJA SIGURNOSTI WEB APLIKACIJA

DIPLOMSKI RAD

Zagreb, 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET PROMETNIH ZNANOSTI
POVJERENSTVO ZA DIPLOMSKI ISPIT

Zagreb, 25. svibnja 2021.

Zavod: **Zavod za informacijsko komunikacijski promet**
Predmet: **Sigurnost i zaštita informacijsko komunikacijskog sustava**

DIPLOMSKI ZADATAK br. 6527

Pristupnik: **Mario Sekondo (0135237471)**
Studij: Promet
Smjer: Informacijsko-komunikacijski promet

Zadatak: **Istraživanje primjenjivosti alata otvorenog koda u svrhu unaprjeđenja sigurnosti web aplikacija**

Opis zadatka:

U diplomskom radu potrebno je pružiti pregled dosadašnjih istraživanja i području sigurnosti web rješenja. Potom, potrebno je analizirati tehnologije primjenjive u razvoju web aplikacije. Nastavno, potrebno je klasificirati i objasniti najčešće ranjivosti prisutne u web aplikacijama te analizirati alate otvorenog koda i njihovu primjenjivost i utjecaj na unaprjeđenje sigurnosti web aplikacija.

Mentor:

dr. sc. Ivan Cvitić

Predsjednik povjerenstva za diplomski ispit:

Sveučilište u Zagrebu

Fakultet prometnih znanosti

DIPLOMSKI RAD

Istraživanje primjenjivosti alata otvorenog koda u svrhu unaprjeđenja sigurnosti web aplikacija

Exploring the Applicability of Open-Source Tools for Web Application Security Improvement

Mentor: dr.sc. Ivan Cvitić

Student: Mario Sekondo

JMBAG: 0135237471

Zagreb, rujan 2021.

Istraživanje primjenjivosti alata otvorenog koda u svrhu unaprjeđenja sigurnosti web aplikacija

SAŽETAK

Sigurnost informacijsko-komunikacijskog sustava je ključna kako bi izbjegli potencijalne kibernetičke napade. Web aplikacije su najizloženije napadima stoga je vrlo važno utvrditi koje su najčešće ranjivosti i najbolji alati kako bi poboljšali sigurnost takvih aplikacija. U ovom radu analizirat će se tehnologije koje se koriste pri razvoju web aplikacija, najčešće ranjivosti s kojima se susreću i alati otvorenog koda koji se mogu iskoristiti u svrhu poboljšanja sigurnosti web aplikacija. Putem analize ovih alata donijeti će se zaključak o primjenjivosti istih te prijedlog implementacije alata otvorenog koda u razvojni proces aplikacije.

KLJUČNE RIJEČI: web aplikacija, sigurnost, alat otvorenog koda, ranjivost

SUMMARY

The security of the information-communication system is crucial to avoid potential cyber attacks. Web applications are most vulnerable to attacks so it is very important to determine what the most common vulnerabilites are and the best tools to improve the security of such applications. This paper will analyze technologies that are used for development of web applications, the most coommon vulnerabilities encountered and open source tools that can be used to improve web application security. Through the analysis of these tools, a conclusion will be made on their applicability and the proposal for the implementation of open source tools in the application development process.

KEY WORDS: web application, security, open source tools, vulnerability

Sadržaj

1. Uvod

U današnjem svijetu tehnologija je neizostavan dio ljudskih života. Razvoj Interneta i informacijskih tehnologija promijenio je način života u kojem sada ljudi posvećuju sve više slobodnog vremena koristeći razne informacijsko-komunikacijske usluge, ili ih pak koriste za obavljanje rada. Sve češće poslovanja proširuju ili mijenjaju način poslovanja upravo zbog razvoja ovih tehnologija. Bilo da poslovanja izrađuju vlastita web rješenja ili angažiraju kompanije koje se bave razvojem takvih rješenja, bitno je imati na umu da je sigurnost i zaštita informacijsko-komunikacijskog sustava ključna komponenta.

Web aplikacije sve su češće žrtve kibernetičkih napada, te tako ugrožavaju i svoje krajnje korisnike. Stoga sve više poslovanja angažira kompanije čije se usluge temelje na poboljšanju sigurnosti informacijskih sustava kako bi validirali i testirali svoje sustave. Takve kompanije obično imaju razvijenu vlastitu metodologiju pristupa sigurnosti i vlastite alate kojima provode svoje usluge, što na kraju rezultira skupocjenom uslugom što možda i nije prihvatljivo za manja poslovanja s troškovnog aspekta. Stoga je bitno razmotriti i alternativne načine kojim bi poslovanja mogla unaprijediti sigurnost vlastitih aplikacija ili web stranica.

Ranjivosti *web* aplikacija mogu proizlaziti iz različitih izvora, a za rješavanje sigurnosnih propusta potrebna je određena količina znanja i resursa kako bi se uspješno unaprijedila sigurnost sustava. Jedan od načina unaprjeđenja sigurnosti *web* aplikacija je korištenje softvera otvorenog koda za detektiranje ranjivosti. Obično takvi alati su besplatni ili pak imaju prihvatljivu cijenu koja odgovara manjim poslovanjima. Glavni zadaci takvih alata su pronalaženje ranjivosti i nedostataka u *web* aplikacijama, a mogu pružati automatsko pronalaženje, ili pak konfiguraciju prema željenim parametrima.

Ovaj diplomski rad podijeljen je u sedam poglavlja:

1. Uvod
2. Osvrt na dosadašnja istraživanja
3. Tehnologije primjenjive u razvoju *web* aplikacija
4. Sigurnosni aspekti *web* temeljenih rješenja

5. Komparativna analiza alata otvorenog koda na primjeru ranjive *web* aplikacije
6. Analiza primjenjivosti alata otvorenog koda
7. Zaključak

U drugom poglavlju napravljen je osvrt na dosadašnja istraživanja iz relevantnog područja. Statistički podaci pomažu pri razumijevanju trendova ranjivosti kod *web* aplikacija, a analize nad alatima otvorenog koda pomažu pri razumijevanju zastupljenosti i korisnosti ovakvih alata u ovom području.

Treće poglavlje analizira relevantne tehnologije koje se primjenjuju u izradi *web* aplikacija. Analiziraju se tehnološke skupine MEAN i MERN, kao vodeće u tom području, i tehnologije koje se primjenjuju unutar tih skupina.

Četvrto poglavlje analizira temeljna načela sigurnosti informacijsko komunikacijskih sustava koja se moraju održavati zbog zaštite istih. Uz to se analiziraju i najčešće ranjivosti i sigurnosni rizici prema OWASP *Top Ten* dokumentu s kojima se susreću web aplikacije.

Peto poglavlje predstavlja praktični dio ovog diplomskog rada. U petom poglavlju analizirana su četiri različita sigurnosna alata za skeniranje ranjivosti. Za potrebe istraživanja alata korištena je namjerno ranjiva aplikacija kako bi se dobili okvirni podaci o skeniranim ranjivostima i zaključile mogućnosti ovih alata.

Šesto poglavlje analizira primjenjivost alata iz petog poglavlja. U ovom poglavlju napravljena je sinteza rezultata dobivenih iz istraživanja alata otvorenog koda. Na temelju funkcionalnosti i mogućnosti istraživanih alata dan je prijedlog o načinima implementacije ovih alata u razvojni proces aplikacije.

Zaključak rada i osvrt na cjelokupni rad dan je u posljednjem poglavlju.

2. Osvrt na dosadašnja istraživanja

Područje razvoja *web* aplikacija je brzorastuće. Brzo se razvijaju nove tehnologije primjenjive u tom području, stoga dolazi i potreba za razvojem sigurnosnog aspekta *web* aplikacija i proučavanjem već postojećih ranjivosti. Zbog toga, važno je pratiti znanstvena istraživanja i radove kako bi se mogli bilježiti trendovi i davati preporuke sigurnosnih rješenja. Osim toga postoje i mnogobrojni alati koji pokušavaju riješiti pitanje sigurnosti *web* aplikacija njihovom primjenom. Činjenica je da bilo koji alat ne može u potpunosti riješiti sigurnosno pitanje *web* aplikacija, stoga se kroz istraživanje može pokušati približiti primjenjivost pojedinih alata kako bi se njihov potencijal mogao maksimalno iskoristiti.

2.1 Podaci o vrstama ranjivosti i njihovoj zastupljenosti

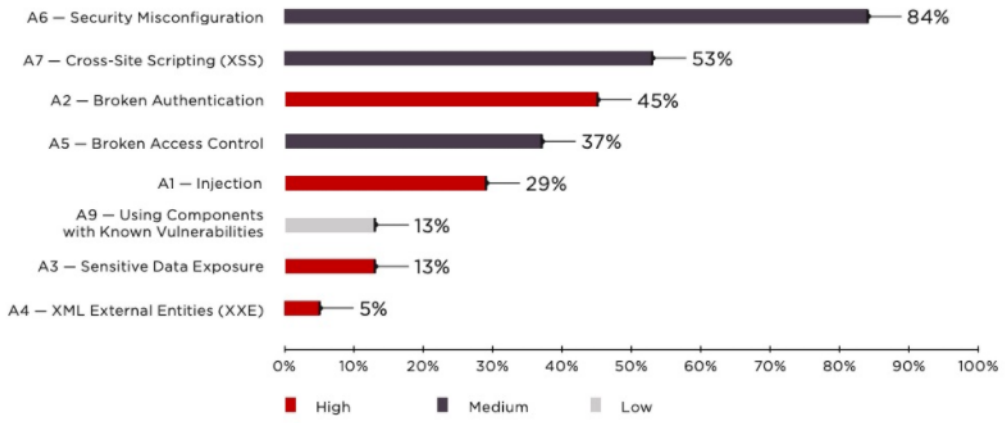
Statistički podaci prikupljeni putem relevantnih istraživanja pokazatelj su zastupljenosti pojedinačnih ranjivosti *web* aplikacija, ali i trendova koji se razvijaju po pitanju tog područja. Samim time, takve podatke se može iskoristiti u svrhu poboljšanja sigurnosti *web* aplikacija, ali i potencijalno za odabir odgovarajućeg alata koji bi se koristio za poboljšanje sigurnosti.

OWASP (engl. *The Open Web Application Security Project*) je neprofitna organizacija koja svoju svrhu pronalazi u poboljšavanju sigurnosti *software*-a. OWASP prikuplja podatke o mrežnoj sigurnosti iz raznih izvora, te je njihov *OWASP Top Ten* dokument jedan od najpopularnijih dokumenta koji djeluje kao pokazatelj najkritičnijih sigurnosnih rizika s kojima se susreću *web* aplikacije. Dokument se nadopunjuje u prosjeku svake treće ili četvrte godine, a zadnje izdanje je iz 2017. godine. Prema tom dokumentu najkritičniji rizici za web aplikacije su [1]:

1. Umetanje (engl. *Injection*)
2. Neispravna autentikacija (engl. *Broken Authentication*)
3. Otkrivanje osjetljivih podataka (engl. *Sensitive Data Exposure*)
4. XML vanjski entiteti (engl. *XML External Entities*)
5. Nesigurna kontrola pristupa (engl. *Broken Access Control*)
6. Pogreške u sigurnosnoj konfiguraciji (engl. *Security Misconfiguration*)

7. XSS (engl. *Cross Side Scripting*)
8. Nesigurna deserijalizacija (engl. *Insecure Deserialization*)
9. Korištenje komponenti s poznatim ranjivostima (engl. *Using Components with Known Vulnerabilities*)
10. Nedovoljno nadziranje i praćenje sustava (engl. *Insufficient Logging & Monitoring*)

Osim ovih čestih sigurnosnih rizika, čak 20% poslovanja i organizacija se susreće često i s napadima uskraćivanja usluge, što je zabilježeno kao trend u istraživanju Agencije Europske unije za kibernetičku sigurnost (ENISA). Kroz ovo istraživanje je ustanovljeno da se u 2019. godini povećao broj napada na *web* aplikacije za 52%, te je ustanovljeno da čak 84% ranjivosti proizlazi iz pogrešaka u sigurnosnoj konfiguraciji. Na slici 1. prikazani su podaci u korelaciji s OWASP *Top Ten* sigurnosnim rizicima [2].



Slika 1. Statistički podaci prema OWASP Top Ten ranjivostima

Izvor:[2]

Zastupljenost napada uskraćivanje usluge potvrđuje i znanstveni članak *Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study* koji mapira podatke znanstvenih članaka s javno dostupnih izvora. U ovom članku ukupno je analizirano 78 znanstvenih članaka koji su prošli finalnu selekciju, te je ustanovljeno da napad uskraćivanjem usluge se javlja u 37% slučajeva [3].

2.2 Istraživanja provedena nad sigurnosnim alatima otvorenog koda

Sigurnosno testiranje *web* aplikacija ili web stranica zahtijeva unaprijed definiranu strategiju i planiranje zbog odabira alata, ali i konstantnih novih izazova u sigurnosnom sektoru u informacijskoj industriji. Stoga i odabir samog alata mora odgovarati krajnjem cilju zbog kojeg se sigurnosno testiranje provodi. Kako bi odabrali alat važno je znati ranjivosti te način rada prosječne web aplikacije.

Web aplikacijama se smatra svaki program koji se pokreće u pregledniku i moguće mu je pristupiti preko Interneta. Web aplikacije pružaju razne funkcionalnosti korisnicima, a zbog razvoja novih tehnologija postaju sve kompleksnije. Aplikacije su obično podijeljene u tri sloja. Prvi sloj je korisnički, koji se može sastojati od preglednika preko kojeg korisnik učitava sadržaj web aplikacije. Zatim drugi sloj čini server, koji poslužuje korisnika i dinamički učitava sadržaj. Treći sloj se sastoji od krajnje strukture *web* aplikacije, odnosno „*backend“*, koji služi za pohranu podataka [4].

Sigurnosni alati koji izvode penetracijsko testiranje su obično definirani kao skeneri. Odnosno smatraju se automatiziranim alatima koji analiziraju *web* aplikacije i *web* stranice, te tako pokušavaju pronaći potencijalne sigurnosne ranjivosti. Osim jednostavnog pretraživanja i skeniranja takvi alati još mogu i pretraživati postoje li pogreške u izvornom kodu, koje bi mogle omogućavati druge vektore napada, kao što je prepunjavanje spremnika. Skeneri web aplikacija obično koriste metodu crne kutije kod penetracijskog testiranja [5].

U radu Baykara, M. *Investigation and Comparison of Web Application Vulnerabilities Test tools* analizirano je šest različitih alata. Alati su se sastojali od komercijalnih i alata otvorenog koda, a za testiranje se koristila ranjiva aplikacija napravljena od strane Netsparker tima na adresi *http://aspnet.testsparker.com*. Testirani su alati *Netsparker, Acunetix, Vega, OWASP ZAP, Wapiti* te *Iron WASP.* Komparativna analiza rada podijeljena je u devet kategorija, koje uključuju vrijeme skeniranja alata, broju skeniranih ranjivosti, mogućnosti izvješća alata, mogućnosti skeniranja po modulu, mogućnosti manualnog skeniranja , dostupnosti grafičkog sučelja, dostupnosti na određenim operacijskim sustavima, cijeni alata te području uporabe [5].

Gledajući isključivo performanse alat *Vega* je imao najbrže vrijeme skeniranja od 45 minuta, dok je alat *Wapiti* imao najdulje vrijeme skeniranja od 13 sati. *IronWASP* alat je otkrio najviše ranjivosti, 213, dok je alat *OWASP ZAP* otkrio tek 7 ranjivosti, najmanje od svih alata. Iako su u ovom istraživanju besplatni alati otvorenog koda postigli najbolje performanse, autor rada ipak sugerira da izbor alata nije tako jednostavan, već da svaki alat ima svoju namjenu [5].

Slična sugestija se nalazi i u radu A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almashfi, „*Web Application Security Tools Analysis*“, gdje su autori rada testirali deset vrsta sigurnosnih alata ovisno o njihovim namjenama. U ovom alatu radu su testirani alati koji su više fokusirani na pojedinačna područja ranjivosti, kao što su nedostaci zaštite u transportnom sloju, ranjivosti curenja informacija, XSS ranjivosti te ranjivosti putem injekcije. Autori ovog rada su zaključili da sigurnosno testiranje *web* aplikacija ili stranica zahtjeva pažljivo osmišljen proces i plan testiranja, te pažljiv odabir alata s obzirom na to da svaki alat ima prednosti i nedostatke [6].

3. Tehnologije primjenjive u razvoju *web* aplikacija

Većina današnjih *web* aplikacija i *web* stranica se dizajnira putem skupine tehnologija. Tehnološke skupine ili popularnije stogovi su skupine alata koje se koriste kako bi se implementirale određene ideje u smislu *web* aplikacija. Stogovi se sastoje od programskih jezika, radnih okruženja, programskih biblioteka i raznih razvojnih alata. Odabir određene tehnološke skupine ovisit će o samoj primjeni i potrebama *web* aplikacije, a to može uključivati njezinu funkcionalnost, skalabilnost, održivost, sigurnost itd. [7].

3.1 Utjecaj arhitekture *web* aplikacija na izbor razvojnih tehnologija

Web aplikacije su aplikacijski programi koji se pokreću na serveru. Korisnici pristupljuju *web* aplikacijama preko raznih preglednika, te šalju zahtjeve serveru, a zatim server, u pozadini, procesira zahtjeve korisnika te odgovara odgovarajućom logikom. Za dizajniranje i izradu takve logike koriste se različiti alati, ali i za prikaz sučelja na korisnikovoj strani. Stoga kako bi mogli razumjeti tehnologije i tehnološke skupine koje se koriste u dizajniranju *web* aplikacija, potrebno je definirati arhitekturu *web* aplikacije.

Arhitekturu *web* aplikacije se može podijeliti na dva dijela, korisnički dio i poslužiteljski dio. Korisnički dio, poznatiji po terminu *frontend*, je odgovaran za prezentiranje podataka korisniku, primanje zahtjeva i podataka od korisnika te usmjeravanje do poslužitelja. Poslužiteljski dio *web* aplikacije, poznatiji po terminu *backend*, je zaslužen za obradu zahtjeva, upravljanje klijentima, pohranu i obradu podataka. Za izradu *frontend*-a i *backend*-a često se koriste različite tehnologije [7].

Za izradu *frontend* sučelja koriste se programski jezici kao što su *HTML/HTML5, CSS, JavaScript*, a uz njih se koriste i radna okruženja i programske biblioteke kao što us *ReactJS, AngularJS, React, Node.js, jQuery*.

Za poslužiteljski, *backend* dio koriste se programski jezici kao što su *C#, Java, PHP, Python, Objective-C* i još mnoštvo drugih programskih jezika, uz njih se koriste radni okviri *web* poslužitelja koji su napravljeni preko programskih jezika kako bi se

omogućila skalabilnost i lakša izrada aplikacija. Primjeri takvih radnih okruženja su *Node.js, .NET , Django* itd. Za pohranu podataka i upravljanje njima koriste se baze podataka kao što su *SQL, MySQL, PostgreSQL, Oracle, MongoDB* itd. Za pokretanje i rad *web* poslužitelja koriste se aplikacije kao što su *Apache* i *Nginx*, a često se koriste *cloud* poslužitelji za dodatne usluge kao što su *AWS, Microsoft Azure, Google Cloud* [7].

Na izbor tehnologije utječe mnogo faktora. Neki od glavnih faktora su veličina projekta i njegova kompleksnost, a uz to će na izbor alata utjecati i ukupni raspoloživi resursi projekta zajedno sa razinom iskustva projektnog tima koji će koristiti te alate. Sama veličina projekta utjecat će na skalabilnost, održivost te brzinu razvoja projekta što se svakako treba uzeti u obzir prilikom izbora alata kako bi se postigli željeni ciljevi. Sigurnost je također jedan od bitnih faktora prilikom izbora tehnologija i alata koji se koriste, stoga je uz ostale faktore jednako važno razmotriti izbor adekvatnog alata za rješavanje pitanja sigurnosti projekta.

Trenutno dva najpopularnija stoga su MERN i MEAN. Ove dvije tehnološke skupine odnosne se na zbirku tehnologija temeljenih na *JavaScript* programskom jeziku. U ispitivanju *StackOverflow*-a čak 68,62% ispitanika koristi ili je koristilo *JavaScript* u profesionalnom okruženju na duži period vremena, što ovaj programski jezik čini najkorištenijim već devetu godinu zaredom, a posebice u *web* razvoju. Za *web* radni okvir servera, izbor u ovim tehnološkim skupinama je *Node.js* (36.19%), a za web razvojna okruženja najčešće su korišteni *React.js* (41,4%)*, jQuery* (34,52%), *Angular* (26,23%) te *Express.js* (23.6%). *MongoDB* (28.03%) je glavni izbor alata za rad s bazama podataka u ovim tehnološkim skupinama [8].

Postoji još mnogo tehnologija i tehnoloških skupina koje se koriste u izradi *web* aplikacija. U sljedeća dva potpoglavlja analizirat će se tehnologije u MEAN i MERN skupinama, koje su jedne od najčešće korištenih tehnologija za razvoj *web* aplikacija.

3.2 MEAN tehnološka skupina

MEAN je skupina tehnologija koje su besplatne i otvorenog koda, bazirane na *JavaScript* programskog jeziku. MEAN služi kao radni okvir za izradu i razvoj *web*

stranica i *web* aplikacija, a glavna prednost korištenja ovakve skupine alata je što korisniku omogućuje rad u jednom programskom jeziku, koristeći već postojeće koncepte programiranja za taj jezik.

Ova tehnološka skupina koristi četiri ključne tehnologije, koje predstavljaju slojeve skupine [9]:

- **M**ongoDB – baza podataka,
- **E**xpress.js – *web* radni okvir,
- **A**ngular.js - radni okvir klijentske strane i
- **N**ode.js – *web* poslužitelj.

3.2.1 MongoDB

MongoDB je baza dokumenata otvorenog koda koja pruža postojanost za podatke aplikacija, dizajnirana je za skalabilnost i lakoću korištenja korisnicima programerima. *MongoDB* popravlja jaz između baza podataka koje koriste ključeve kao primarne vrijednosti spremanja podataka, koji su brzi i skalabilni, te relacijskih baza podataka koje imaju bogatu funkcionalnost. To je vrsta *NoSQL* spremišta baze podataka gdje se podaci pohranjuju u obliku vrijednosti parova ključeva umjesto u mrežu redova i stupaca [10].

Prednosti *MongoDB-a* [10]:

- Nije potrebno kreiranje sheme za bazu podataka,
- jasna struktura objekta,
- jednostavna pridruživanja podataka,
- podržava duboke upite,
- velika skalabilnost i mogućnost dorađivanja,
- automatsko mapiranje aplikacijskih objekata i
- veoma brz pristup podacima.

Nedostaci *MongoDB-a* [10]:

- velika veličina podataka,
- manja fleksibilnost izvođenja upita,

- nema transakcijske podrške i
- sporije mapiranje i agregiranje podataka.

3.2.2 Express.js

Express.js je veoma brzi, minimalistički i fleksibilan radni okvir baziran na *Node.js* tehnologiji. Ovaj okvir pruža robustan set funkcionalnosti za razvoj *web* aplikacija, koji razvojni proces čini jednostavnijim i lakšim u odnosu na korištenje isključivo *Node.js*-a za razvoj web aplikacije. Ovaj alat nudi na raspolaganje brojne *HTTP* metode i posredničke funkcije za lakše spajanje raznih dijelova aplikacije, te pruža tanki sloj osnovnih značajki *web* aplikacija, bez ometanja značajki *Node.js*-a [11].

Prednosti Express.js-a [10]:

- Pojednostavljuje proces razvoja *web* aplikacija u *Node.js*,
- jednostavna konfiguracija,
- omogućava podešavanje usmjeravanja aplikacije preko HTTP metoda i URL adresa,
- posrednički moduli za izvođenje dodatnih rutiranja i primanja zahtjeva,
- podržava postupke za rješavanje pogrešaka i
- jednostavno povezivanje s bazama podataka.

3.2.3 Angular.js

Angular.js je radno okruženje za razvoj *frontend*-a u *web* aplikacijama. Razvijan je u *JavaScript* okviru, od strane *Google*-a 2009. godine, te je sada potpuno besplatan alat otvorenog koda. *Angular.js* u potpunosti se temelji na *HTML* i *JavaScript* programskim jezicima, pa nema potrebe za učenjem druge sintakse ili jezika. *Angular.js* mijenja statički kod napisan u *HTML-u* u dinamički *HTML* kod. Proširuje mogućnosti klasičnih statičkih *web* stranica dodavanjem ugrađenih atributa i komponenti te tako omogućuje i stvaranje novih atributa i komponenti kombinirajući *JavaScript* i *HTML* kod. Kombiniranjem više komponenti *Angular.js* omogućuje razvoj veoma kompleksnih *web* aplikacija. Glavna namjena mu je razvoj aplikacija koje se

sastoje od jedne stranice (engl. *Single-page Application*), razvijane po MVC (engl. *Model-View-Controller*) arhitekturi koja pomaže u uređivanju i organiziranju *web* aplikacije [12].

Prednosti *Angular.js-a* [12]:

- Dinamičko povezivanje podataka: Omogućuje automatsku sinkronizaciju podataka između prikaza ('V' u MVC -u) i komponente modela ('M' u MVC -u) u arhitekturi *web* aplikacije.
- MVC arhitektura – omogućuje skalabilnost i mogućnost proširenja projekta,
- Podrška za lokalizaciju,
- Ubrizgavanje ovisnosti za klase,
- Brzo povezivanje sa serverom i preuzimanje podataka i
- Direktive omogućuju da programeri dodijele posebna ponašanja objektnim modelima.

Nedostaci *Angular.js-a* [12]:

- Različite tehnike kodiranja kompliciraju integraciju različitih komponenti,
- Veoma složen životni ciklus,
- Loša skalabilnost,
- Problemi s kašnjenjem korisničkog sučelja pri učitavanju više korisnika,
- Nekompatibilnost sa prijašnjim verzijama i
- Kompleksnost *Angular.js-a* otežava svladavanje radnog okvira u cjelini.

3.2.4 Node.js

Node.js je *JavaScript runtime* okruženje koje pokreće *backend* aplikaciju (putem *Expressa*). Temelji se na *Googleovom* V8 radnom okviru, te tako omogućuje izvršavanje *JavaScript* koda izvan preglednika. *Node.js* je asinkroni mehanizam pogonjen događajima unutar *web* aplikacije, što omogućuje kada korisnik pošalje zahtjev aplikaciji nastavak rada na drugim rutinama, umjesto čekanja na odgovor. Uz to *Node.js* omogućuje pokretanje *HTTP* aplikacija napisanih u *JavaScriptu,* te tako ispunjava ulogu *web* servera. Zahvaljujući *Node.js-u web* aplikacije izvršavaju *JavaScript* kod unutar preglednika davajući korisniku bogatije korisničko iskustvo, dok

web server zasebno pokreće drugi *JavaScript* kod kako bih obradio zahtjeve klijenta [13].

Prednosti *Node.js-a* [10]:

- Veoma brzo procesiranje i odgovaranje na događaje,
- Otvorenog je koda i stalno se proširuje,
- Podržava razvoj aplikacija u stvarnom vremenu i
- Skalabilnost u pogledu na razvoj mikrousluga,

Nedostaci *Node.js-a* [10]*:*

- Loše performanse za intenzivne upute koje koriste procesorsku moć,
- Povratni odgovori mogu uzrokovati veliki broj ugniježđenih petlji,
- Loše performanse s relacijskim bazama podataka i
- Loše performanse s velikim aplikacijama.

3.3 MERN tehnološka skupina

MERN je jedna od varijacija MEAN skupine, gdje se *Angular.js* zamjenjuje sa *React.js* radnim okvirom. MERN je također baziran na *JavaScript* programskom jeziku, no njegov cilj je omogućiti lakši i kraći razvojni period *web* aplikacija zahvaljujući *React.js* alatu.

Ova tehnološka skupina sadrži sljedeća četiri alata [14]:

- **M**ongoDB – baza podataka,
- **E**xpress.js – *web* radni okvir,
- **R**eact.js - radni okvir klijentske strane i
- **N**ode.js – *web* poslužitelj.

3.3.1 React.js

React.js je *JavaScript* biblioteka za izgradnju interaktivnih korisničkih sučelja na klijentskoj strani *web* aplikacije. *React.js* omogućuje razvoj velikih aplikacija koje podržavaju mogućnost učitavanja novih podataka bez osvježavanja stranice

omogućujući brzinu aplikacije i pružajući bolje korisničko iskustvo. Za razliku od *Angular.js-a* koji može svojim kodom cjelovito povezivati MVC arhitekturu, *React.js* je *rendering* biblioteka koja se povezuje isključivo na prikaze ('V' u MVC-u) [10].

React.js je veoma moćan alat za stvaranje kompleksnih korisničkih sučelja. To mu omogućuju komponente unutar *React.js* koda zajedno sa korištenjem virtualnog DOM-a (engl. *Document Object Model*). Komponente su entiteti, koji se koriste za kreiranje elemenata korisničkog sučelja te se one mogu ponovno iskorištavati u različitim prikazima. Uz to *React.js* stvara predmemoriju strukture podataka koja izračunava izvršene promjene u odnosu na prethodnu memoriju, a zatim ažurira preglednik. To dopušta korisnicima ovog alata da stvaraju korisnička sučelja u pokretu, dok *React.js* ne ažurira sve komponente, već samo novonastale komponente. To ga čini jako fleksibilnim, učinkovitim i brzim alatom za stvaranje kompleksnih sučelja [15].

Prednosti *React.js-a* [10]*:*

- Bolje performanse u pogledu brzine, učinkovitosti i fleksibilnosti,
- Virtualna DOM podrška,
- Jednosmjerni tok podataka,
- *JSX* podrška,
- DOM omogućuje bolje performanse i
- Krivulja učenja je jednostavnija od *Angular.js-a* za svladavanje alata.

Nedostaci *React.js-a* [10]:

- Pokriva samo prikaze u MVC arhitekturi,
- Potrebni su dodatni alati i vanjska podrška za stvaranje funkcionalne *web* aplikacije.

3.3.2 Usporedba MERN i MEAN tehnološke skupine

MERN i MEAN tehnološke skupine koriste identične tehnologije za *backend* usluge, no razlikuju se pri korištenju tehnologije za izgradnju korisničkog sučelja tj. *frontend*-a. Intuitivno, može se reći da se izbor između ove dvije skupine jednostavno svodi na izbor između *React.js-a* i *Angular.js-a*, no kako su ovo dva različita alata po

načinu rada i funkcionalnosti, treba također razmotriti i druge bitne faktore kako bi se maksimizirao potencijal korisnosti ovih tehnologija.

Angular.js i *React.js* se koriste u istu svrhu, za dizajniranje korisničkog sučelja u *frontend*-u. *Angular.js* je *JavaScript* radni okvir, dok je *React.js JavaScript* biblioteka. Radni okvir je način strukturiranog prikazivanja koda, što se odražava i u samom *Angular.js*-u koji MVC arhitekturu u potpunosti povezuje i na *frontend* sučelju, dok *React.js* radi samo s prikazima povezujući se na ostatak MVC arhitekture pomoću virtualnog DOM-a [10].

- Performanse

React.js i *Angular.js* se razlikuju u načinu usmjeravanja podataka. *Angular.js* koristi dvosmjerni tok podataka kako bi sinkronizirao sloj prikaza i podatkovni sloj u *backend*-u. To donosi više mogućnosti u odnosu na jednosmjerni tok podataka koji koristi *React.js,* ali upravo zbog jednosmjernog toka podataka s kombinacijom otkrivanja stanja uz pomoć virtualnog DOM-a *React.js* se pokazuje kao brži alat kada je u pitanju *frontend* dio aplikacije [10].

- Arhitektura

Kao što je već spomenuto *Angular.js* je cjelokupni radni okvir, dok je *React.js* programska biblioteka. Kao okvir, *Angular.js* provodi MVC dizajn, prisiljavajući programere da bolje organiziraju svoj kod, a dobro organiziran kod omogućuje veću skalabilnost aplikacija i lakše održavanje. *React.js* je fleksibilniji po pitanju dizajna aplikacije, što omogućuje brži razvoj korisničkog sučelja, stoga je na programerima da organiziraju svoj kod, što u nekim slučajevima može ograničiti skalabilnost aplikacije i učiniti kod puno težim za održavanje [10].

- Programske biblioteke treće strane i podrška

React.js je sam po sebi biblioteka, te zahtjeva dodatnu podršku biblioteka treće strane kako bi omogućio pravilno funkcioniranje aplikacija. *React.js* nije u mogućnosti samostalno usmjeravati zahtjeve, te nema samostalnih rješenja za pozive na pozadinske poslužitelje, stoga su potrebne dodatne konfiguracije i podrška. *Angular.js* više djeluje kao samostalan alat koji ima već ugrađenu podrške za takve radnje [10].

- Sigurnost

Sigurnost *web* aplikacija najviše ovisi o programerima koji koriste tehnologije za izradu takvih aplikacija. Važno je pratiti trendove i otkrivene ranjivosti u alatima, te prilikom rada s istima primjenjivati u praksi rješenja za već poznate ranjivosti. Ova dva alata su različita u načinu funkcioniranja, stoga imaju i različite otkrivene ranjivosti, ali prema istraživanju *StackOverflow-a React.js* se više preferira od *Angular.js-a.* Neki od razloga su jer *React.js* pruža bolju sigurnost za osjetljive podatke, a i sam kod na klijentskoj strani se može sakriti od korisnika, ograničavajući mogućnosti pronalaženja ranjivosti u samom kodu [10], [8].

3.4 Odabir tehnologije za izradu *web* aplikacije

U prethodnim poglavljima analizirane su neke od najpopularnijih i najkorištenijih tehnoloških skupina i tehnologija. Iz toga se može zaključiti da svaka tehnologija ima svoje prednosti i mane, stoga ne postoji jedna ili više tehnologija koje bi se smatrale najboljim odabirom.

Pri izboru tehnologije bitno je razmotriti i ostale bitne faktore koje utječu na adekvatan izbor tehnoloških alata koji će se koristiti za izradu *web* aplikacije. Jedan od tih faktora je veličina projekta. Pa tako za male projekte neke tehnologije će odgovarati bolje od kompleksnih alata kao što je *Angular.js,* koji potencijalno može dodavati nepotrebnu kompleksnost i produljenje vremenskog roka za pokretanje projekta u radno stanje. Ako se pak radi o velikom projektu, koji će rukovati s velikom količinom podataka, tada *Angular.js* je možda i najbolji izbor, uz pomoć dodatnih tehnoloških alata. Osim veličine projekta, u izboru alata je bitan i projektni tim programera koji će se koristiti njima. Bitno je odabrati alate s kojima su programeri upoznati, te posjeduju određenu razinu stručnosti u radu s njima. U suprotnom, riskiranjem odabira alata koji nisu adekvatni programerima, postoje nepogodne mogućnosti kao što su kašnjenje projekta, nefunkcionalne aplikacije, ali i moguće ranjivosti u *web* aplikacijama koje su mogle biti izbjegnute [10].

Kada je u pitanju sama sigurnost *web* aplikacija i tehnologija koje će se koristiti za izradu istih, ne postoji savršen alat. Prije izrade *web* aplikacije, ali i tijekom, bitno je obaviti istraživanje o potencijalnim ranjivostima tehnologija koje će se koristiti, te načinima kako ih spriječiti. Jedan od načina poboljšavanja sigurnosti aplikacija jest korištenje dodatnih tehnologija i alata za pronalaženje ranjivosti. O takvim alatima će se više pisati u petom i šestom poglavlju ovog rada.

4. Sigurnosni aspekti web temeljenih rješenja

Postoji puno različitih definicija za sigurnost informacijskog sustava. Sigurnost se može definirati kao sposobnost ili mogućnost biti siguran, odnosno oslobođen od opasnosti. Jedna od definicija informacijsko komunikacijske sigurnosti je:

„*Informacijska sigurnost je stanje povjerljivosti, cjelovitosti i raspoloživosti podataka, koje se postiže primjenom propisanih mjera i standarda informacijske sigurnosti te organizacijskom podrškom za poslove planiranja, provedbe, provjere i dorade mjera i standarda*.“ – Zakon o informacijskog sigurnosti Republike Hrvatske (NN79/07) [16].

4.1 Temeljna načela sigurnosti

Načela sigurnosti informacijskog sustava je bitno održavati zbog zaštite informacijsko komunikacijskog sustava. To podrazumijeva primjenu niza mjera, standarda i postupaka s ciljem održavanja željene razine sigurnosti informacijsko komunikacijskog sustava [16].

Postoji šest temeljnih načela sigurnosti prema [16]:

Povjerljivost je osobina informacijskog sustava koja osigurava otkrivanje informacija i podataka isključivo autoriziranim osobama, entitetima i procesima, u definirano vrijeme i definiranom procedurom.

Cjelovitost sustava i informacija podrazumijeva zaštitu od namjerne ili slučajne neovlaštene modifikacije uzrokovane ljudskim utjecajem ili pogreške u radu sustava.

Dostupnost se odnosi na raspoloživost informacije ovlaštenim korisnicima u traženom trenutku i prema zadanim uvjetima.

Autentikacija je provjera legitimiteta korisnika koji zahtjeva pristup određenim resursima informacijskog sustava.

Autorizacija korisnika se odnosi na dodjeljivanje određene razine prava pristupa sustavu nakon potvrde autentičnosti korisnika.

Revizija je proces evaluacije učinkovitosti provedenih sigurnosnih mehanizama.

4.2 Ranjivosti *web* aplikacija

Prijetnja sigurnosti *web* aplikacijama je svaka moguća pojava, zlonamjerna ili ne, koja bi mogla naštetiti *web* aplikaciji na neki način. Ranjivost je slabost koja prijetnju čini mogućom. Ranjivosti mogu uzrokovati loš dizajn, greška u konfiguraciji ili nesigurne i neprovjerene tehnike kodiranja. Napad je radnja koja iskorištava ranjivosti u *web* aplikaciji.

Kao što je već u prethodnim poglavljima objašnjeno *web* aplikacije se izrađuju korištenjem skupinom tehnologija. *Web* aplikacije najčešće sadrže bazu podataka, *web* poslužitelj koji odgovara na zahtjeve i *web* preglednik koji nudi korisničko sučelje putem kojeg korisnik može uputiti zahtjeve. Svaku od tih komponenti moguće je ugroziti ako *web* aplikacija nije pravilno dizajnirana. U sljedećim potpoglavljima analizirat će se najčešće ranjivosti s kojima se *web* aplikacije susreću.

4.2.1 Umetanje

Ranjivosti umetanjem su prisutne u svim situacijama u kojima napadač može poslati zlonamjerne podatke prevoditelju programskog jezika ili poslužitelju. Gotovo svaki izvor podataka može biti vektor umetanja ukoliko ne postoji pravilna zaštita koja nastoji spriječiti takve ranjivosti. Ranjivosti umetanja se često nalaze u *SQL, LDAP, Xpath* ili *NoSQL* upitima za baze podataka, *OS* naredbama, *XML* prevoditeljima, *STMP* zaglavljima, programskim jezicima za izraze itd [17].

Umetanje može rezultirati gubitkom podataka, korupcijom ili otkrivanjem osjetljivih podataka neovlaštenim stranama, gubitkom odgovornosti ili uskraćivanjem pristupa *web* aplikaciji. Umetanje zloćudnog koda ponekad može dovesti i do potpunog preuzimanja poslužitelja *web* aplikacije. Aplikacija je osjetljiva na napade ako korisnički podaci koji se šalju u zahtjevu nisu provjereni, filtrirani ili dezinficirani od strane aplikacije [17].

Jednostavan primjer iskorištavanja ove ranjivosti je zaobilaženje legitimne prijave u sustav ako ne postoji provjera podataka koje je korisnik unio. Prilikom prijave, korisnik treba upisati korisničko ime i lozinku u dva polja za upis podataka, program

zatim dohvaća podatke iz polja kao varijable i kreira *SQL* upit za dohvat podataka iz baze podataka. Primjer *SQL* upita koji se formira je sljedeći:

Upit = "SELECT KorisnickoIme FROM Korisnici WHERE KorisnickoIme = '" + user + "' AND Lozinka = '" + pass + ""

Na primjeru ovog upita možemo vidjeti da *user* i *pass* su varijable koje će program koristiti za dohvat podataka u bazi. Ako ne postoji zaštita ili provjera podataka koje korisnik unese, tada korisnik može umetnuti dio malicioznog koda koji će onda moći dohvatiti podatke iz baze podataka koje korisnik ne bi smio dohvatiti. Primjer malicioznog koda koji se može umetnuti je:

'OR '1' = '1 ,

kojeg kada korisnik unese u dva polja rezultira sljedećim upitom:

SELECT KorisnickoIme FROM Korisnici WHERE KorisnickoIme = '' OR '1'='1' AND Lozinka = '' OR '1'='1'

Ovaj upit omogućava korisniku da baza podataka više ne uspoređuje podatke u tablici s korisničkim imenom, već kod nalaže programu da provjerava istinitost tvrdnje *'OR '1' ='1.* Ova tvrdnja je uvijek točna, stoga će ova radnja rezultirati da program vrati podatke iz baze podataka koji su u prvom redu u tablici *Korisnici*, čime će se napadač uspješno prijaviti u sustav pod tuđim legitimnim korisničkim računom [18].

4.2.2 Neispravna autentikacija

Aplikacijske funkcije povezane s provjerom autentičnosti i upravljanjem sesijom ukoliko se pogrešno provode, mogu dopustiti napadačima da ugroze lozinke, ključeve ili tokene sesija ili iskoriste druge nedostatke implementacije kako bi privremeno ili trajno preuzeli identitete drugih korisnika [19].

Napadači imaju pristup stotinama milijuna valjanih kombinacija korisničkih imena i lozinke za popunjavanje vjerodajnica, popise administrativnih računa, alate za automatiziranu grubu silu i za napad pomoću rječnika. Aplikacije su izložene ranjivostima ako dopuštaju pokušaje prijave u sustav bez ograničenja, dozvoljavaju

korištenje slabih lozinki i spremaju podatke o lozinkama na neispravan način, implementiraju neefikasne procese za oporavak vjerodajnica itd [19].

Najjednostavniji primjer iskorištavanja ranjivosti neispravne autentikacije je nedostatak implementacije automatizirane zaštite od punjenja vjerodajnica, gdje se napadačima omogućava automatizirani napad korištenjem rječnika lozinki i korisničkih imena. Jedan od načina zaštite je implementirana funkcija prekinute autentifikacije, gdje se korisnicima omogućava određeni broj pokušaja upisivanja vjerodajnica prije nego se postavi zaštita na upisivanje samih. Takav način zaštite značajno otežava napore automatiziranih napada [19].

4.2.3 Otkrivanje osjetljivih podataka

Mnoge *web* aplikacije ne štite ispravno osjetljive podatke korisnika. Napadači mogu ukrasti ili manipulirati slabo zaštićenim korisničkim podacima kako bi izveli nepoćudne radnje koristeći ukradene podatke. Najčešća greška koja uzrokuje ranjivost u *web* aplikacijama je nekriptiranje osjetljivih podataka ili pak korištenje slabe enkripcije. Korištenje slabih ključeva i algoritama za kriptiranje podataka ne osigurava adekvatnu zaštitu osjetljivih podataka [20].

Napadači se često koriste i ručnim tehnikama napada kako bi presreli podatke između korisnika i poslužitelja. Slabosti na strani poslužitelja za podatke u prijenosu se uglavnom lako otkrivaju, ali teško za podatke koji su u mirovanju. Enkripcija je nužna za podatke, no ukoliko napadač uspije ukrasti takve podatke, tada je u mogućnosti i dekriptirati ih, stoga je vrlo važno imati veoma dobre enkripcijske tehnike koje će otežati napore napadača [20].

Za minimiziranje ovakve vrste ranjivosti, potrebno je klasificirati podatke koje aplikacija obrađuje, pohranjuje i prenosi. Potrebno je odrediti koji su podatci osjetljivi u skladu sa zakonima o privatnosti, regulatornim zahtjevima ili poslovnim potrebama, te uz to primijeniti prikladnu kontrolu i zaštitu podataka. Šifriranje i enkripcija osjetljivih podataka je nužna, te je potrebno primijeniti jake i ažurirane algoritme, protokole i ključeve. Uz to dobro je i odrediti kakvi podatci se pohranjuju, te tako izbjeći nepotrebno pohranjivanje osjetljivih podataka [20].

4.2.4 XML vanjski entiteti

Mnogi stariji ili loše konfigurirani *XML* procesori podataka procjenjuju reference vanjskih entiteta unutar *XML* dokumenata. Vanjski entiteti mogu se koristiti za otkrivanje unutarnjih datoteka pomoću rukovatelja *URI* datoteka, unutarnjih dijeljenja datoteka, unutarnjeg skeniranja portova, udaljenog izvršenja koda i napada uskraćivanja usluge. Napadači iskorištavaju ove ranjivosti ako su u mogućnosti učitati zlоćudnu *XML* datoteku ili pak uključiti maliciozni sadržaj u *XML* dokument, iskorištavajući ranjivost koda [21].

Najjednostavniji način iskorištavanja ove ranjivosti jest upravo učitavanje zlonamjerne *XML* datoteke. Ako je datoteka prihvaćena posljedice toga mogu biti otkrivanje podataka, pristup mrežnim informacijama, uskraćivanje usluge itd. Sljedeći primjer malicioznog koda u *XML* datoteci predstavlja napadačev pokušaj da izvuče podatke s poslužitelja:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Pomoću ovog koda napadač može pokušati izvući osjetljive podatke s poslužitelja, a četvrta linija koda pokušava narediti poslužitelju odakle da izvuče podatke iskorištavajući ranjivosti loše konfiguriranog *XML* procesora [21].

Ako napadač želi pokušati izvršiti napad uskraćivanja usluga, onda će zamijeniti četvrtu liniju koda tako da uključi datoteku potencijalno beskonačne veličine. Primjer toga koda je sljedeći [21]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
<foo>&xxe;</foo>
```

Sprečavanje ove vrste ranjivosti je u odgovornosti programera. Na njemu je da ažurira loše *XML* procesore i biblioteke koje koristi *web* aplikacija. Preporučljivo je

provjeravati podatke koji se unose na strani poslužitelja kako bi se izbjegli maliciozni kodovi. Ako je moguće preporučuje se koristiti i druge formate podataka koji su manje složeni, kao što je primjerice *JSON*. U slučaju da neke od ovih opcija nisu moguće programeri mogu razmisliti o uporabi API sigurnosnih pristupnika ili vatrozida za web aplikacije za otkrivanje, nadziranje i blokiranje ovakvih vrsta napada [21].

4.2.5 Nesigurna kontrola pristupa

Ograničenja u pogledu toga što je dopušteno autentificiranim korisnicima često se ne primjenjuju pravilno. Napadači mogu iskoristiti ranjivosti loše implementirane kontrole pristupa za pristup neovlaštenim funkcijama i/ili podacima, kao što su pristup računima drugih korisnika, pregled osjetljivih datoteka, izmjena podataka drugih korisnika, promjena prava pristupa itd. Ranjivosti kontrole pristupa su česte zbog nedostatke automatiziranog otkrivanja i nedostatka učinkovitog funkcionalnog testiranja od strane programera [22].

Uobičajene ranjivosti kontrole pristupa uključuju zaobilaženje provjere kontrole pristupa izmjenom *URL-a*, stanja unutar aplikacije ili *HTML* stranice, dopuštanje promjene primarnog ključa u zapis drugih korisnika, omogućavajući napadaču da pregledava ili uređuje tuđe podatke. Česta ranjivost je i mogućnost povišenja privilegija, ova akcija napadaču omogućuje da djeluje kao korisnik ili u nekim slučajevima administrator, iako nije prijavljen. Na sljedećem primjeru se može vidit kako jednostavnom izmjenom *URL-a* napadač može pristupiti administratorskoj stranici, ukoliko zaista izmjenom *URL-a* pristupi administratorskoj stranici tada postoji ozbiljna ranjivost [22].

Prvobitni *URL: http://example.com/app/getappInfo*

Izmijenjeni *URL: http://example.com/app/admin_getappInfo*

Kontrola pristupa učinkovita je samo ako se provodi u pouzdanom kodu na strani poslužitelja ili API-ju bez poslužitelja, gdje napadač ne može izmijeniti provjeru kontrole pristupa ili metapodatke. Stoga je potrebno dizajnirati i implementirati efikasne mehanizme kontrole pristupa i koristiti ih u cijeloj aplikaciji [22].

4.2.6 Pogreške u sigurnosnoj konfiguraciji

Pogreške u sigurnosnoj konfiguraciji su jedne od najčešćih ranjivosti. To obično je rezultat nesigurnih zadanih konfiguracija, nepotpunih ili *ad hoc* konfiguracija, otvorene pohrane u oblaku, opsežnih poruka o pogrešci koje sadrže osjetljive podatke ili pogrešno konfiguriranih *HTTP* zaglavlja. Pogrešna konfiguracija može se dogoditi na bilo kojem sloju web aplikacije. Napadači će često pokušati iskoristiti ranjivosti koje mogu nastati sigurnosnim propustima u konfiguraciji. Pokušat će iskoristiti nezakrpane ranjivosti ili nedostatke, pristupiti zadanim računima, neiskorištenim stranicama, nezaštićenim datotekama ili direktorijima kako bi stekli neovlašteni pristup [23].

Kako bi se spriječili sigurnosni propusti u konfiguraciji potrebno je razmotriti sustav kao cjelinu i ustvrditi da li su svi dijelovi sustava pravilno sigurnosno konfigurirani. To uključuje operacijske sustave, radne okvire, programske biblioteke, aplikacije itd. Svi dijelovi sustava moraju biti zakrpani i ako je moguće nadograđeni, a ukoliko postoje funkcionalnosti ili komponente koje se ne koriste ili su nepotrebne potrebno ih je ukloniti [23].

4.2.7 XSS

XSS ranjivosti nastaju kada *web* aplikacija uključi nepouzdane podatke u novu *web* stranicu bez valjane provjere, ili pak kada ažurira postojeću *web* stranicu s podacima koji su dostavili korisnici pomoću API-ja preglednika koji može stvoriti *HTML* ili *JavaScript* kod*.* XSS dopušta napadačima izvršavanje skripti u pregledniku žrtve koje mogu oteti korisničke sesije, uništiti *web* stranice ili preusmjeriti korisnika na zlonamjerna mjesta. Jednom kada napadač umetne maliciozni kod na *web* aplikaciju, ta *web* aplikacija djeluje kao posrednik, odnosno prenositelj malicioznog koda do žrtve [24].

XSS napadi se klasificiraju u tri klase prema [24]:

- Pohranjeni (engl. *Stored*) XSS napadi,
- Reflektirani (engl. *Reflected*) XSS napadi,
- DOM (engl. *Document-Object Model)* XSS napadi.

Tipični XSS napadi uključuju krađu sesija, preuzimanje računa, zaobilaženje višefaktorske autentifikacije, zamjenu ili izobličenje DOM čvora, napade na preglednike, bilježenje lozinki i ključeva itd. Za sprječavanje XSS napada potrebno je odvajanje nepouzdanih podataka od aktivnog sadržaja preglednika. To se može postići korištenjem radnih okvira koji po dizajnu automatski izlaze iz domene XSS napada, kao što je naprimjer *React.js* ili *Ruby on Rails.* Osim toga mogu se koristiti tehnike filtriranja podataka koji pristižu od strane korisnika, te enkodiranja podataka na izlazu. Omogućavanje politike sigurnosti sadržaja (engl. *Content Policy Security – CSP)* kao dubinske obrane za ublažavanje kontrole protiv XSS je učinkovito ako ne postoje druge ranjivosti koje bi omogućile postavljanje zlonamjernog koda putem lokalnih datoteka [24].

4.2.8 Nesigurna deserijalizacija

Nesigurna deserijalizacija često dovodi do udaljenog izvođenja koda. Čak i ako nedostaci deserijalizacije ne rezultiraju udaljenim izvršavanjem koda, mogu se koristiti za izvođenje napada, uključujući napade ponovne reprodukcije, napade umetanjem i napade eskalacije privilegija. *Web* aplikacije će biti ranjive na ovakve napade ako dopuštaju deserijalizaciju podataka ili objekata, a napadi se izvršavaju tako da se deserijaliziraju maliciozno promijenjeni objekti koje dostavlja napadač [25].

Primjer takvog napada može biti trovanje kolačića. Na primjeru koda ispod možemo primijetiti sadržaj jednog kolačića koji koristi serijalizaciju podataka za pohranu.

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Podatak „*user“* sadržava ulogu vlasnika tog kolačića. Napadač tada može izmijeniti serijski objekt tj. kolačić kako bi sebi dao administratorske privilegije, kao što je to prikazano u sljedećoj liniji koda [25]:

a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Jedino sigurno rješenje za ovaj tip ranjivosti jest ne prihvaćanje serijaliziranih objekata iz nepouzdanih izvora ili korištenje medija za serijalizaciju koji dopuštaju samo primitivne tipove podataka. Ostala prihvatljiva rješenja mogu biti provođenje digitalnih potpisa nad serijaliziranim objektima, provođenje ograničenja nad tipovima podataka tijekom deserijalizacije, izoliranje koda koji se deserijalizira itd. [25].

4.2.9 Korištenje komponenti s poznatim ranjivostima

Komponente koje sadrži *web* aplikacija kao što su programske biblioteke, radni okviri i drugi softverski moduli, izvode se s istim ovlastima kao i aplikacija. Ako se iskoristi ranjiva komponenta, takav napad može uzrokovati ozbiljan gubitak podataka ili preuzimanje poslužitelja. *Web* aplikacije i API-ji povezani na aplikacije koji koriste komponente s poznatim ranjivostima mogu narušiti sigurnost aplikacija i omogućiti različite napade i utjecaje na rad aplikacije [26].

Za sprječavanje ovakve vrste ranjivosti organizacijama se preporučuje osmisliti proces koji bi obuhvaćao praćenje, testiranje i primjenu zakrpa ili ažuriranja konfiguracije svih komponenti koje se koriste u *web* aplikacijama. Takav proces upravljanja treba pratiti *web* aplikaciju kroz cijeli životni ciklus kako bi se izbjegle moguće ranjivosti. Uz to bitno je uklanjati komponente ili alate koji su nepotrebni ili se pak ne koriste, a sve korištene komponente treba preuzimati sa službenih izvora putem sigurnih veza [26].

4.2.10 Nedovoljno nadziranje i praćenje sustava

Većina uspješnih napada počinje probiranjem ranjivosti na *web* aplikacijama. Dopuštanje probiranja ranjivosti gotovo uvijek vodi do uspješnog iskorištavanja ranjivosti što rezultira napadom na *web* aplikaciju. Napadači stoga se često oslanjanju na nedostatak nadzora i praćenja sustava, gotovo uvijek napad se izvrši bez pravovremenih otkrića malicioznih aktivnosti. Aplikacije su ranjive ako nemaju implementirane adekvatne funkcije bilježenja, otkrivanja i praćenja aktivnosti. Prilikom implementacije takvih funkcija, treba paziti da se upozorenja i greške generiraju jasno i da se zapisuju u dnevnike koji nisu pohranjeni samo lokalno. Gotovo sve maliciozne

aktivnosti se ne mogu upozoriti ni otkriti u stvarnom vremenu, stoga je bitno obratiti pozornost na već zabilježene sumnjive aktivnosti kako ne bi došlo do neželjenih napada. U slučaju već izvršenog napada, ili napada u tijeku važno je imati i planove za aktivni odgovor na takve situacije, kako bi se što je više moguće minimizirao njihov utjecaj. Jedan od načina poboljšavanja sustava nadziranja i praćenja jest obaviti penetracijsko testiranje nad *web* aplikacijom. Jednom kada je testiranje obavljeno, tada se mogu pregledati dnevnici i utvrditi koliko efikasno prate aktivnosti u sustavu [27].

5. Komparativna analiza alata otvorenog koda na primjeru ranjive *web* aplikacije

U prethodnom poglavlju analizirane su ranjivosti u *web* aplikacijama, a u nastavku ovog poglavlja koristit će se besplatni alati otvorenog koda kako bi se pronašle ranjivosti u ranjivoj *web* aplikaciji. Alati koji će se koristiti za analizu primjenjivosti su sigurnosni skeneri za dinamičko testiranje sigurnosti *web* aplikacija. Alati koji će se analizirati su *OWASP ZAP, Vega, Arachni* i *Nikto*. Odabir ovih alata temeljio se na specifičnim funkcionalnostima i načinu primjene alata.

5.1 Skeneri ranjivosti

Skeneri ranjivosti su automatizirani alati koji se koriste za otkrivanje ranjivosti unutar računala, mreža ili aplikacija. Ovisno o vrsti alata, skeneri mogu imati različite funkcionalnosti i različite tehnike za otkrivanje ranjivosti. U nastavku ovog potpoglavlja analizirat će se neke od ključnih značajki.

5.1.1 Tehnike skeniranja ranjivosti

Glavna podjela sigurnosnog testiranja aplikacija se dijeli na statičku i dinamičku analizu. Statičko sigurnosno testiranje aplikacija (SAST – engl. *Static Application Security Testing)* često koristi razne vrste tehnika statičke analize za otkrivanje ranjivosti, dok dinamičko testiranje sigurnosti aplikacija (DAST – engl. *Dynamic Application Security Testing)* koristi tehnike implementiranja grafa napada.

- Statička analiza

Statička analiza je brza i pouzdana tehnika. Ova tehnika se fokusira na analizu programske strukture unutar aplikacije, prvenstveno na sam programski kod kako bi se otkrile moguće ranjivosti unutar aplikacije. Ova tehnika se smatra jako efikasnom za otkrivanje ranjivosti. Statička analiza često koristi programske biblioteke ili baze podataka za usporedbu s analiziranim kodom kako bi verificirali programski kod. Jedan od nedostataka ovakve analize je taj što u slučaju otkrivanje nove nepoznate

ranjivosti nije moguće napraviti nikakvu usporedbu s programskom bibliotekom kako bi verificirali sigurnost programskog koda [28].

- Analiza grafa napada

Graf napada definira se kao sažeti prikaz svih putova koje napadač prati u mreži kako bi postigao željeno stanje. Željeno stanje može uključivati oštećenje mreže, krađu mrežnih paketa ili potpuni pristup nad njom kako bi se utvrdilo što se događa u mreži. Grafovi napada pomažu u utvrđivanju sigurnosnih slabosti koje leže unutar aplikacije, obično su dosta veliki jer predstavljaju cijelu aplikaciju, stoga su poprilično složeni za razumijevanju i analizu. Generiranje grafa napada implementira se unutar skenera ranjivosti kako bi se identificirale temeljne ranjivosti aplikacije, a potom ustanovili cjelokupni grafovi napada za analizu jačine pojedinog napada.

5.1.2 Funkcionalnosti skenera ranjivosti

Procjena ranjivosti znači identificiranje ranjivosti u sustavu prije nego što ih može iskoristiti bilo tko čije namjere su zloćudne. Ovo je proaktivan pristup sigurnosti gdje se ranjivosti otkrivaju i rješavaju kako ih ne bi nitko zlonamjeran iskoristio. Ranjivosti ne proizlaze samo iz aplikacije, ranjive mogu biti i platforme na kojoj je aplikacija, operacijski sustavi, međuoprema koja povezuje razne dijelove sustava aplikacije itd. Stoga je potrebno skenirati cjelokupni sustav uključujući mrežu i softvere koji aplikacija koristi [28].

S obzirom na vrstu skenera, mogu se podijeliti na tri sljedeće kategorije [28]:

- **Skeneri portova** - skeneri portova koriste se za skeniranje portova radi utvrđivanja otvorenih i zatvorenih portova, operacijskog sustava, ponuđenih usluga.
- **Skeneri aplikacija** - Skeneri aplikacija koriste se za procjenu određene aplikacije na mreži kako bi se pratile njezine slabosti koje se mogu dalje koristi se za izazivanje rizika za sustav.
- **Skeneri ranjivosti** - Skeneri ranjivosti otkrivaju ranjivosti u sustavu koje, ako im pristupi zlonamjeran korisnik ili haker, mogu ugroziti cijeli mrežni sustav.

Skeneri su obično automatizirani alati, koji skeniraju aplikaciju iz perspektive napadača. Napadači često svoje napade pomno planiraju, počevši s probiranjem aplikacije kako bi pronašli ranjivosti koje kasnije mogu iskoristiti. Početno istraživanje aplikacije ne uključuje umetanje upita ili izmjenu podataka na stranici, takvim pristupom napadači često ostanu neprimijećeni prije stvarnog napada.

Skeneri ranjivosti imaju isti pristup pri istraživanju ranjivosti, s toga skeniranje možemo podijeliti na dva tipa [28]:

- **Pasivno skeniranje** - U pasivnom skeniranju utvrđuje se može li alat pronaći ranjivosti uzimajući u obzir postojeću strukturu aplikacije i njene mreže. U ovom dijelu skeniranja skeneri pokušavaju identificirati koje se komponente ili tehnologije koriste u aplikaciji, kao što su operacijski sustavi, portovi, trenutne zakrpe na uređajima ili tehnologijama itd.
- **Aktivno skeniranje** - U aktivnom skeniranju utvrđuje se mogu li zahtjevi koji se šalju aplikaciji upućivati na moguće ranjivosti. Aktivno skeniranje simulira stvarni napad na aplikaciju, otkrivajući ranjivosti koje bi potencijalni napadači mogli iskoristiti. Ako se napad već dogodio, aktivno skeniranje može pokazati način na koji je napadač iskoristio ranjivost na aplikaciji.

5.2 Konfiguracija ranjive aplikacije

Cilj istraživanja je utvrditi razinu primjenjivosti raznovrsnih skenera ranjivosti pri unaprjeđenju sigurnosti *web* aplikacija. Istraživanje je provedeno u laboratorijskim uvjetima. Kako bi se dobili objektivni rezultati prilikom skeniranja ranjivosti na *web* aplikaciji korištena je predefinirana namjerno ranjiva *web* aplikacija *OWASP Mutilidae II.*

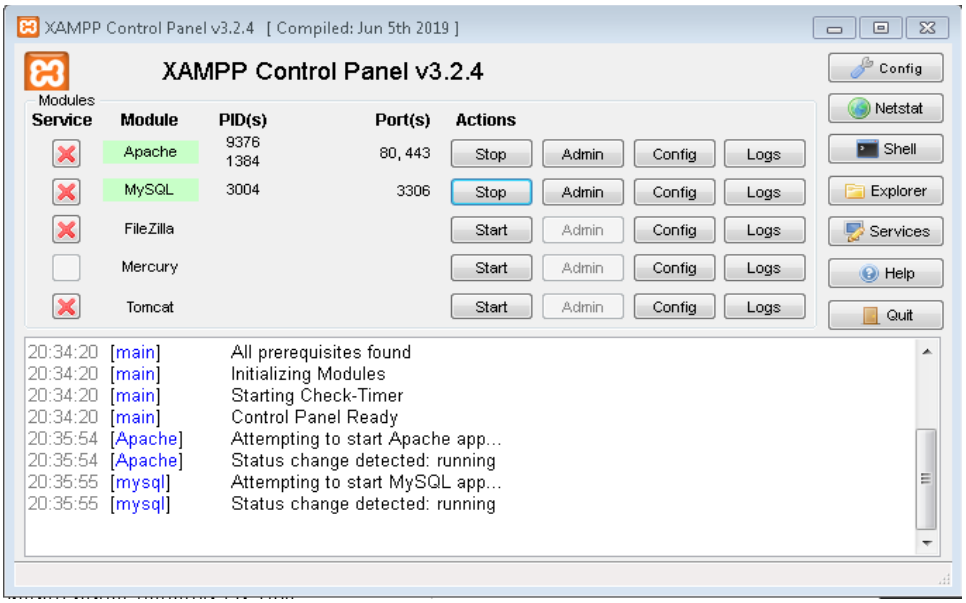
OWASP Mutilidae II je besplatna, namjerno ranjiva *web* aplikacija otvorenog koda, koja pruža uslugu simulacije stvarne ranjive *web* aplikacije te služi kao meta napada za potencijalne korisnike koji žele naučiti više o *web* sigurnosti. *Mutilidae* se može instalirati na *Linux* i *Windows* operacijskim sustavima uz pomoć neke od skupine rješenja poslužitelja za posluživanje na lokalnoj adresi, kao što su LAMP, WAMP ili XAMPP. Neke od korisnih značajki ove ranjive web aplikacije su [29]:

- Sadrži više od 40 vrsta ranjivosti i izazova,
- Simulacija je stvarne ranjive aplikacije,
- Instalacija na više operacijskih sustava i laka konfiguracija za lokalno posluživanje,
- *Web* aplikacija se može vratiti na zadane postavke pomoću gumba „Setup“,
- Korisnik se može prebacivati između sigurnog i nesigurnog načina rada,
- Često ažuriranje *web* aplikacije,
- Izvorni kod sadrži rješenja za sigurnosne ranjivosti koje se nalaze unutar aplikacije.

Mutilidae ranjiva aplikacija je preuzeta s izvora [29].

Za potrebe ovog rada, web aplikacija će se pokretati u XAMPP-u na operacijskom sustavu *Windows 7* (x64). XAMPP je besplatni paket rješenja za više poslužitelja s više platformi otvorenog koda koji je razvio *Apache Friends*, a sastoji se uglavnom od *Apache HTTP* poslužitelja, baze podataka *MariaDB* i tumača za skripte napisane u programskim jezicima *PHP* i *Perl*. XAMPP se može preuzeti s izvora [30].

Za posluživanje ranjive *web* aplikacije na lokalnoj adresi, potrebno je pokrenuti *Apache* poslužitelj i *MySQL* bazu podataka u korisničkom sučelju XAMPP programa kao što je prikazano na slici 2.



Slika 2. Prikaz XAMPP sučelja i pokretanje lokalnog poslužitelja

Nakon pokretanja servera i baze podataka, potrebno je premjestiti direktorij s izvornim kodom *Mutilidae* aplikacije u izvršni direktorij XAMPP-a na lokalnom disku. U ovom slučaju je to na lokaciji *C:\xampp\htdocs* u Windows 7 operacijskom sustavu pod imenom *mutilidae*. Aplikaciji se tada može pristupiti na lokalnoj adresi URL adresi *127.0.0.1/mutilidae* prikazano na slici 3.

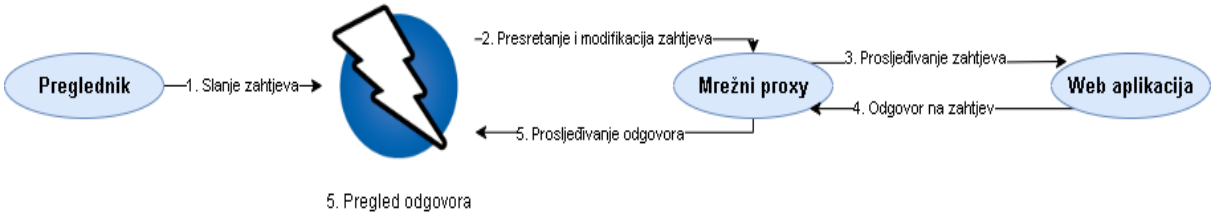
Slika 3. Prikaz grafičkog sučelja Mutilidae aplikacije na lokalnoj adresi

Aplikacija je sada spremna za rad. Kao što je prikazano na slici 3., *Mutilidae* aplikacija nudi mnogo mogućnosti. Na lijevoj strani aplikacije nalaze se izbornici za testiranje pojedinih ranjivosti, dokumentacija, obavijesti i slično. U zaglavlju aplikacije, nalaze se korisne funkcionalnosti aplikacije kao što su gumbovi za podešavanje razine sigurnosti, savjeta prilikom penetracijskog testiranja te gumbovi za ponovno postavljanje baze podataka, provedbe TLS protokola i za prikaze log i snimljenih podataka. Uz to na radnom prostoru početne stranice mogu se pronaći korisne informacije, obavijesti te popis ranjivosti po stranicama u aplikaciji.

U sljedećim potpoglavljima će se koristiti već navedeni alati za skeniranje ranjivosti na web aplikaciji *Mutilidae*. Nakon svakog obavljenog skeniranja ranjivosti aplikacija će biti postavljena na predefinirane postavke kako bi se vjerodostojno prikazali rezultati skeniranja za svaki alat.

5.3 OWASP Zed Attack Proxy (ZAP)

Zed Attack Proxy (ZAP) je besplatni alat otvorenog koda koji se koristi za penetracijsko testiranje koji je razvijen i održavan pod okriljem OWASP organizacije. ZAP je dizajniran i razvijen posebno za testiranje sigurnosti *web* aplikacija, uz to je veoma fleksibilan i proširiv. ZAP funkcionira na principu „*man-in-the-middle proxy*“. Na taj način ZAP presreće i pregledava zahtjeve i odgovore između preglednika testera i *web* aplikacije, po potrebi može izmijeniti sadržaje zahtjeva i proslijediti ih na odredište. Ako između ZAP alata i web aplikacije postoji mrežni *proxy* tada se ZAP može konfigurirati da se spaja na *proxy* kao što je prikazano na slici 4. [31].



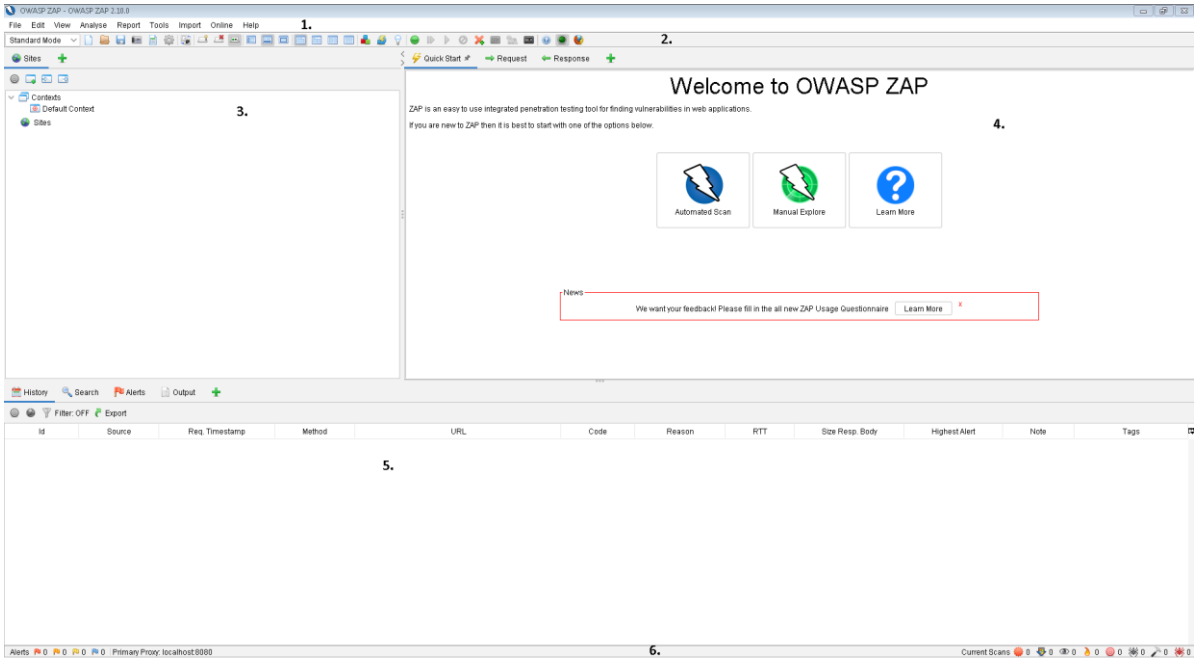
Slika 4. Prikaz djelovanja ZAP alata između web aplikacije i preglednika[31]

ZAP nudi niz funkcionalnosti koji omogućuju korištenje alata naprednim programerima, ali i početnicima u sigurnosnom testiranju. Dostupan je na svakom poznatom operacijskom sustavu kao što su Linux, Windows i OS X, a proširiv je putem dodatnih funkcija koje se mogu preuzeti slobodno sa ZAP *Marketplace*-a.

Neke od osnovnih funkcionalnosti i mogućnosti ovog alata su [31]:

- Mogućnost aktivnog i manualnog skeniranja,
- Upozorenja i rangiranje prijetnji po ozbiljnosti,
- Mogućnost postavljanja politike skeniranja i točaka prekida,
- Korištenje osnovnog „*spider*“ i „*AJAX spider*“ alata za pronalaženje novih izvora za skeniranje na aplikaciji,
- Postavljanje opsega skeniranja, autentifikacije putem alata, načina skeniranja,
- Mogućnost podešavanja strukturnih parametara i modifikatora prema aplikaciji,
- Kreiranje zasebnih korisnika i sesija za različite aplikacije,
- Dodatne mogućnosti putem dodataka i *Marketplace*-a,
- Grafičko sučelje za oba načina skeniranja.

Nakon preuzimanja i instalacije ZAP alata, prilikom prvog pokretanja korisnik je upitan da li želi nastaviti sa ZAP sesijom. Prema zadanim postavkama, ZAP sesije uvijek se snimaju na disk u bazi podataka *HSQLDB* sa zadanim imenom i mjestom. Ako ne nastavite sesiju, te se datoteke brišu pri izlasku iz ZAP -a. Nakon odabira nove sesije korisniku je prikazano grafičko sučelje alata kao što je prikazano na slici 5. [31].



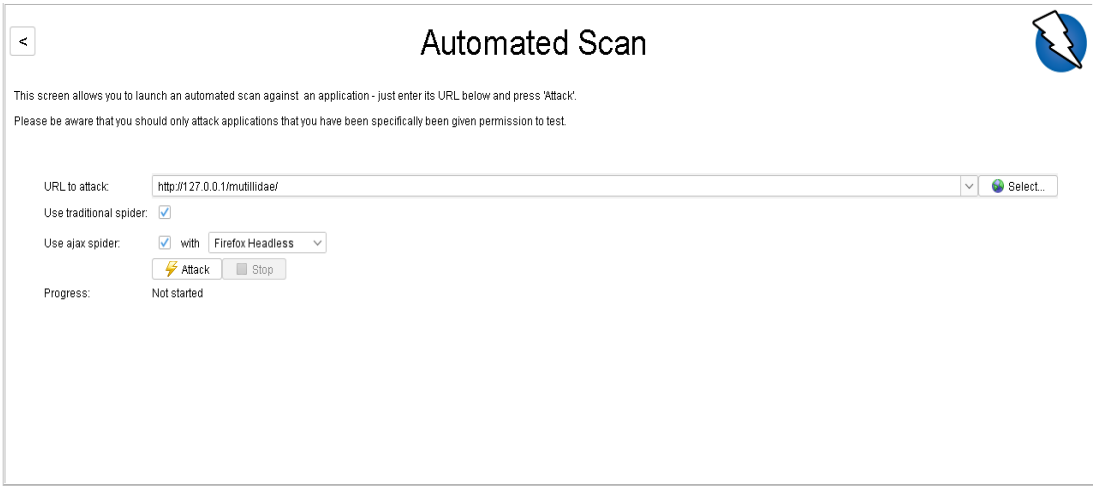
Slika 5. Prikaz grafičkog sučelja ZAP alata

Grafičko sučelje se sastoji 6 ključnih elemenata [31]:

1. **Traka izbornika** – Omogućuje pristup automatiziranim i ručnim alatima.
2. **Alatna traka** – Uključuje gumbe koji omogućavaju jednostavan pristup najčešće korištenim alatima.
3. **Navigacijski prozor povezan sa skeniranjem** – Prikazuje stablo posjećenih *web* stranica i skripti.
4. **Prozor radnog prostora** – prikazuje zahtjeve, odgovore i skripte, koje je moguće urediti.
5. **Prozor s informacijama** – Prikazuje pojedinosti i informacije o automatiziranom i ručnom skeniranju.
6. **Podnožje** – Prikaz sažete informacije i statuse glavnih alata koji se trenutno koriste.

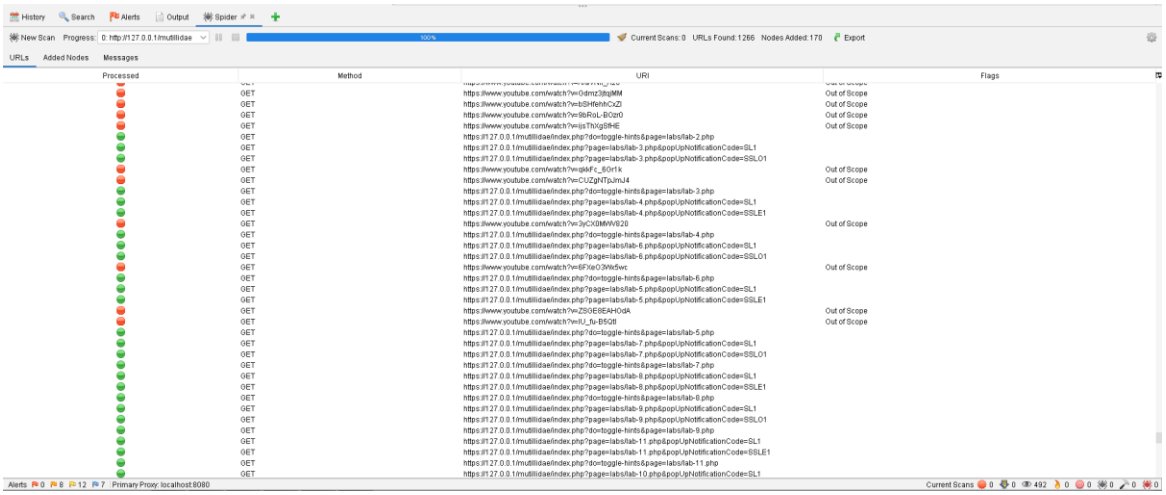
5.3.1 Pokretanje ZAP skeniranja ranjivosti na *Mutilidae* aplikaciji

Za početak automatskog skeniranja potrebno je stisnuti gumb *Automated Scan.* Nakon toga od korisnika se traži da unese željenu metu skeniranja i odabere dodatne opcije skeniranja. Za potrebe ovog rada odabrane su opcije na slici 6.



Slika 6. Prikaz odabira mete i dodatnih opcija za skeniranje ranjivosti

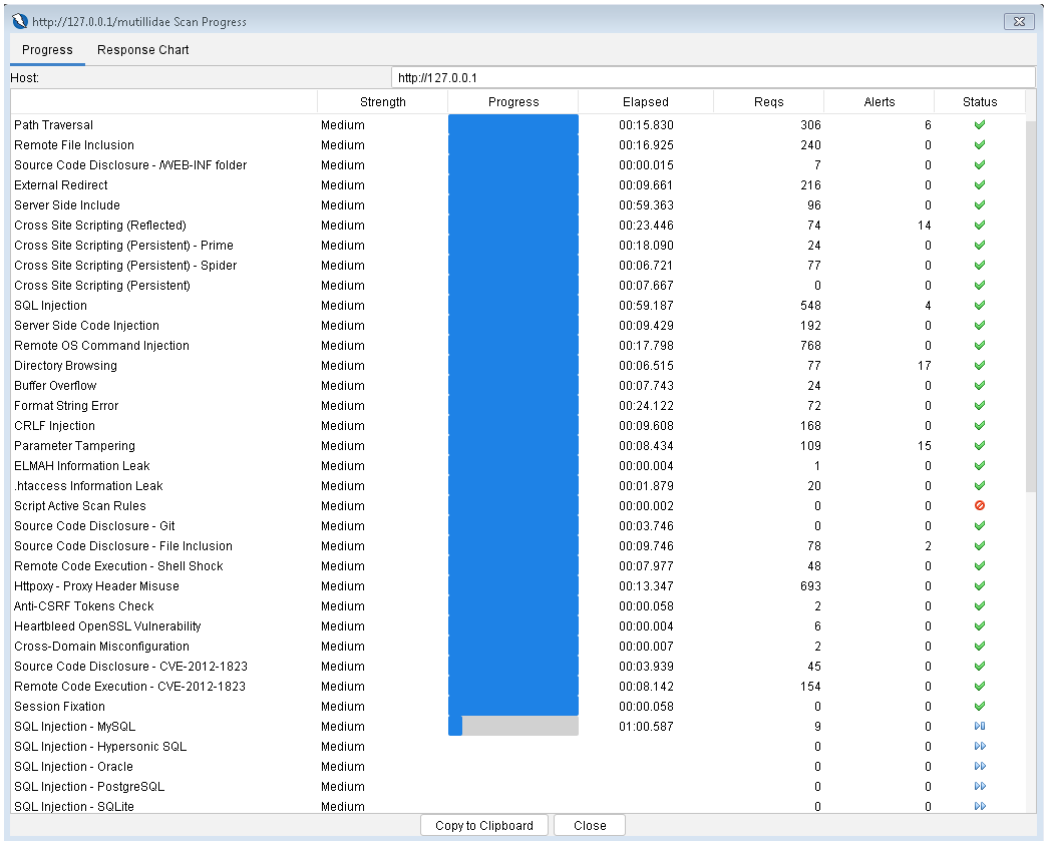
Nakon odabira opcija, potrebno je pritisnuti gumb *Attack* da započne skeniranje. Nakon pokretanja skeniranja, alat prvo obavlja pasivno skeniranje, istraživajući moguće puteve napada te informacije o aplikaciji i sustavu. Rad alata se može promatrati u prozoru informacija i podnožju kao što je prikazano na slici 7.



Slika 7. Prikaz dobivenih putanja napada putem pasivnog skeniranja

Nakon što je pasivno skeniranje završeno, alat prelazi na aktivno skeniranje. Također tijekom aktivnog skeniranja je moguće pratiti informacije o radu alata, te već pronađene ranjivosti su dostupne putem *Alerts* izbornika u prozoru informacija. Osim

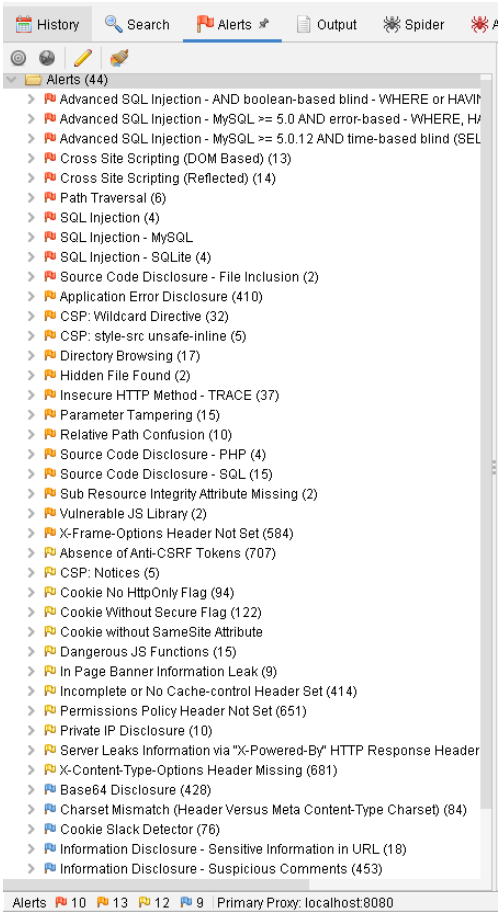
toga moguće je pratiti i detaljan napredak skeniranja pritiskom na gumb *Show scan progress details* u *Active Scan* izborniku. Tada korisnik dobiva detaljniji prikaz napretka skeniranja kao što je prikazano na slici 8.



Slika 8. Prikaz napretka aktivnog skeniranja u ZAP alatu

5.3.2 Rezultati skeniranja ranjivosti putem ZAP alata

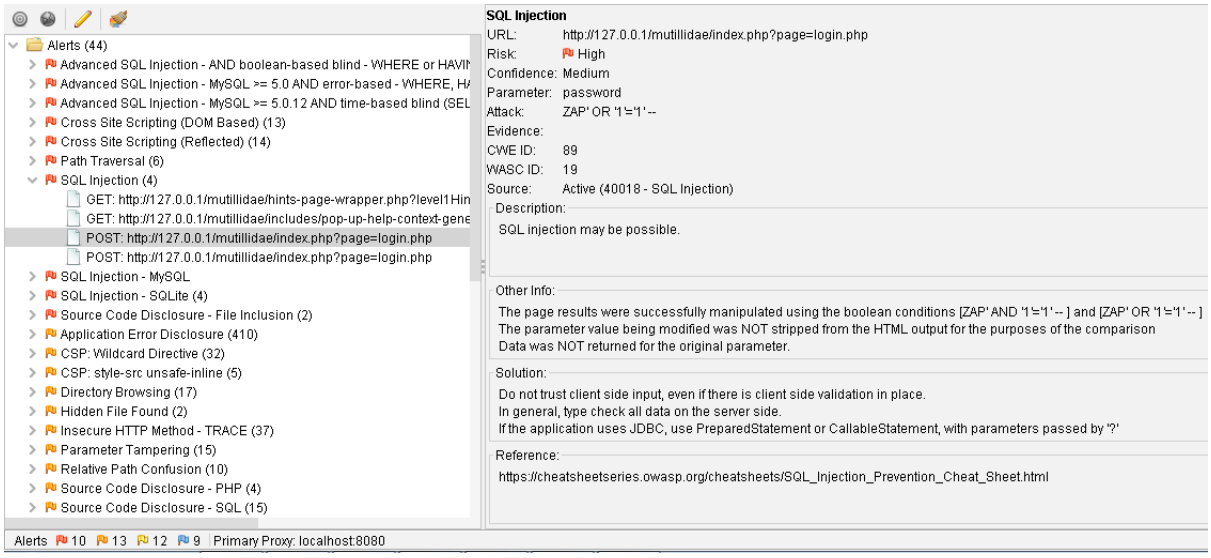
Nakon završetka automatskog skeniranja rezultati su posloženi u *Alerts* izborniku u prozoru s informacijama. Ukupno vrijeme skeniranja trajalo je 30 minuta, a ZAP alat je pronašao ukupno 44 ranjivosti u *Mutilidae* aplikaciji. *Alerts* izbornik sortira ranjivosti po razini ozbiljnosti i prioriteta od visoke, srednje i niske razine, te informacijske ranjivosti koje mogu potencijalno biti ranjivosti ili pak odaju nepotrebne informacije o web aplikaciji. Na slici 9. prikazan je izbornik *Alerts* koji pokazuje ranjivosti koje su pronađene u *web* aplikaciji *Mutilidae*.



Slika 9. Prikaz ranjivosti nakon automatskog skeniranja

Za detaljniju analizu pojedinih ranjivosti moguće je dobiti više informacija klikom na ranjivosti i odabirom određenog zahtjeva ili odgovora aplikacije. Na slici 10. prikazana je SQL ranjivosti koji je moguće iskoristiti unosom upita *„ZAP' OR '1'='1' –„* u obrascu za prijavu na adresi *http://127.0.0.1/mutillidae/index.php?page=login.php.* U ovom slučaju parametar koji se manipulira je obrazac za lozinku kao što je vidljivo na slici 10. pod pojmom *Parameter*. Nakon iskorištavanja ove ranjivosti napadač je u mogućnosti se logirati kao administrator na *Mutilidae* aplikaciji.

Osim informacija o napadu, u informacijskom prozoru se nalaze i pomoćne informacije za rješavanje ranjivosti u tom slučaju. Osim informacija u aplikaciji za rješavanje ranjivosti su uključeni dodatni izvori kako bi korisnici *web* aplikacija mogli istražiti specifičnu ranjivosti i pronaći adekvatno rješenje za vlastite aplikacije.



Slika 10. Prikaz detaljnijih informacija o SQL ranjivosti unutar Mutilidae aplikacije

5.4 Vega

Vega je platforma za testiranje sigurnosti *web* aplikacija. *Vega* je besplatan alat otvorenog koda napisan u *Java* programskom jeziku i može se pokretati na *Windows*, *Linux* i *OS X* operacijskim sustavima. Temelji se na grafičkom sučelju i može se lako proširiti pomoću modula napisanih u *Javascript*-u. Razvijen je od strane Subgraph kompanije koja se bavi informacijskom sigurnosti baziranom isključivo na otvorenom kodu [32].

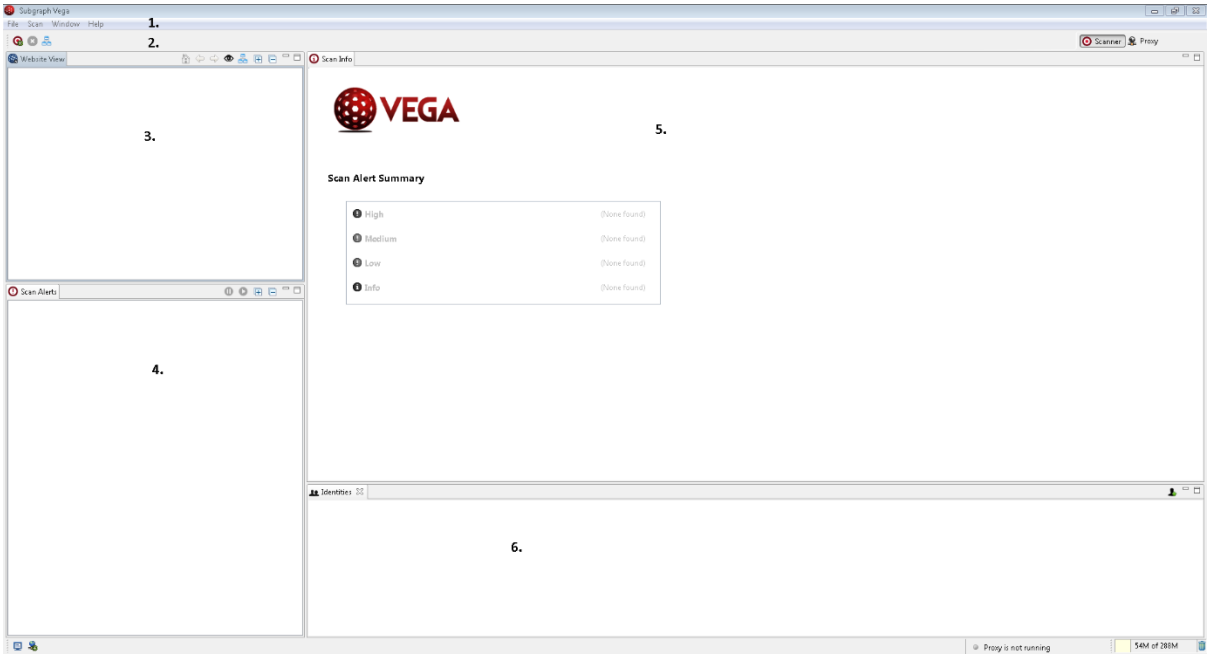
Vega radi na sljedeća dva načina [32]:

- **Automatski skener -** Automatizirani skener automatski indeksira *web* stranice, izdvaja veze, obrađuje obrasce i pokreće module na mogućim točkama ubrizgavanja koje otkrije. Ovi moduli mogu činiti stvari poput automatskog podnošenja zahtjeva koji *fuzziraju* parametre, na primjer, za testiranje ranjivosti kao što su XSS ili ubrizgavanje SQL upita.
- **Presretanje proxy-a -** Presretački *proxy* omogućuje detaljnu analizu interakcije preglednika i aplikacije. Kada je omogućeno, *proxy* sluša na *localhost*-u kao *proxy* poslužitelj. Kada preglednik koristi Vega *proxy*, zahtjevi i odgovori vidljivi su alatu. Tada se u alatu mogu postavljati točke prekida, kriterije presretanja za odlazne zahtjeve ili dolazne odgovore. Ti se zahtjevi i odgovori čuvaju u stanju u kojem se mogu uređivati do objavljivanja.

Osim osnovnih funkcionalnosti Vega posjeduje i dodatne funkcionalnosti kao što su [32]:

- ***Proxy* skeniranje** – Omogućuje aktivno testiranje i skeniranje aplikacije putem preglednika.
- **Obrada odgovora** – Vega podržava module koji obrađuju odgovore tražeći dodatne informacije.
- **Zajednička baza znanja** – Informacije, zahtjevi i odgovori se pohranjuju i mogu dijeliti između komponenti.
- **Upozorenja** - Obje vrste modula mogu generirati upozorenja koja uključuju kombinaciju dinamičkog sadržaja iz modula i statičkog sadržaja u XML datoteci navedenoj u upozorenju.
- **Proširivost** – Proširenje funkcionalnosti pisanjem novih modula u *JavaScript* jeziku.

Nakon instalacije *Vega* alata i pokretanja korisniku je prikazano grafičko sučelje alata kao na slici 11.



Slika 11. Grafičko sučelje *Vega* alata

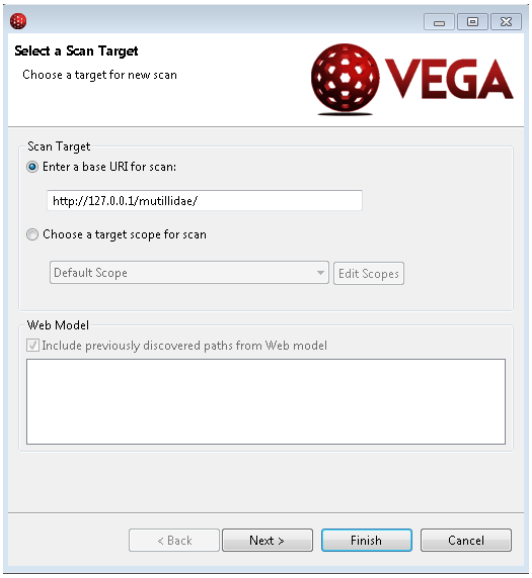
Grafičko sučelje se sastoji od 6 osnovnih elemenata:

1. **Izbornik alata** – Pristup alatima i funkcionalnostima Vega alata.

2. **Alatna traka** – Omogućuje pokretanje novog skeniranja i odabir načina rada alata.
3. **Prozor prikaza puteva *web* aplikacije** – Sadrži informacija o mogućim putanjama unutar *web* aplikacije.
4. **Prozor o upozorenjima na aplikaciji** – Prikazuje sažete informacije o upozorenjima i ranjivostima na *web* aplikaciji.
5. **Radni prozor s informacijama o skeniranju** – Prikaz cjelokupnih informacija o pronađenim ranjivostima.
6. **Prozor korisničkih identiteta** – Prozor povezan za omogućavanje autentifikacije korisnika na skeniranoj *web* aplikaciji.

5.4.1 Pokretanje Vega skeniranja ranjivosti na *Mutilidae* aplikaciji

Za pokretanje automatskog skeniranja potrebno je kliknuti na gumb *Start New Scan* na alatnoj traci u *Scanner* načinu rada. Nakon toga otvara se novi prozor i od korisnika se traži da unese adresu mete skeniranja kao što je prikazano na slici 12. Nakon postavljanja mete skeniranja potrebno je pritisnuti gumb *Finish* da skeniranje započne.

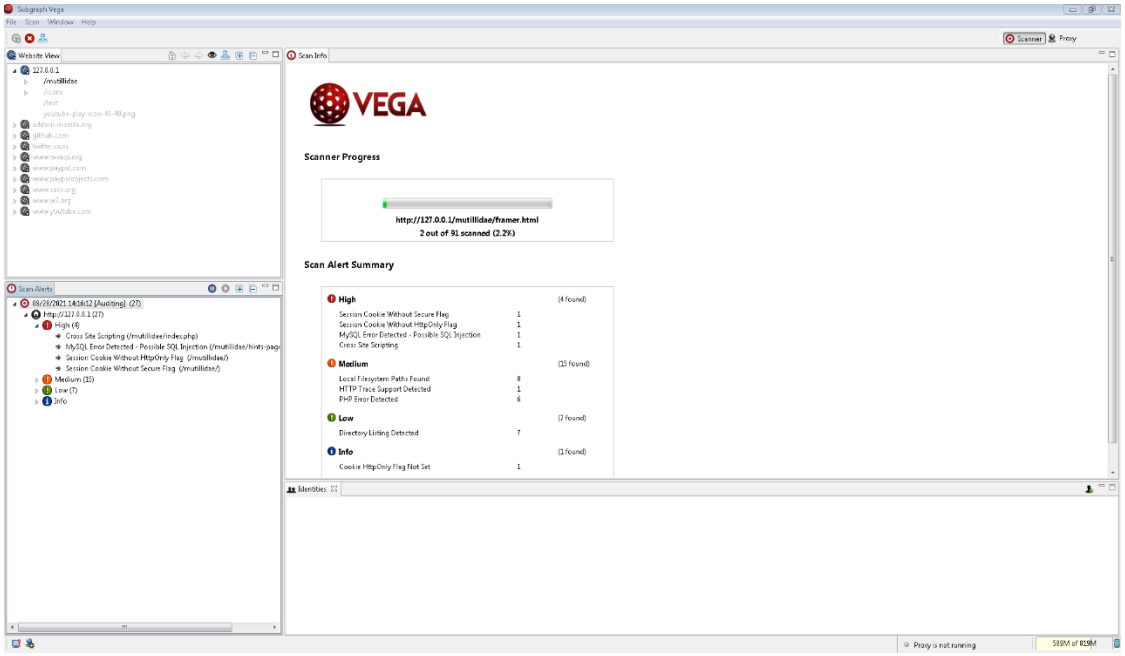


Slika 12. Postavljanje mete skeniranja u *Vega* alatu

Korisnik je osim toga u mogućnosti postavljati i dodatne opcije skeniranja pritiskom na gumb *Next*. Dodatne opcije uključuju dodatne module napada kao SQL

napade bazirane na vremenu, slijepo umetanje komandi itd. Osim toga korisnik može postaviti i identitet kako bi se autentificirao na aplikaciji za veći opseg skeniranja. Zbog potreba ovog rada koristit će se predefinirane opcije.

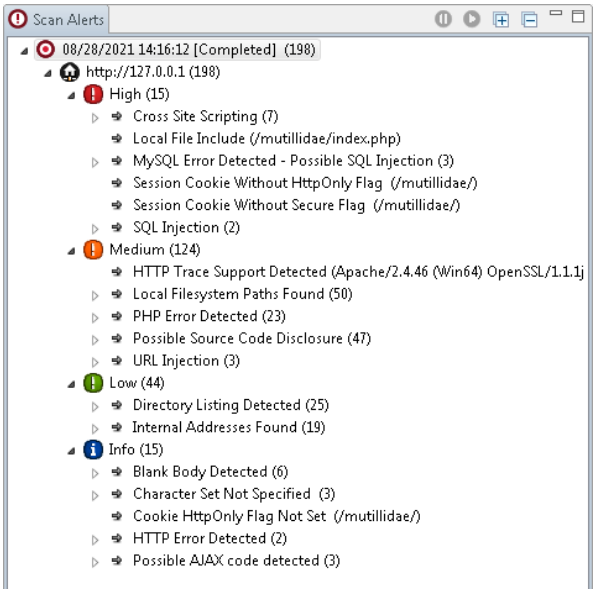
U tijeku procesa skeniranja korisnik je u mogućnosti pratiti pronađene puteve koji će se aktivno skenirati, upozorenja i ranjivosti koje su pronađene, te gledati zahtjeve i odgovore koji uzrokuju ranjivosti. Na slici 13 prikazano je grafičko sučelje *Vega* alata u tijeku skeniranja aplikacije.



Slika 13. Prikaz tijeka skeniranja ranjivosti u *Vega* alatu

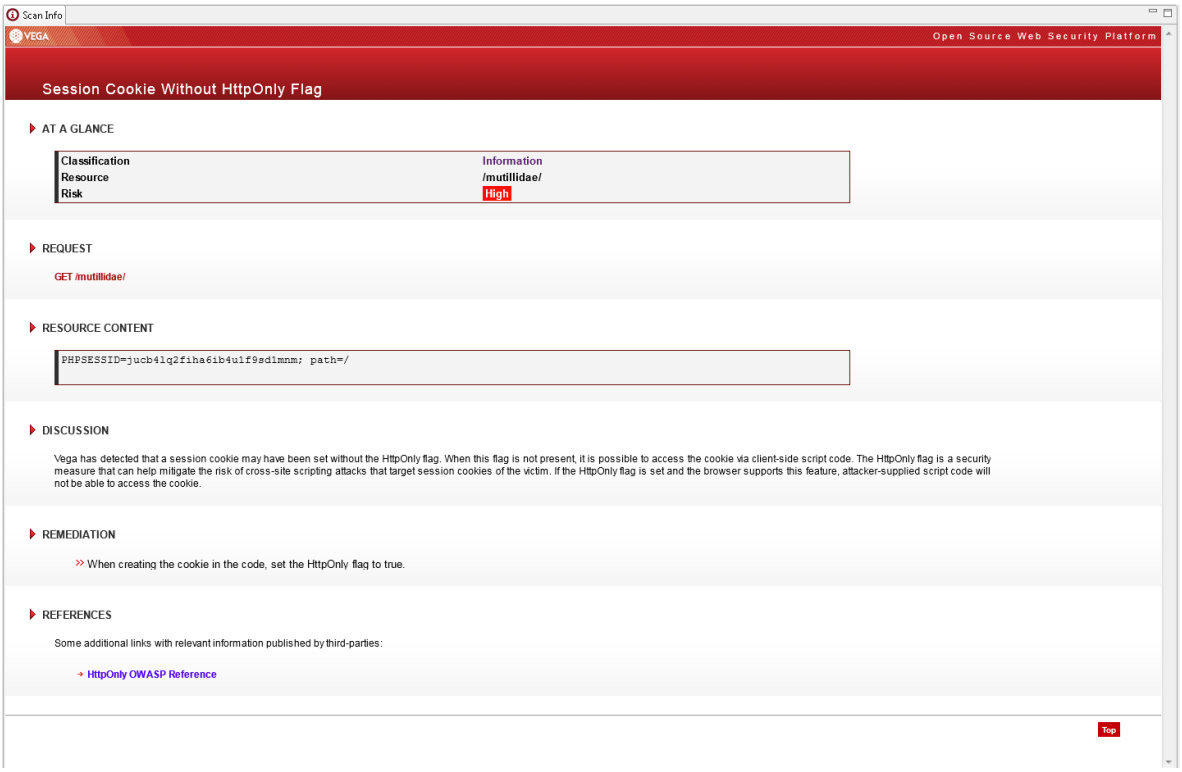
5.4.2 Rezultati skeniranja ranjivosti putem Vega alata

Ukupno vrijeme skeniranja je trajalo 18 minuta, a *Vega* alat je pronašao 198 različitih ranjivosti unutar *Mutilidae* aplikacije. Vega alat također rangira ranjivosti po ozbiljnosti i sortira ih po prioritetu u *Scan Alerts* prozoru kao što je prikazano na slici 14. Unutar tog prozora moguće je pogledati sve ranjivosti koje su pronađene unutar aplikacije, te zahtjeve ili odgovore koji su korišteni za pronalaženje istih.



Slika 14. Prikaz skeniranih ranjivosti putem *Vega* alata

Za više informacija o pojedinoj ranjivosti potrebno je kliknuti na zahtjev ili ranjivost unutar *Scan Alerts* prozora. Nakon toga unutar *Scan Info* prozora korisniku se omogućuje detaljniji pregled ranjivosti, zajedno s informacijama o toj ranjivosti i poveznicama o potencijalnom sprječavanju takvih ranjivosti.



Slika 15. Prikaz informacija o pronađenoj ranjivosti u Vega alatu

Ako korisnik želi detaljnije analizirati zahtjev ili odgovor koji je poslan koji je otkrio ranjivosti aplikacije, tada je potrebno kliknuti na zahtjev unutar *Request* sekcije u *Scan Info* prozoru.

5.5 Arachni

Arachni je modularni *Ruby* okvir visokih performansi ispunjen značajkama čiji je cilj pomoći testerima i administratorima penetracije u procjeni sigurnosti modernih *web* aplikacija. Besplatan je i otvorenog koda, podržavan na glavnim operacijskim sustavima *Linux*, *Windows* i *OS X* i distribuira se putem prijenosnih paketa koji omogućuju trenutnu implementaciju. Opcije implementacije su putem *Ruby* programske biblioteke, komandnog sučelja (CLI – engl. *Command Line Interface*), web grafičkog sučelja (*WebUI*) te distribuiranog sustava koristeći udaljene agente [33].

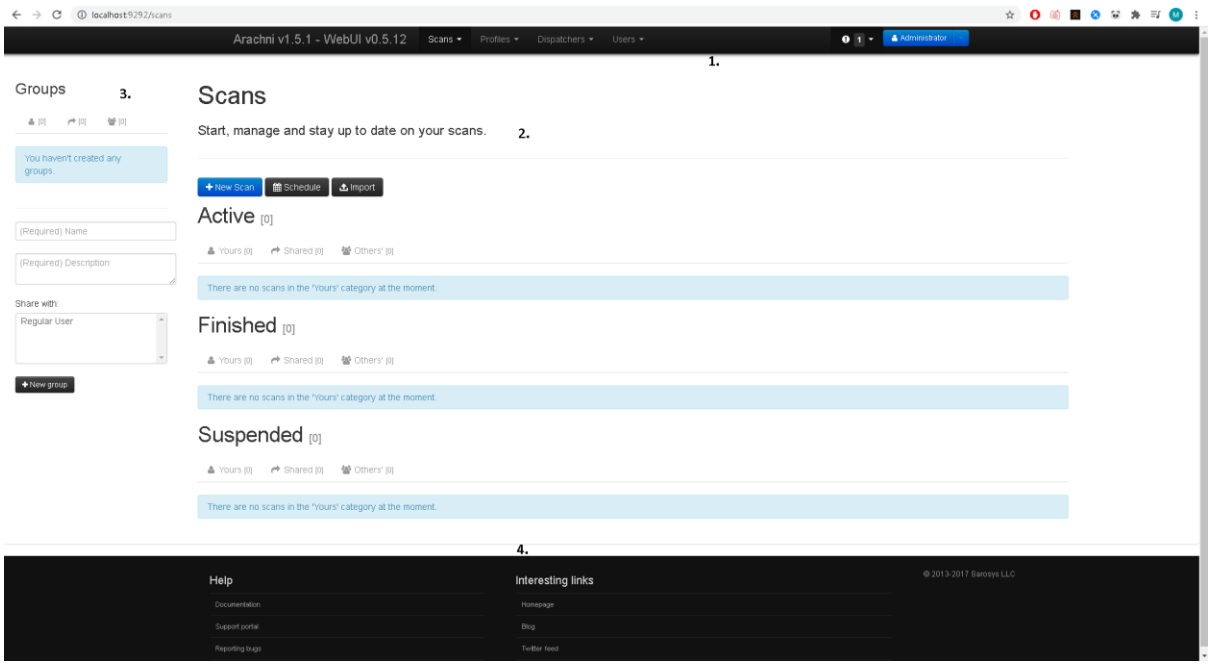
Arachni je veoma fleksibilan i svestran alat koji pokriva mnoge slučajeve uporabe kada je u pitanju sigurnost *web* aplikacija. Može se koristiti kao jednostavni pomoćni program putem komandnog sučelja za skeniranje, do globalne mreže skenera visokih performansi, zahvaljujući REST API-u integracija je jednostavna [33].

Glavne značajke *Arachni* alata [33]:

- *Arachni* uključuje integrirano, stvarno okruženje preglednika kako bi se osigurala dovoljna pokrivenost suvremenim web aplikacijama koje koriste tehnologije kao što su HTML5, *JavaScrip*t, DOM manipulacija, AJAX itd.
- Preglednički klaster koordinira analizu resursa preglednika i omogućuje sustavu obavljanje operacija koje bi obično zahtijevale dosta vremena na način visokih performansi.
- Velika pokrivenost prilikom interakcije sa složenim *web* aplikacijama zbog integriranog okruženja preglednika.
- Otvorena distribuirana arhitektura omogućuje da se *Arachni* uklopi u tijek rada i integrira sa već postojećim arhitekturama.
- Veoma konfigurabilan,proširiv i skalabilan.
- Automatsko skeniranje i mogućnost pauze i nastavljanja skeniranja.
- Razne podrške za *proxy*, kolačiće, zaglavlja itd.

- Skenira veliki broj ranjivosti aktivnim i pasivnim načinom rada.

Nakon preuzimanja i instalacije samostalnih paketa za pokretanje *Arachni*-a, potrebno je poslužiti *WebUI* na lokalnoj adresi kako bi mogli pristupiti funkcionalnostima alata. Za jednostavno pokretanje posluživanja potrebno je unutar *Arachni* direktorija pokrenuti skriptu *arachni_web.bat*, te nakon izvršenja skripte program ispisuje lokalnu adresu na kojoj se poslužuje *WebUI*, obično je to *localhost:9292* adresa kao što je prikazano na slici 16.



Slika 16. Prikaz *WebUI* sučelja *Arachni* alata

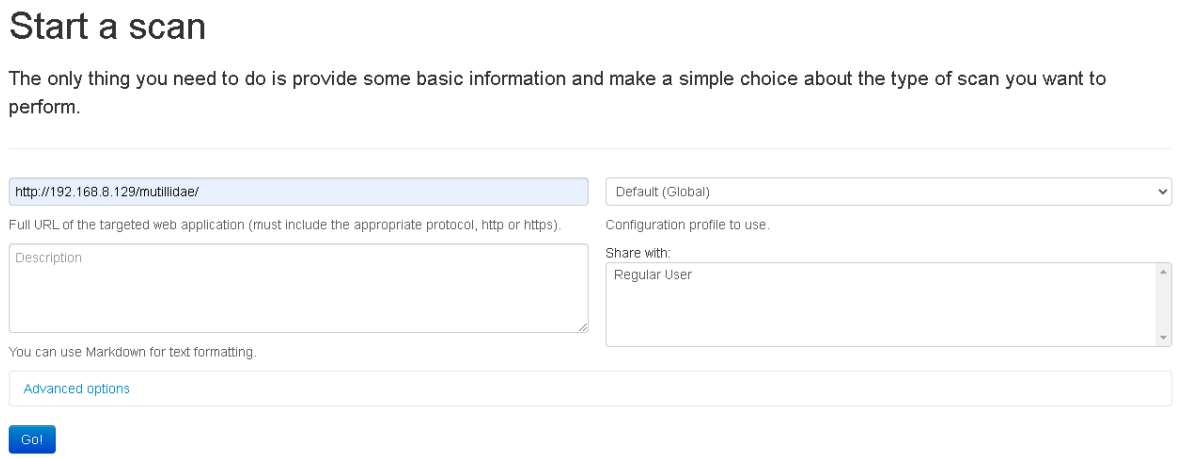
Sučelje se može podijeliti na 4 elementa:

1. **Zaglavlje** – Zaglavlje nudi mnogo funkcionalnosti i pristup svim alatima za novo skeniranje, uređivanje profila skeniranja, kreiranju novih korisnika i organizaciji skeniranja ili udaljenih uređaja za skeniranje.
2. **Radna površina** – Na radnoj površini se nalaze svi bitni elementi za skeniranje. Tu se mogu pronaći svi alati i informacije vezani za skeniranje, kao što su podaci o aktivnim,završenim i prekinutim skeniranjima. Nakon odabira skeniranja moguće je dobiti uvid u sve informacije o skeniranju, ranjivostima, zahtjevima itd.
3. **Prozor za grupe** – Ovaj prozor nudi uvid u organizaciju grupnih skeniranja i svih funkcionalnosti povezanih s njima.

4. **Podnožje** – Nudi pristup dokumentaciji i raznim informacijama koje mogu pomoći pri korištenju alata.

5.5.1 Pokretanje Arachni skeniranja ranjivosti na *Mutilidae* aplikaciji

Korisnik može pokrenuti automatsko skeniranje putem izbornika u zaglavlju ili klikom na gumb *New scan* na radnoj površini. Nakon odabira od korisnika se traži da unese URL mete skeniranja, te uz to može odabrati profil skeniranja, dodatne napredne opcije i opis skeniranja, te s kime će podijeliti skeniranje. *Arachni* nije u mogućnosti skenirati putem lokalne adrese, već je potrebno unijeti privatnu IP adresu računala na kojem se poslužuje aplikacija. U ovom slučaju aplikacija će se skenirati na URL adresi *http://192.168.8.129/mutillidae/* kao što je prikazano na slici 17. Nakon toga potrebno je pritisnuti gumb *Go!* i skeniranje će započeti.

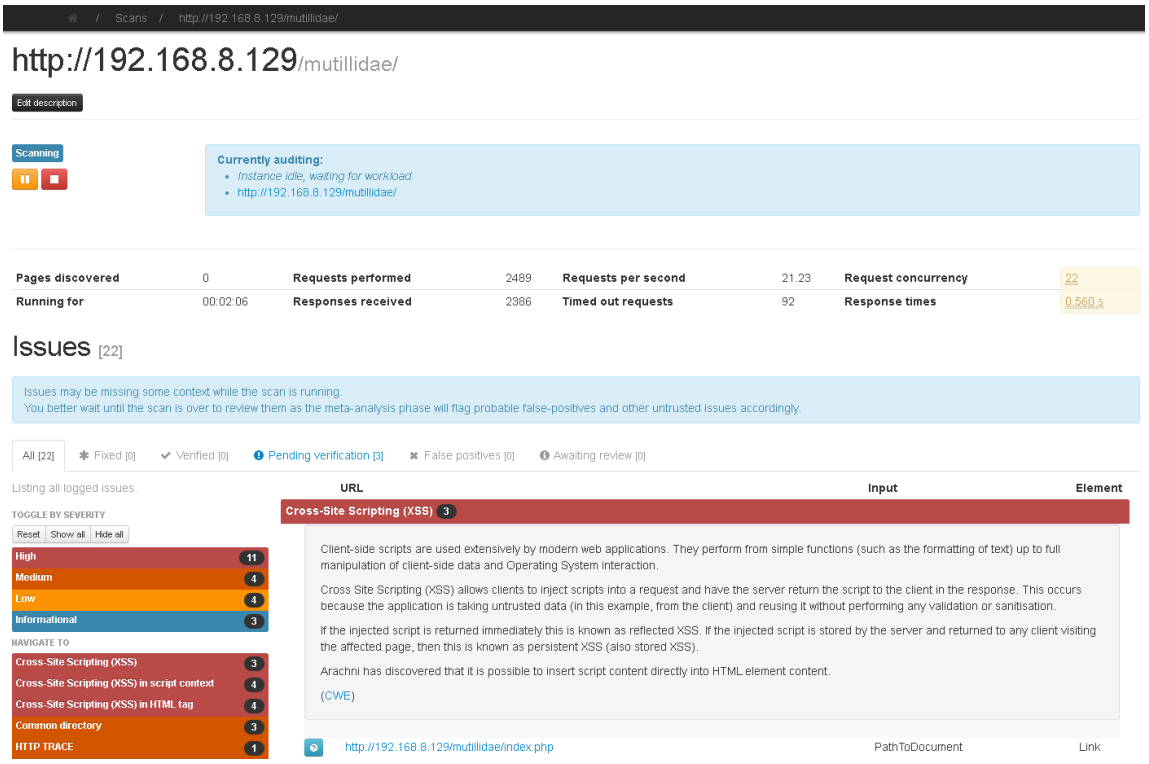


Slika 17. Postavljanje mete skeniranja na *WebUI Arachni* alata

U toku automatskog skeniranja, kao i kod ostalih alata, korisnik je u mogućnosti promatrati proces skeniranje te dostupne pronađene ranjivosti su mu dostupne na pregled. Osim toga *Arachni* nudi i jednostavniji prikaz informacija o performansama skeniranja kao što su broj poslanih zahtjeva i dobivenih odgovora, vrijeme odgovora, pronađene stranice itd. Nakon završetka aktivnog skeniranja *Arachni* provodi meta analizu skeniranih rezultata kako bi potencijalno označio moguće lažno pozitivne ranjivosti.

Na slici 18. prikazano je *web* sučelje u toku skeniranja putem *Arachni* alata. Osim navedenih informacija, korisnik je u mogućnosti i pauzirati trenutno skeniranje ili

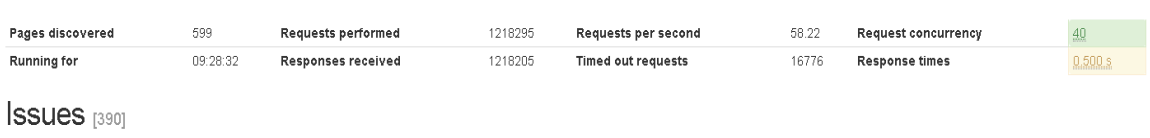
prekinuti. Što često je korisno ukoliko *Arachni* krivo koristi resurse ili je pogrešno konfiguriran.



Slika 18. Prikaz tijeka skeniranja u *Arachni* alatu

5.5.2 Rezultati skeniranja ranjivosti putem Arachni alata

Ukupno vrijeme skeniranja trajalo je 9 sati i 28 minuta te je *Arachni* pronašao ukupno 390 potencijalnih ranjivosti. Uz to *Arachni* je otkrio 599 različitih stranica ili putova unutar aplikacije, te je zabilježio veliki broj zahtjeva i odgovora, kao što je prikazano na slici 19. Kao i kod ostalih alata ranjivosti su rangirane po ozbiljnosti kao što je prikazano na slici 20.

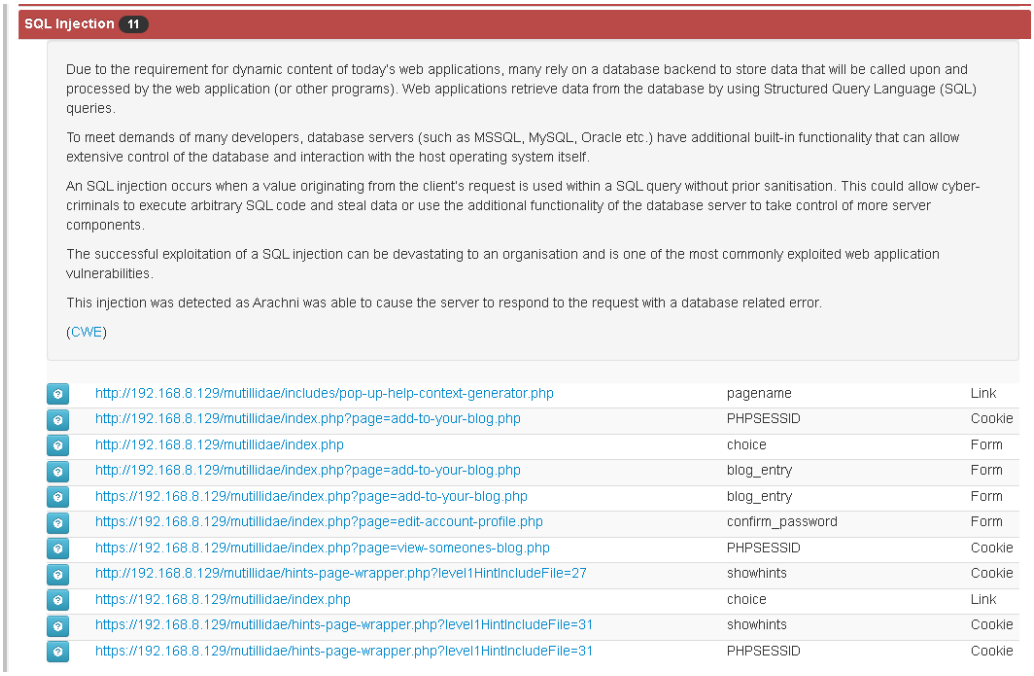


Slika 19. Prikaz performanse nakon skeniranja u *Arachni* alatu

Kao i kod ostalih alata ranjivosti su rangirane po ozbiljnosti kao što je prikazano na slici 20. Unutar tog izbornika mogu se pretražiti sve ranjivosti te na glavnom izborniku je moguće pristupiti detaljnijim informacijama o pronađenoj ranjivosti.

Slika 20. Prikaz ranjivosti unutar *WebUI Arachni* alata

Unutar informacijskog prozora o pronađenoj ranjivosti dostupno je objašnjenje o ranjivosti te mjesta u *web* aplikaciji gdje je potencijalno pronađena ta ranjivost, kao što je prikazano na slici 21. Za detaljniju analizu potrebno je kliknuti na gumb kako bi se utvrdio cijeli proces slanja zahtjeva i odgovora.



Slika 21. Prikaz informacijskog prozora o pronađenoj ranjivosti

5.6 Nikto

Nikto je besplatni alat otvorenog koda za skeniranje ranjivosti *web* poslužitelja. Nikto obavlja opsežna ispitivanja nad *web* poslužiteljima za više stavki, uključujući više od 6700 potencijalno opasnih datoteka/programa, provjerava zastarjele verzije preko 1250 poslužitelja i probleme specifične za verzije na više od 270 poslužitelja. Također provjerava postoje li stavke konfiguracije poslužitelja, kao što su prisutnost više datoteka indeksa, opcije HTTP poslužitelja, te će pokušati identificirati instalirane *web* poslužitelje i softver. Stavke i dodaci za skeniranje često se ažuriraju i mogu se automatski ažurirati [34].

Nikto je napisan u *Perl* programskom jeziku, i može se pokretati na svim operacijskim sustavima koji su bazirani na *Unix*-u. *Nikto* se isključivo koristi putem komandnog sučelja.

Neke od značajki i mogućnosti Nikto alata prema [34]:

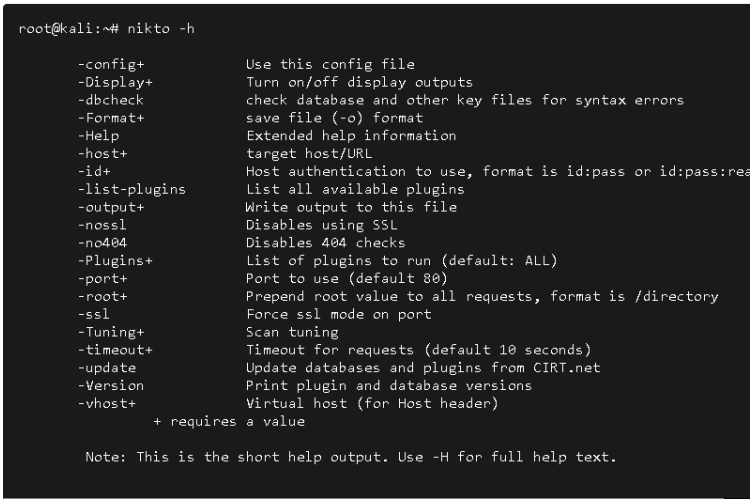
- Podrška za SSL i HTTP *proxy*,
- Provjerava postoje li zastarjele komponente poslužitelja,
- Mogućnost spremanja izvješća u raznim formatima i predlošcima,

- Skeniranje više portova na poslužitelju ili više poslužitelja,
- Indentificira instalirani softver putem zaglavlja, ikona i datoteka,
- Mogućnost nagađanja autorizacije, vjerodajnica ili poddomena,
- Tehnike mutacije za "lov" na sadržaj na *web* poslužiteljima,
- Izvješća o "neobičnim" zaglavljima,
- Maksimalno vrijeme izvođenja po cilju,
- Poboljšano smanjenje lažno pozitivnih rezultata na više načina: zaglavlja, sadržaj stranice i raspršivanje sadržaja,
- Temeljita dokumentacija.

Cilj *Nikto* alata je ispitati *web* poslužitelj kako bi se pronašli potencijalni problemi i sigurnosne ranjivosti što uključuje pogrešne konfiguracije poslužitelja i softvera, zadane datoteke i programe, nesigurne datoteke i programe, zastarjele poslužitelje i programe. *Nikto* upućuje na neke informacije o pronađenim ranjivostima ukoliko se koristi funkcija spremanja izvješća u nekim od formata, no više služi kao alat koji daje pokazivače korisnicima kako bi došli do boljeg rezultata ručnog testiranja [34].

5.6.1 Pokretanje Nikto skeniranja ranjivosti na Mutilidae poslužitelju

Nikto je izgrađen na *LibWishker2* modulu i može raditi na bilo kojoj platformi koja ima *Perl* okruženje. Za pristupanje *Nikto* alata potrebno je koristiti komandno sučelje unutar platforme. Na slici 22. prikazan je izbornik *Nikto* alata na *Linux* komandnom sučelju unošenjem komande *nikto -h*.

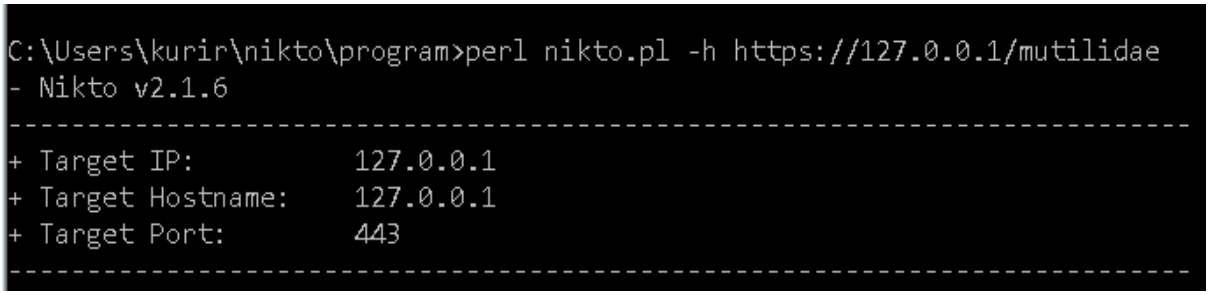


Slika 22. Izbornik *Nikto* alata unutar komandnog sučelja

Unutar izbornika nude se mnoge opcije za konfiguraciju skeniranja. Glavne naredbe koje se koriste za skeniranje su *-host* i *-port* opcije, koje omogućavaju korisniku da skenira poslužitelj putem URL adrese ili pak kombinirajući IP adresu poslužitelja i *port* na kojem se nalazi. Uz to se dodatno mogu podešavati konfiguracije skeniranja uključivanjem ili isključivanjem parametara skeniranja, postavljanjem autentifikacijski podataka, pauziranjem skeniranja itd. Jedna od prednosti *Nikto* alata je što se putem *-output* komande mogu spremiti podaci o skeniranju u raznim formatima.

Za pokretanje *Nikto* alata unutar Windows operacijskog sustava koje se koristi u ovom radu preuzet je *Perl* programski jezik kako bi se alat mogao izvršavati. Nakon preuzimanja i instalacije alata i programskog jezika, izborniku alata se može pristupiti unutar *Windows* komandnog sučelje putem naredbe *perl nikto.pl -h.*

Za skeniranje *web* poslužitelja potrebno je upisati URL adresu web aplikacije, ili IP adresu poslužitelja i broj *porta*. Za ovaj slučaj skenirati će se URL adresa *Mutilidae* aplikacije, kao što je prikazano na slici 23.



Slika 23. Pokretanje skeniranja *web* aplikacije u *Nikto* alatu

Ako korisnik želi generirati izvješće s ranjivostima nakon završetka skeniranja potrebno je dodati komandu u obliku *-o <Ime izvješća>.format* koje odgovara slici 24.

Slika 24. Generiranje izvješća putem *Nikto* alata

5.6.2 Rezultati skeniranja ranjivosti putem Nikto alata

Nikto je prijavio 166 različitih potencijalnih ranjivosti u svega 59 sekundi skeniranja. Vrijeme skeniranja na lokalnom poslužitelju je nešto brže nego skeniranje na *web* poslužitelju zbog korištenja lokalnih resursa.

Nikto ispisuje rezultate i informacije unutar komandnog sučelja, ukoliko je generirano ispisivanje rezultata u nekom od mogućih formata, tada rezultati ostaju spremljeni i mogu se pristupiti u bilo kojem trenutku u pristupačnijem formatu. Za ispisivanje rezultata *Nikto* prvo ispisuje informacije o IP adresi i broj *porta* mete, zatim slijede informacije o SSL-u i vrijeme početka skeniranja. Nakon toga *Nikto* ispisuje informacije o poslužitelju, operacijskom sustavu i tehnologijama unutar URL adrese koje se koriste, a nakon toga slijedi ispis informacija o ranjivostima kao što je prikazano na slici 25.

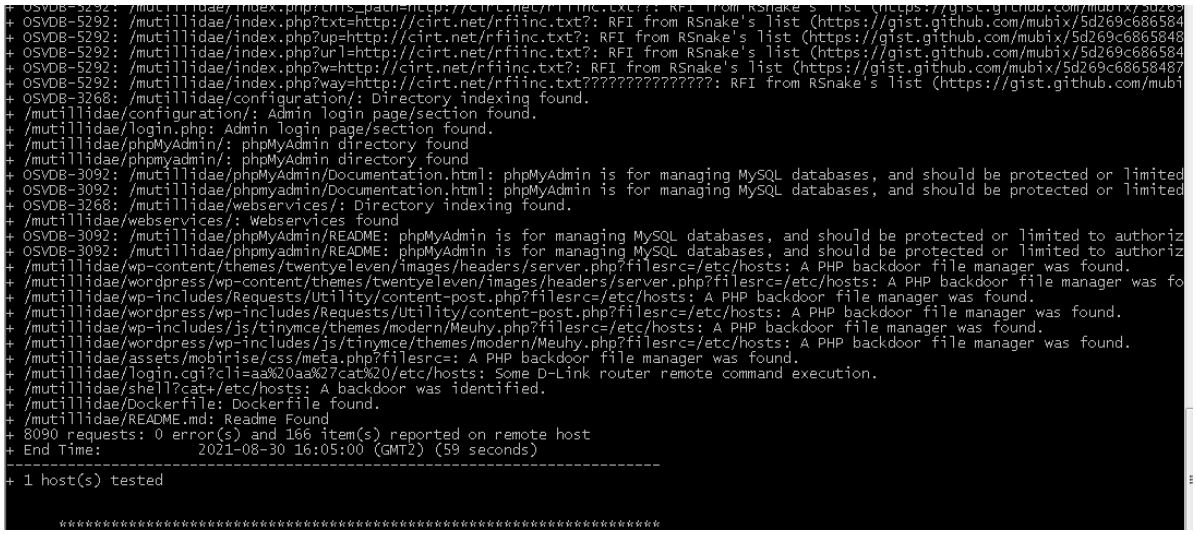
```
C:\Users\kurir\nikto\program>perl nikto.pl -h https://127.0.0.1/mutillidae/ -o report1.html
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    127.0.0.1
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=localhost
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /CN=localhost
+ Start Time:         2021-08-30 16:04:01 (GMT2)
---------------------------------------------------------------------------
+ Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1j PHP/8.0.3
+ Cookie PHPSESSID created without the secure flag
+ Cookie PHPSESSID created without the httponly flag
+ Cookie showhints created without the secure flag
+ Cookie showhints created without the httponly flag
+ Retrieved x-powered-by header: PHP/8.0.3
+ The anti-clickjacking X-Frame-Options header is not present.
+ Uncommon header 'logged-in-user' found, with contents:
+ The site uses SSL and the Strict-Transport-Security HTTP header is set with max-age=0.
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the conten
+ No CGI Directories found (use '-C all' to force check all possible dirs)
line: includes/
line: javascript/
line: classes/
line: owasp-esapi-php/
line: config.inc
line: passwords/
line: documentation/
line: phpmyadmin/
+ "robots.txt" contains 8 entries which should be manually viewed.
+ Hostname '127.0.0.1' does not match certificate's names: localhost
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
```

Slika 25. Ispis početnih informacija i ranjivosti u Nikto alatu

Kao što je vidljivo na slici 25. *Nikto* je prepoznao vrstu poslužitelja, operacijski sustav i programski jezik koji se koristi unutar *web* aplikacije. Nakon toga Nikto ispisuje ranjivosti. Prilikom analize rezultata bitno je obratiti pozornost na OSVDB ranjivosti.

To su već otkrivene ranjivosti u sličnim tehnologijama otvorenog koda koje se mogu istražiti unutar *Open Source Vulnerability Database Project* projekta.

Nakon ispisivanja ranjivosti *Nikto* ispisuje broj poslanih zahtjeva, eventualni broj greški, i broj pronađenih stavki, zajedno s ukupnim trajanjem skeniranja, kao što je prikazano na slici 26. Ako je generirano izvješće, moguće mu je pristupiti u lokalnom direktoriju Nikto alata, te ga koristiti za analizu rezultata.



Slika 26. Krajnji ispis rezultata u *Nikto* alatu

6. Analiza primjenjivosti alata otvorenog koda

Kroz analizu statističkih podataka iz relevantnih istraživanja zaključeno je da je broj napada na *web* aplikacije u porastu,a najčešća ranjivost proizlazi iz pogrešaka u sigurnosnoj konfiguraciji. U prethodnom poglavlju analizirana su četiri različita alata otvorenog koda kako bi se ispitala njihova primjenjivost u svrhu poboljšanja sigurnosti *web* aplikacija. Cilj istraživanja je utvrditi da li alati otvorenog koda mogu predstavljati alternativu komercijalnim alatima ili penetracijskom testiranju od treće strane.

Istraživanje alata provedeno je nad ranjivom *web* aplikacijom *Mutilidae.* Kroz istraživanje korišten je automatiziran način skeniranja kako bi se prikazala funkcionalnost alata, te dobio okviran pregled performansi alata. U tablici 1. mapirani su podaci o osnovnim funkcionalnostima alata.

Tablica 1. Osnovne funkcionalnosti alata korištenih u istraživanju

Ime alata	Način skeniranja	Operacijski sustav	Proširivost modula	Korisničko sučelje	Generiranje izvješća
OWASP Zap	Automatizirano i manualno skeniranje	Linux Windows OS X	Da	GUI	Da
Vega	Automatizirano skeniranje	Linux Windows OS X	Da	GUI	Da
Arachni	Automatizirano skeniranje	Linux Windows OS X	Da	WebUI	Da
Nikto	Automatizirano skeniranje	Unix bazirani sustavi	Da	CLI	Da

U sljedećim potpoglavljima će se pobliže razmotriti performanse alata i njihove mogućnosti, te potencijalni načini implementacije unutar razvojnog procesa *web* aplikacije.

6.1 Sinteza istraživanja provedenog nad odabranim alatima otvorenog koda

Odabir alata unutar ovog istraživanja temeljio se na potencijalnim primjenama alata za razvojni proces *web* aplikacije. Namjerno ranjiva *Mutilidae* aplikacija je odabrana kao okvirni pokazatelj mogućnosti ovih alata, a podaci o provedenim skeniranjima će služiti kao prijedlozi o načinu implementacije alata prije produkcije aplikacija kako bi se poboljšala njihova sigurnost. U tablici 2. mapirani su podaci o skeniranju *Mutilidae aplikacije.*

Tablica 2. Rezultati provedenog skeniranja nad Mutilidae aplikacijom

Ime alata	Broj pronađenih ranjivosti	Vrijeme skeniranja	Vrste ranjivosti koje su pronađene	Broj ranjivosti po razinama kritičnosti
OWASP Zap	44	30 minuta	• SQL umetanje, • XSS umetanje, • Napadi prijelaza direktorija, • Pogreške u sigurnosnoj konfiguraciji, • Otkrivanje osjetljivih podataka, • Nesigurna kontrola pristupa, • Otkrivanje izvornog koda i greški. Više informacija na slici 9.	**Visoka razina** 10 **Srednja razina** 12 **Niska razina** 13 **Informacijska razina** 9
Vega	198	18 minuta	• Pogreške u sigurnosnoj konfiguraciji, • SQL umetanje, • URL umetanje, • XSS umetanje, • Otkrivanje osjetljivih podataka, • Informacije o kolačićima, • Otkrivanje izvornog koda i greški. Više informacija na slici 14.	**Visoka razina** 15 **Srednja razina** 124 **Niska razina** 44 **Informacijska razina** 15

Arachni	390	9 sati i 28 minuta	• SQL umetanje, • XSS umetanje, • CSRF napadi, • OS i URL umetanje, • Otkrivanje nezaštićenih podataka, • Nesigurni kolačići, • Pogreške u sigurnosnoj konfiguraciji, • Nesigurna kontrola pristupa, • Neispravna autentikacija. Više informacija na slici 20.	**Visoka razina** 276 **Srednja razina** 21 **Niska razina** 77 **Informacijska razina** 16
Nikto	166	59 sekundi	• Otkrivanje nesigurnih kolačića, • Otkrivanje nezaštićenih informacija, • Otkrivanje nesigurnih konfiguracija, • Daljinsko izvršavanje naredbi, • Otkrivanje XST ranjivosti, Više informacija na slikama 25. i 26.	N/A

U prvom slučaju korišten je ZAP alat razvijen od strane OWASP organizacije. ZAP nudi mogućnosti automatskog i manualnog skeniranja. Putem automatskog skeniranja na *Mutilidae* aplikaciji ZAP je pronašao 44 vrste različitih ranjivosti u 30 minuta skeniranja. ZAP nudi mogućnosti manualnog skeniranja aplikacije što može biti korisno za penetracijsko testiranje u različitim koracima unutar razvojnog procesa aplikacije što ga čini jako fleksibilnim alatom. Uz to nudi mogućnosti generiranja izvješća, pristup temeljitim informacijama i dokumentaciji o ranjivostima i samom alatu, što programerima s jako malo iskustva u sigurnosnom testiranju može olakšati da aplikaciju učine sigurnijom.

U drugom slučaju korišten je *Vega* alat. *Vega* se pokazao kao veoma jednostavan alat za dinamičko skeniranje sigurnosti *web* aplikacija. Nudi veoma jednostavno korisničko sučelje i skeniranje što također može pomoći programerima pri analizi sigurnosti *web* aplikacije. *Vega* je u svega 18 minuta pronašao 198 različitih ranjivosti. Zajedno s mogućnostima skeniranja i presretanja *proxy* zahtjeva to mu

omogućava taktičku analizu sigurnosti *web* aplikacija, što se kasnije može iskoristiti za unakrsnu provjeru ranjivosti s ostalim sigurnosnim alatima.

U trećem slučaju korišten je *Arachni* sigurnosni okvir. *Arachni* nudi veliki niz mogućnosti i funkcionalnosti koji se mogu implementirati unutar projekta. To čini *Arachni* nešto složenijim alatom za korištenje, ali nudi prednosti kao što su automatizacija penetracijskog testiranja unutar cjelokupnog životnog ciklusa *web* aplikacije. Arachni je veoma temeljit alat, što pokazuje i sama činjenica da je automatsko skeniranje *Mutilidae* aplikacije trajalo 9 sati i 28 minuta, te je *Arachni* pronašao 390 različitih ranjivosti.

U četvrtom slučaju korišten je *Nikto* alat za skeniranje *web* poslužitelja *Mutilidae* aplikacije. *Nikto* je dovoljno jednostavan za inicijalno korištenje, ali je dovoljno fleksibilan za složenija skeniranja. U ovom slučaju *Nikto* je skenirao *web* poslužitelj svega 59 sekundi te je pronašao 166 potencijalnih ranjivosti. Iako Nikto nudi mogućnost automatskog skeniranja, zbog nedostatka korisničkog sučelja teže je pristupiti informacijama o ranjivostima.

6.2 Prijedlozi implementacije alata otvorenog koda unutar razvojnog procesa aplikacije

Bitno je napomenuti da sama sigurnost *web* aplikacije će ovisiti o više faktora. To može uključivati izbor tehnologija koje će se koristiti za izradu *web* aplikacije, stručnost i kompetentnost programera i ostalog osoblja, dostupni resursi, kompleksnost projekta i još niz drugih faktora.

Kada je u pitanju izbor tehnologija važno je birati tehnologije za izradu *web* aplikacije koje će biti sukladne veličini i kompleksnosti projekta, ali i sposobnostima programera koje će koristiti te tehnologije. Stoga je bitno temeljito istražiti mogućnosti pojedinih tehnologija kako bi se izbjegle nepotrebne komplikacije i odabrali adekvatni alati. Nakon odabira adekvatnih alata važno se pridržavati najboljih praksi pisanja izvornog koda, i prihvatljivih tehnika sigurnosnog kodiranja kako bi se izbjegle sigurnosne ranjivosti.

U ovom radu istraživani su dinamički skeneri sigurnosti *web* aplikacija stoga će se prijedlog implementacije alata otvorenog koda temeljiti na njima. No uz dinamičke

skenere, bitno je istražiti i razmotriti korištenje ostalih vrsta skenera za sigurnost, kao što su statički sigurnosni skeneri koji analiziraju izvorni kod.

Nakon istraživanja i odabira alata koji će se koristiti za sigurnosno testiranje već u početnim koracima dizajniranja aplikacije se mogu implementirati alati otvorenog koda. Prilikom prvog posluživanja aplikacije na poslužitelj *Nikto* se može koristiti kako bi se skenirale ranjivosti unutar aktivnog poslužitelja. Rezultati skeniranja se mogu koristiti za manualno otklanjanje ranjivosti, te je moguće držati dokumentaciju o izvješćima skeniranja nakon svake implementacije novih komponenti unutar *web* aplikacije na poslužitelj. Ovisno o kompleksnosti projekta i aplikacije, može postojati više poslužitelja i domena koje će se koristiti, stoga *Nikto* sigurnosno skeniranje omogućava način da se implementira sigurnosna revizija nad svim poslužiteljima unutar projekta.

U toku dizajniranja *web* aplikacije ZAP i *Vega* se mogu koristiti kako bi se skenirale ranjivosti unutar *web* aplikacije. Dizajniranje *web* aplikacije je kompliciran proces koji može uključivati pisanje velikog broja komponenti i funkcionalnosti unutar aplikacije. Putem ZAP alata moguće je manualno testirati nove komponente koje se implementiraju unutar aplikacije te zaključiti postoji li ranjivosti koji potencijalni napadači mogu iskoristiti kao vektor napada. To omogućava programerima utvrđivanje postoji li sigurnosnih propusta unutar izvornog koda prije same produkcije aplikacije, te na taj način im daje mogućnost da implementiraju bolji tehniku sigurnosnog kodiranja. Osim toga prednost kombiniranja više dinamičkih skenera sigurnosti unutar *web* aplikacije omogućava unakrsnu analizu rezultata skeniranja, što može dati bolji uvid u sigurnost *web* aplikacije.

Arachni alat može služiti kao sigurnosni okvir za potpuno sigurnosno skeniranje i praćenje sigurnosti *web* aplikacije. Kao alat *Arachni* nudi mogućnost automatizacije cijelog sigurnosnog sustava, uz to moguće je podešavanje performansi i iskorištavanje „inteligencije“ alata kako bi se dobili opsežni rezultati skeniranja u svrhu analize sigurnosti *web* aplikacije. To omogućava programerima da obave temeljitu reviziju sigurnosti *web* aplikacije nakon svake faze razvojnog procesa aplikacije. A uz dodatne mogućnosti kao što su pisanje novih modula i dodataka omogućava programerima da sigurnosno testiranje prilagode potrebama aplikacije i cjelokupnog projekta.

7. Zaključak

Internet je nedvojbeno postao svakodnevica velikom broju ljudi. Veliki broj *web* stranica se u posljednjih nekoliko godina redizajnirao u interaktivne *web* aplikacije koje privlače sve veći broj korisnika. *Web* aplikacije mogu imati različite svrhe, ali često im je zajednička značajka da traže određene podatke od korisnika kako bi se mogli služiti *web* aplikacijom, što kasnije u slučaju napada, može rezultirati negativnim posljedicama i za korisnika i za vlasnika aplikacije.

S popularnošću Interneta i *web* aplikacija, povećao se i rizik od kibernetičkih napada. Trendovi pokazuju da su napadi na *web* aplikacije u porastu, a jedini parametri koji se mijenjaju su načini na koji napadači iskorištavaju ranjivosti u *web* aplikacijama. S obzirom na već spomenutu popularnost često su i poslovanja primorana proširiti svoje poslovanje u obliku *web* aplikacija. Često raspolažući s ograničenim resursima sigurnost *web* aplikacija nije jedan od prioriteta. Sigurnosni alati otvorenog koda su često besplatni, stoga je u ovom radu bio cilj prikazati njihovu primjenjivost u svrhu poboljšavanja sigurnosti *web* aplikacija.

Kroz ovaj rad analizirana su četiri različita alata otvorenog koda ZAP, *Vega*, *Arachni* i *Nikto*. Ovi alati su odabrani zbog svoje dostupnosti i različitom načinu primjenjivanja, što za poslovanja s ograničenim resursima može biti izrazito povoljno rješenje. U petom poglavlju prikazane su mogućnosti alata i rezultati skeniranja. Na primjeru ranjive aplikacije alati su iskorišteni kako bi se mogla razumjeti njihova primjena u poboljšanju sigurnosti *web* aplikacija. Nakon toga u sljedećem poglavlju su mapirani podaci iz istraživanja ovih alata kako bi se bolje razumila njihova primjenjivost i donio prijedlog o implementaciji ovih alata za razvojni proces aplikacije.

Konačni zaključak je da sigurnost korisnika, njihovih podataka, ali i sigurnost same *web* aplikacije je vrlo važna, stoga je nužno zaštiti. Prije samog razvoja aplikacije potrebno je istražiti tehnologije koje će odgovarati funkciji i veličini aplikacije, primijeniti odgovarajuće sigurnosne tehnike pisanja izvornog koda, a alati otvorenog koda mogu biti primjenjivani u svrhu poboljšanja sigurnosti kroz razvojni proces aplikacije. Programeri pomoću sigurnosnih alata mogu na temelju rezultata skeniranja implementirati sigurnosne zaštite za sprječavanje napada i uklanjanje ranjivosti kako bi poboljšali sigurnost *web* aplikacija.

Literatura

[1] OWASP Top Ten, https://owasp.org/www-project-top-ten/ (pristupljeno 31. srpnja 2021.)

[2] ENISA Threat Landscape 2020 - Web application attacks, https://www.enisa.europa.eu/publications/web-application-attacks (pristupljeno 31. srpnja 2021.)

[3] Humayun, M., Niazi, M., Jhanjhi, N. et al. „Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study“ Arab J Sci Eng 45, 3171–3189 (2020). https://doi.org/10.1007/s13369-019-04319-2 (pristupljeno 31. srpnja 2021.)

[4] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almashfi, „Web Application Security Tools Analysis“ 2017 IEEE 3rd International Conference, 2017, pp. 237-242,doi:10.1109/BigDataSecurity.2017.47. https://ieeexplore.ieee.org/abstract/document/7980348 (pristupljeno 1. kolovoza 2021.)

[5] M. Baykara, „Investigation and Comparison of Web Application Vulnerabilities Test Tools“ Software Engineering Department & Firat University, Turkey, IJCSMC, Vol. 7, Issue. 12, December 2018, pg.197 – 212. https://www.academia.edu/38045666/Investigation_and_Comparison_of_Web_Application_Vulnerabilities_Test_Tools (pristupljeno 1. kolovoza 2021.)

[6] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almashfi, „Web Application Security Tools Analysis“ 2017 IEEE 3rd International Conference, 2017, pp. 237-242, doi:10.1109/BigDataSecurity.2017.47 https://ieeexplore.ieee.org/abstract/document/7980348 (pristupljeno 1. kolovoza 2021.)

[7] A. Suschevich, D. Birukova: „What Is a Technology Stack? Choosing the Right Tech Stack For Your Web Project“ Preuzeto sa: https://www.intexsoft.com/blog/post/tech-stack.html (pristupljeno 2. kolovoza 2021.)

[8] „2021 Developer Survey“ Preuzeto sa: https://insights.stackoverflow.com/survey/2021 (pristupljeno 5. kolovoza 2021.)

[9] „What is MEAN Stack?“ Preuzeto sa: https://www.mongodb.com/mean-stack (pristupljeno 5. kolovoza 2021.)

[10] Aggarwal, S. and Verma, J., 2018. „Comparative analysis of MEAN stack and MERN stack“ International Journal of Recent Research Aspects, 5(1), pp.127-32. http://www.ijrra.net/Vol5issue1/IJRRA-05-01-26.pdf (pristupljeno 5. kolovoza 2021.)

[11] Express.js službena stranica, Preuzeto sa: https://expressjs.com/ (pristupljeno 5. kolovoza 2021.)

[12] Angular.js službena stranica, Preuzeto sa: https://angular.io/ (pristupljeno 5. kolovoza 2021.)

[13] Node.js službena stranica, Preuzeto sa: https://nodejs.org/en/about/ (pristupljeno 5.kolovoza 2021.)

[14] „What is MERN Stack?“, Preuzeto sa: https://www.mongodb.com/mern-stack (pristupljeno 6.kolovoza 2021.)

[15] React.js službena stranica, Preuzeto sa https://reactjs.org/ (pristupljeno 6.kolovoza 2021)

[16] Peraković, D., Cvitić, I.: Sigurnost i zaštita informacijsko komunikacijskog sustava, predavanja iz kolegija Sigurnost i zaštita informacijsko komunikacijskog sustava, Fakultet prometnih znanosti, Zagreb, 2019.

[17] OWASP Top Ten:2017-Injection, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A1_2017-Injection (pristupljeno 8.kolovoza 2021)

[18] Napadi umetanjem SQL koda, Cis.hr Lipanj 2011. , Preuzeto sa: https://www.cis.hr/files/dokumenti/CIS-DOC-2011-09-025.pdf (pristupljeno 8. kolovoza 2021.)

[19] OWASP Top Ten: 2017 – Broken Authentication, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication (pristupljeno 9. kolovoza 2021.)

[20] OWASP Top Ten: 2017 – Sensitive Data Exposure, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure (pristupljeno 10. kolovoza 2021.)

[21] OWASP Top Ten: 2017 – XML External Entities (XXE), Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE) (pristupljeno 11. kolovoza 2021.)

[22] OWASP Top Ten: 2017 – Broken Access Control, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control (pristupljeno 11. kolovoza 2021.)

[23] OWASP Top Ten: 2017 – Security Misconfiguration, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration (pristupljeno 12. kolovoza 2021.)

[24] OWASP Top Ten: 2017 – Cross-Site Scripting (XSS), Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS) (pristupljeno 12. kolovoza 2021.)

[25] OWASP Top Ten: 2017 – Inescure Deserialization, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization (pristupljeno 14. kolovoza 2021.)

[26] OWASP Top Ten: 2017 – Using Components witk Known Vulnerabilites, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities (pristupljeno 15. kolovoza 2021.)

[27] OWSAP Top Ten: 2017 – Insufficient Logging and Monitoring, Preuzeto sa: https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring (pristupljeno 15. kolovoza 2021.)

[28] S. Bairwa, B. Mewara, J. Gajrani „Vulnerability Scanners: A Proactive Approach to Assess Web Application Security“ International Journal on Computational Science & Applications, 4.10.5121/ijcsa.2014.411 (2014.) Preuzeto sa: https://www.researchgate.net/publication/261182006_Vulnerability_Scanners-A_Proactive_Approach_To_Assess_Web_Application_Security (pristupljeno 23.kolovoza 2021.)

[29] OWASP Mutilidae II, Preuzeto sa: https://github.com/webpwnized/mutillidae (pristupljeno 24. kolovoza 2021.)

[30] XAMPP službena stranica, Preuzeto sa: https://www.apachefriends.org/index.html (pristupljeno 24. kolovoza 2021.)

[31] OWASP Zed Attack Proxy (ZAP) službena stranica, Preuzeto sa: https://www.zaproxy.org/ (pristupljeno 24. kolovoza 2021.)

[32] Vega službena stranica, Preuzeto sa: https://subgraph.com/vega/ (pristupljeno 25. kolovoza 2021.)

[33] Arachni službena stranica, Preuzeto sa: https://www.arachni-scanner.com/ (pristupljeno 26. kolovoza 2021.)

[34] Nikto službena stranica, Preuzeto sa: https://cirt.net/Nikto2 (pristupljeno 30.kolovoza 2021.)

Popis kratica

API	(Application Programming Interface) Sučelje za programiranje aplikacija
CLI	(Command Line Interface) Sučelje naredbenog retka
CSS	(Cascading Style Sheets) Stilski jezik za opis prezentacije dokumenta napisanog pomoću HTML-a
CSP	(Content Security Policy) Politika sigurnosti sadržaja predstavlja sigurnosni standard
DAST	(Dynamic Application Security Testing) Vrsta sigurnosnog testiranja metodom crne kutije
DOM	(Document Object Model) Sučelje za programiranje koje prikazuje dokumente u strukturnom obliku
GUI	(Graphical User Interface) Grafičko korisničko sučelje
HTML	(HyperText Markup Language) Prezentacijski jezik za izradu *web* stranica
JSON	(JavaScript Object Notation) Standardizirani format za razmjenu podataka
JSX	(JavaScript XML) Sintaksno proširenje za JavaScript programski jezik
LAMP	(Linux,Apache,MySQL,PHP/Perl/Python) Softverska skupina
LDAP	(Lightweight Directory Access Protocol) Softverski protokol za pristup autentičnosti putem imenika
MEAN	(MongoDB, Express.js, Angular.js, Node.js) Skupina tehnologija za izradu web aplikacija
MERN	(MongoDB, Express.js, React.js, Node.js) Skupina tehnologija za izradu web aplikacija
MVC	(Model-View-Controller) Uzorak dizajna za web aplikacije
OS	(Operating System) Operacijski sustav računala
OSVDB	(Open Source Vulnerability Database) Otvorena i neovisna baza podataka sigurnosnih ranjivosti
OWASP	(Open Web Application Security Project) Organizacija koja se bavi sigurnošću web aplikacija

PHP	(Hypertext Preprocessor) Programski jezik koji se koristi u razvijanju webrješenja
SAST	(Static Application Security Testing) Vrsta sigurnosnog testiranja metodom bijele kutije
SPA	(Single Page Application) Način dinamičkog prepisivanja trenutne web stranice novim podacima
SSL	(Secure Sockets Layer) Sigurnosni protokol za šifriranje
STMP	(Simple Mail Transfer Protocol) Komunikacijski protokol za prijenos elektroničke pošte
SQL	(Structured Query Language) Programski jezik za relacijske baze podataka
TLS	(Transport Layer Security) Kriptografski protokol za pružanje komunikacije putem mreže
URL	(Uniform Resource Locator) Referenca za mrežnu lokacija
URI	(Uniform Resource Identifier) Niz znakova koji identificiraju fizičke ili logičke resurse
WAMP	(Windows, Apache, MySQL, PHP) Tehnološka skupina
WebUI	(Web User Interface) Korisničko sučelje dostupno putem preglednika
XAMPP	(X , Apache, MySQL, PHP,Perl) Tehnološka skupina za više platformi (X označava više platformi)
XML	(eXtensible Markup Language) Programski jezik za označavanje podataka
XSS	(Cross Site Scripting) Vrsta kibernetičkih napada putem malicioznih skripti
ZAP	(Zed Attack Proxy) Sigurnosni alat OWASP organizacije

Popis slika

Popis tablica

Sveučilište u Zagrebu
Fakultet prometnih znanosti
10000 Zagreb
Vukelićeva 4

IZJAVA O AKADEMSKOJ ČESTITOSTI I SUGLASNOST

Izjavljujem i svojim potpisom potvrđujem kako je ovaj diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu što pokazuju korištene bilješke i bibliografija.

Izjavljujem kako nijedan dio rada nije napisan na nedozvoljen način, niti je prepisan iz necitiranog rada, te nijedan dio rada ne krši bilo čija autorska prava.

Izjavljujem također, kako nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Svojim potpisom potvrđujem i dajem suglasnost za javnu objavu diplomskog rada pod naslovom Istraživanje primjenjivosti alata otvorenog koda u svrhu unaprjeđenja sigurnosti web aplikacija na internetskim stranicama i repozitoriju Fakulteta prometnih znanosti, Digitalnom akademskom repozitoriju (DAR) pri Nacionalnoj i sveučilišnoj knjižnici u Zagrebu.

U Zagrebu, 09/09/2021

Student/ica:

(potpis)