Evaluacija sigurnosti web aplikacija

Bogut, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Bogut

EVALUACIJA SIGURNOSTI WEB APLIKACIJA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Bogut

Matični broj: 37847/02–I

Studij: *Poslovni sustavi*

EVALUACIJA SIGURNOSTI WEB APLIKACIJA

ZAVRŠNI RAD

Mentor/Mentorica:

Izv. prof. dr. sc. Petra Grd

Varaždin, veljača 2024.

Luka Bogut

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu će se spominjati vektori napada i načini zaštite koji se provode tokom izrade i održavanja samih aplikacija. Prvo će se obraditi vrste napada na web aplikacije i kada čitatelj razumije teoriju iza toga, rad bude prošao kroz alate koji se mogu koristiti za penetracijsko testiranje. Uz objašnjenje načina napada, biti će objašnjeni kako se koristi koji alat za penetracijsko testiranje i testiranje određenog vektora napada. Neki od napada koji su opisani su napad grubom silom, napadi vezani u dizajnu aplikacije, prisilno pregledavanje, falsificiranje parametara, manipulacija HTTP zaglavljem. Postoje i napadi koji utječu na bazu podataka, jedan od njih je napad ugniježđenim SQL naredbama i slijepi napad ugniježđenim SQL naredbama. Primjeri tih napada su i izvedeni u praktičnom dijelu rada. Prikazano je XXL IN, SQL injekcija, XSS napad, DDoS napad. Uz navedenu terminologiju rada navedeni su i načini obrane i zaštite od napada. Jedan od ključnih faktora obrane od napada su sigurnosti timovi, postoji šest vrsta timova koji imaju različita zaduženja u svrhu čuvanju povjerljivih informacija. Isto tako postoje i sigurnosne organizacije poput OWASP-a i EDB-a. Pobliže je opisan OWASP i mogućnosti koje sama organizacija nudi, a tijekom izrade su korišteni mnogi izvori s njihove službene stranice. Postojanje sigurnosnih timova, štoviše samih organizacija, upućuje na to kako je stavljanje zaštitnih mehanizama bitno i tijekom izrade i rada web stranica, i nakon što se već desi proboj u sustav.

Ključne riječi: web aplikacija, sigurnost, testiranje, automatizacija, napad, eksplotacija, timovi

Sadržaj

1. Uvod

U ovom radu je objašnjeno koje su procedure testiranja sigurnosti web aplikacija od kibernetičkih napada. Danas se iz dana u dan sve više ljudi spaja u svijet interneta, ogromne količine podataka prolaze svake sekunde i zato je dovoljan jedan proboj da se ugroze osobni podatci. Podatci su danas vrlo vrijedan izvor informacija, i mnogo različitih stranaka na legalnei ilegalne načine pokušava doći do njih. Budući da postoje razni načini napada online aplikacija, postoji i puno metoda koje se koriste za zaštitu tih podataka. Svrha ovoga rada je istraživanje metoda i vektora napada, te sigurnost tih aplikacija. Razvojni tim tokom izrade web aplikacija mora imati na umu koje stvari treba pozorno promotriti uzimajući u obzir kakve podatke korisnici ostavljaju u toj aplikaciji. Spominju se mogući vektori radi proučavanja načina napada jer znanje tih sigurnosnih propusta pomaže pri izradi alata i tehnika obrane od istih. Alati za napad su primarno operativni sustavi poput Kali Linuxa koji su napravljeni za tzv. penetracijsko testiranje (eng. Penetration Testing). Svrha penetracijskog testiranja je otkrivanje slabosnih točaka unutar aplikacija, pa će to biti način testiranja samih načina napada. Za primjere je korišteno Cross-Site Scripting, SQL injekcija i XXL IN.

1.1. Metode i tehnike rada

Za izradu ovog rada korišteni su podatci pronađeni na internetu. U svrhu razrade teme upotrebljeni su simulacijski programi kako bi se opisali koraci napada te obrana od njih. Dok se u praksi primjenjuje operacijski sustav Kali Linux radi zakonodavnih pravila mi ga nismo koristili. Kali Linux nije ilegalan za korištenje, dok god se njegove funkcionalnosti koriste u primarnu svrhu digitalne forenzike i za penetracijsko testiranje. Umjesto toga korišteni su neki simulacijski programi i web stranice poput acunetix acurat i PortSwigger.

2. Procedure testiranja sigurnosti web aplikacije

U današnje vrijeme, sigurnost web aplikacija je od iznimne važnosti. Sve više ljudi koristi internet za obavljanje različitih aktivnosti, od kupovine do upravljanja financijama. Stoga, testiranje sigurnosti web aplikacija postaje sve važnije kako bi se osiguralo da su podaci korisnika sigurni i zaštićeni. Testiranje sigurnosti web aplikacija uključuje proces provjere ranjivosti i pouzdanosti aplikacije. Cilj je identificirati moguće sigurnosne propuste i nedostatke u aplikaciji kako bi se otklonili prije nego što ih iskoriste zlonamjerni napadači. U radu su se prošle procedure koje se mogu koristiti kako bi se testirala sigurnost web aplikacije.

Postoji nekoliko osnovnih procedura koje se često koriste pri testiranju sigurnosti web aplikacija. Neke od ključnih procedura su identifikacija ciljeva, prikupljanje informacija, analiza ranjivosti, penetracijsko testiranje, testiranje autentikacije i autorizacije, testiranje sigurnosti podataka, procjena rizika, izrada izvještaja i preporuka.[1] Iako su neke detaljnije objašnjene kasnije, sada ćemo ukratko opisati svaku proceduru. Identifikacija ciljeva: određivanje jasnih ciljeva testiranja sigurnosti web aplikacije. Prikupljanje informacija: skupljanje relevantnih informacija o web aplikaciji, uključujući tehničke specifikacije, arhitekturu, postavke poslužitelja, autentikaciju i autorizaciju, ulaze i izlaze podataka, vanjske resurse i integracije, kao i pravila sigurnosti i pristupa.[2] Analiza ranjivosti: Koriste automatske skenere ranjivosti ili ručno provjeravaju aplikaciju kako bi identificirali moguće ranjivosti. Ovo uključuje provjeru na ranjivosti kao što su SQL injection, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) i ostale sigurnosne propuste.[3]

Penetracijsko testiranje: aktivno penetracijsko testiranje aplikacije da se otkrije i iskoristi ranjivosti; uključuje testiranje autentikacije, upravljanje sesijom, izvršavanje neovlaštenih operacija, manipulaciju podacima,...["Metasploit: The Penetration Tester's Guide" by David Kennedy] Testiranje autentikacije i autorizacije: da bi se provjerili jesu li mehanizmi pravilno implementirani i osigurani od napada. Kao na primjer jačinu lozinke, postoje li slabosti u rukovanju sesijama i dozvoljavanju pristupa resursima.[4]

Testiranje sigurnosti podataka: kako aplikacija rukuje osjetljivim podacima; uključuje provjeru sigurnosti baze podataka, enkripciju podataka u prijenosu, zaštitu od curenja podataka i druge aspekte vezane uz sigurnost podataka.[5] Procjena rizika: Procijenjuju se rizici povezani s identificiranim ranjivostima; ocjena koliko je ozbiljna svaka ranjivost i koliko bi utjecala na sigurnost i integritet aplikacije. Ovo će pomoći u prioritetizaciji popravaka i

poboljšanja.[5] Izrada izvještaja i preporuka: Nakon završetka testiranja priprema se detaljan izvještaj o otkrivenim ranjivostima, rizicima i preporukama za ispravke.[6]

2.1. Skeniranje ranjivosti

Skeniranje ranjivosti je proces automatskog pretraživanja web aplikacije kako bi se pronašli potencijalni sigurnosni propusti. Ova metoda se često koristi kao početni korak u testiranju sigurnosti web aplikacija, kako bi se identificirali poznati sigurnosni propusti, poput SQL injekcija, Cross-Site Scripting (XSS) i drugih ranjivosti koje bi mogle biti iskorištene od strane napadača. Na tržištu postoji mnogo alata i oni mogu predprogramirano simulirati napade na web aplikaciju da se vidi gdje postoje sigurnosne rupe unutar aplikacije. To su alati poput Acunetix, Burp Suite, Nikto i drugih. Većina ovih alata omogućuje korisnicima da konfiguriraju postavke skeniranja, kao što su vrste ranjivosti koje se traže, dubina skeniranja i drugo. Važno je napomenuti da svaka aplikacija ima svoje specifične ranjivosti, stoga je važno provesti temeljito testiranje sigurnosti kako bi se identificirale sve ranjivosti i ispravile prije nego što budu iskorištene od strane napadača.[7]

2.2. Penetracijsko testiranje

Penetracijsko testiranje je proces testiranja web aplikacije koji uključuje aktivno testiranje sigurnosti aplikacije simuliranjem napada iz stvarnog svijeta. Ova vrsta testiranja simulira napade koje bi potencijalni napadači mogli koristiti kako bi pronašli slabosti u sustavui iskoristili ih u cilju neovlaštenog pristupa ili krađe podataka. Kada se penetracijsko testiranje primjenjuje na web aplikacije, cilj je identificirati ranjivosti i slabosti koje bi napadači mogli iskoristiti. Ovdje su ključni koraci uključeni u provođenje testa prodora na web aplikaciji[8]:

1. Planiranje i definicija opsega: definiranje ciljeva, opsega i pravila angažmana za penetracijski test. Treba se razumjeti arhitekturu aplikacije, korištene tehnologije i potencijalne rizike, te koje vrste testova se treba izvesti (npr. testiranje crne kutije, bijele kutije ili sive kutije) i razinu pristupa dodijeljenu testerima.
2. Izviđanje: prikupljanje informacija o ciljanoj web aplikaciji, kao što je njena URL struktura, krajnje točke, informacije o poslužitelju, tehnologije i sve druge javno dostupne pojedinosti. Te se informacije mogu dobiti tehnikama kao što su pretraživanje weba, DNS enumeracija i mrežno skeniranje. Ove informacije mogu biti korisne za identificiranje slabosti u aplikaciji.

3. Identifikacija ranjivosti: provedba sustavne procjene web aplikacije kako bi identificirali potencijalne ranjivosti. To može uključivati ispitivanje korisničkog sučelja, provjeru skripti, provjeru autorizacije, provjeru izvršavanja SQL upita i sl. Uobičajene ranjivosti koje treba potražiti uključuju SQL injection, cross-site scripting (XSS), cross-site request krivotvorenje (CSRF), nesigurne izravne reference na objekte te nedostatke autentifikacije i autorizacije.
4. Iskorištavanje: Pokušaj iskorištavanja identificiranih ranjivosti za dobivanje neovlaštenog pristupa, povećanje privilegija ili manipuliranje funkcionalnošću aplikacije. Ovaj korak uključuje aktivno testiranje postojećih sigurnosnih kontrola i provjeru utjecaja uspješnih eksploatacija. Nakon što se identificiraju ranjivosti, testeri pokušavaju iskoristiti te ranjivosti za pristup sustavu. To može uključivati pokušaj izvršavanja zlonamjernog koda, ubrizgavanje SQL upita, krađu lozinki i sl.
5. Nakon iskorištavanja i bočno kretanje: Ako je početno iskorištavanje uspješno, dodatno se procjenjuje sigurnost aplikacije pokušajem bočnog pomicanja unutar sustava, eskalacije privilegija ili pristupa osjetljivim podacima. To pomaže u procjeni opsega potencijalnog kompromisa i učinkovitosti segmentacije i kontrola pristupa.
6. Dokumentacija i izvješćivanje: Dokumentira se sve nalaze, uključujući identificirane ranjivosti, tehnike iskorištavanja i njihov učinak. Priprema se opsežno izvješće koje daje jasan pregled ranjivosti, njihove razine rizika i preporučene mjere za ublažavanje. Uključuju se detaljni koraci za reprodukciju problema kako bi programerima pomogli u razumijevanju i učinkovitom rješavanju ranjivosti.
7. Ispravljanje i ponovno testiranje: Podijela rezultata i konzultacije s relevantnim sudionicima, poput razvojnog tima ili administratora sustava. Radi se na rješavanju i saniranju identificiranih ranjivosti. Nakon što su popravci implementirani, provedi se ponovno testiranje kako bi se potvrdilo da su ranjivosti učinkovito riješene.

Postoje i mnogi alati i tehnike koje se koriste u penetracijskom testiranju web aplikacija. To uključuje alate za otkrivanje ranjivosti, kao što su Burp Suite, OWASP ZAP i Acunetix, te tehnike kao što su SQL injekcija, Cross-Site Scripting i DDoS napadi.[3] Važno je napomenuti da se penetracijsko testiranje uvijek mora provoditi s dozvolom vlasnika sustava, kako bi se spriječilo neovlašteno testiranje i moguće zakonske posljedice.[9]

2.3. Testiranje autentifikacije

Autentifikacija se odnosi na postupak provjere identiteta korisnika, obično traženjem da daju valjane kredencijale. Napadač može biti zlonamjerni ovlašteni korisnik ili napadač koji koristi ukradene vjerodajnice izdane valjanom korisniku.[10] Cilj ovog testiranja je osigurati da su podaci korisnika zaštićeni i da im se ne može pristupiti ili ih izmijeniti bez odgovarajućih ovlasti. Postoje različite metode testiranja autentifikacije, a neke od njih su:

1. Testiranje snage lozinke: Ova metoda uključuje provjeru koliko je teško pogoditi ili razbiti korisničku lozinku. S postavljanjem minimalnih zahtjeva za duljinu lozinke, korištenjem složenih kombinacija znakova (velika i mala slova, brojevi, posebnih znakova) te provjere ima li ograničenja učestalosti promjene lozinke može se popravitit jačina lozinke i otežati neovlašten upad u profil. Tester može koristiti alate za ispitivanje lozinke kako bi otkrio slabosti, kao što su lozinke koje su previše jednostavne ili se lako mogu pogoditi. [11]
2. Testiranje metoda autentifikacije: Ova metoda se fokusira na provjeru sigurnosti metoda autentifikacije koje se koriste u aplikaciji, kao što su jednokratni tokeni, biometrijske metode, dvofaktorska autentifikacija itd. [12]
3. Testiranje vremenskih ograničenja: Ova metoda provjerava ograničenja vremena koja se primjenjuju na korisničke sesije kako bi se spriječio neovlašten pristup nakon što korisnik napusti aplikaciju. Tester može pokušati produžiti vrijeme sesije ili pristupiti aplikaciji nakon što je sesija završila.[11]
4. Testiranje postupaka za oporavak lozinke: Ova metoda se fokusira na testiranje postupaka za oporavak lozinke kako bi se osiguralo da se lozinka može resetirati samo od strane ovlaštene osobe. [11]
5. Provjera protiv CSRF napada: CSRF (Cross-Site Request Forgery) napad iskorištava povjerenje web preglednika korisnika kako bi izvršio neželjene radnje u ime korisnika na drugim web stranicama. Provjerava se ima li aplikacija odgovarajuće zaštite protiv CSRF napada, poput korištenja anti-CSRF tokena u zahtjevima.
6. Provjera blokiranja neuspjelih prijava: provjera je li sustav autentifikacije postavljen da blokira račune nakon određenog broja neuspjelih pokušaja prijave. Ovo pomaže u sprječavanju Brute Force napada i pokušaja probijanja lozinke.
7. Provjera protiv XSS napada: XSS napad omogućuje umetanje zlonamjernog koda u web stranice koje korisnici pregledavaju. Provjerava se je li aplikacija zaštićena od XSS napada, što uključuje provjeru jesu li korisnički unos sanitiziran i/ili enkodiran prije prikaza na stranici.

Primjer testiranja autentifikacije u web aplikaciji bio bi pokušaj napada na račun korisnika pomoću lažne autentifikacije. Tester bi pokušao pristupiti računu korisnika bez znanja lozinke i koristeći neku drugu vrstu autentifikacije koju aplikacija omogućava, kao što su odgovori na tajno pitanje, prikupljanje podataka s društvenih mreža ili preuzimanje lozinke pohranjene u pregledniku. Ako tester uspije u ovoj vrsti napada, to bi značilo da postoji propust u autentifikaciji aplikacije koji bi trebao biti popravljen kako bi se zaštitili korisnički podaci.[8]

3. Prijetnje sigurnosti web aplikacija

U današnje vrijeme, sigurnost web aplikacija je od iznimne važnosti. Sve više ljudi koristi internet za obavljanje različitih aktivnosti, od kupovine do upravljanja financijama. Stoga, testiranje sigurnosti web aplikacija postaje sve važnije kako bi se osiguralo da su podaci korisnika sigurni i zaštićeni. Programeri web aplikacija i organizacije moraju zauzeti proaktivan pristup sigurnosti kako bi ublažili ove prijetnje. To uključuje strogu provjeru valjanosti unosa, korištenje sigurnosnih biblioteka i okvira(eng. Framework), redovita sigurnosna testiranja i revizije, ažuriranje sigurnosnih zakrpa, edukacija programera i korisnika o mogućim opasnostima. Za zaštitu mrežnih aplikacija od stalno promjenjivog okruženja sigurnosnih prijetnji potrebna je kombinacija preventivnih mjera, tehnika otkrivanja i strategija odgovora na incidente. [13] [14]

3.1. Napadi vezani za autentifikaciju

3.1.1. Napad grubom silom

Napad grubom silom metoda je koja se koristi u računalnoj sigurnosti za probijanje lozinki ili algoritama šifriranja sustavnim pokušajem svih mogućih kombinacija znakova dok se ne pronađe ispravna. To je iscrpan i dugotrajan pristup koji se oslanja na pretpostavku da je ispravna lozinka ili ključ za šifriranje unutar skupa svih mogućih kombinacija. U kontekstu razbijanja lozinki, napad grubom silom uključuje isprobavanje svake moguće lozinke dok sene otkrije ispravna. To obično zahtijeva automatiziranu skriptu ili program koji generira i testira velik broj kombinacija zaporki.[15] Proces uključuje ponavljanje kroz različite kombinacije znakova, kao što je isprobavanje svih mogućih kombinacija slova, brojeva i posebnih znakova. Učinkovitost napada grubom silom ovisi o nekoliko čimbenika, uključujući složenost i duljinu ciljane lozinke. Dulje i složenije lozinke općenito su otpornije na napade grubom silom, budući da se broj mogućih kombinacija eksponencijalno povećava sa svakim dodatnim znakom. Na primjer, lozinka od 6 znakova s malim slovima i brojevima imala bi 36^6 (oko 2,18 milijardi) mogućih kombinacija. Kako bi se smanjio rizik od napada grubom silom, obično se koristi nekoliko sigurnosnih mjera. To uključuje provođenje jakih politika zaporki koje potiču korisnike da odaberu dulje i složenije zaporke, implementaciju zaključavanja računa ili odgode nakon određenog broja neuspjelih pokušaja prijave i korištenje dodatnih sigurnosnih slojeva kao što je autentifikacija s više faktora. Vrijedno je napomenuti da se napadi grubom silom također mogu primijeniti na algoritme šifriranja, gdje napadač sustavno testira sve moguće ključeve šifriranja dok se ne pronađe ispravan. Međutim, moderni algoritmi šifriranja dizajnirani su da

izdrže napade grubom silom korištenjem ključeva koji su dovoljno dugi da broj mogućih kombinacija čine računski neizvedivim za testiranje unutar razumnog vremenskog okvira.[15]

3.1.2. Napadi na propuste u dizajnu autentikacije

Napadi na propuste u dizajnu autentikacije su vrsta napada koji se fokusiraju na način koji se autentikacijski sustavi dizajniraju i implementiraju. Ti propusti mogu omogućiti napadačima da zaobiđu ili kompromitiraju autentikacijske mehanizme i dobiju neovlašten pristup računalnom sustavu, aplikaciji ili podacima.[16] Nekoliko primjera napada na propusteu dizajnu autentikacije: Napad ponovnog reproduciranja (Replay Attack): U ovom napadu, napadač snima i reproducira legitimne autentikacijske podatke koji su već korišteni za uspješnu autentikaciju. Napadač može iskoristiti ranije snimljene autentikacijske podatke kako bi zaobišao provjeru identiteta i pristupio sustavu ili aplikaciji.[17] Napad usporedbom vremena (Timing Attack): Ovaj napad koristi razlike u vremenskom odzivu autentikacijskog sustava kako bi otkrio ispravne autentikacijske podatke. Napadač analizira vrijeme odziva sustava na različite autentikacijske zahtjeve te otkriva razlike u vremenskom odzivu koje ukazuju na točne ili netočne autentikacijske podatke.[18] Napad korištenjem ranjivosti protokola (Protocol Vulnerabilities Attack): U nekim slučajevima, autentikacijski protokoli mogu imati ranjivosti koje omogućuju napadačima da zaobiđu ili kompromitiraju autentikaciju. Primjeri uključuju ranjivosti u protokolima kao što su SSL/TLS, LDAP, Kerberos i drugi.[19] Napad na upravljanje sesijama (Session Management Attack): Ovaj napad uključuje iskorištavanje slabosti u načinu upravljanja i provjeravanja autentikacijskih sesija. Napadač može iskoristiti ranjivosti kao štosu predvidljivi identifikatori sesija, nevaljane ili slabije provjere autentikacije prilikom obnove sesije, ili presretanje ili krađa valjanih sesijskih tokena. [20]

Da bi se zaštitili od napada na propuste u dizajnu autentikacije, važno je primijeniti najbolje prakse u dizajnu autentikacijskih sustava. To uključuje korištenje snažnih i složenih lozinki, primjenu dvofaktorske ili višefaktorske autentikacije, redovito ažuriranje autentikacijskih protokola i softvera kako bi se ispravile poznate ranjivosti, i provjeru identiteta kroz sigurne metode.

3.2. Napadi vezani uz autorizaciju

3.2.1. Prisilno pregledavanje

Prisilno pregledavanje, također poznato kao obilazak direktorija ili obilazak putanje, napad je na web aplikacije koji uključuje pristup neovlaštenim direktorijima ili datotekama manipuliranjem ulaznih parametara URL-a. Iskorištava nepravilnu provjeru valjanosti unosa ili nedostatne mehanizme kontrole pristupa na web poslužitelju. Napadač identificira ciljnu web aplikaciju i počinje analizirati njezinu strukturu, tražeći direktorije, datoteke ili resurse koji bi mogli biti skriveni ili ograničeni za normalne korisnike.[21] Napadač mijenja URL ili ulazne parametre u pokušaju pristupa direktorijima ili datotekama izvan predviđenog opsega. To može uključivati korištenje "../" ili drugih posebnih znakova za navigaciju u hijerarhiji direktorija. Izvršenje zahtjeva: manipulirani URL šalje se web poslužitelju koji obrađuje zahtjev. Ako poslužitelj ne uspije pravilno potvrditi ili kontrolirati korisnički unos, može vratiti traženu datoteku ili direktorij koji bi trebao biti nedostupan. Ako je napad uspješan, napadač može doći do osjetljivih informacija kao što su konfiguracijske datoteke, izvorni kod, baze podataka ili drugi resursi koji nisu bili namijenjeni za otkrivanje. To može dovesti do curenja informacija, neovlaštenog pristupa ili daljnjeg iskorištavanja ranjivosti.

Kako bi se spriječili napadi prisilnog pregledavanja, web aplikacije trebaju implementirati odgovarajuću provjeru valjanosti unosa i kontrole pristupa. Provjerite i očistite sve korisničke unose, uključujući parametre URL-a, kako biste spriječili upotrebu posebnih znakova ili nizova koji bi se mogli koristiti za obilazak direktorija.[22] Implementirajte kontrole pristupa i provjere dopuštenja na strani poslužitelja kako biste osigurali da korisnici mogu pristupiti samo direktorijima i datotekama za koje su ovlašteni gledati. Primijenite načelo najmanje privilegije, dopuštajući pristup samo prema potrebi. Osigurajte da su osjetljive datoteke, direktoriji i resursi ispravno zaštićeni i da im se ne može izravno pristupiti putem web poslužitelja. Konfigurirajte poslužitelj da ograniči pristup osjetljivim informacijama. Implementirajte odgovarajuće mehanizme za rukovanje pogreškama koji ne otkrivaju osjetljive informacije korisniku u porukama o pogrešci. Navedite generičke poruke o pogrešci umjesto specifičnih detalja koji bi mogli pomoći napadačima. Primjenom ovih sigurnosnih mjera, web aplikacije mogu ublažiti rizik od napada prisilnog pregledavanja i zaštititi osjetljive informacije od neovlaštenog pristupa.

3.2.2. Falsifikacija parametara

Falsificiranje parametara, također poznato kao petljanje parametara ili manipulacija parametrima, vrsta je napada u kojem napadač mijenja ulazne parametre ili podatke poslane web aplikaciji s ciljem zaobilaženja sigurnosnih kontrola, manipuliranja ponašanjem aplikacijeili dobivanja neovlaštenog pristupa resursima.

U napadima falsificiranjem parametara, napadač obično cilja na polja za unos, URL parametre, podatke obrasca ili skrivena polja unutar web aplikacije.[23] Promjenom ovih parametara napadač može pokušati razne zlonamjerne radnje. Evo nekoliko primjera, promjenama vrijednosti parametara napadač mijenja vrijednosti ulaznih parametara kako bi prevario aplikaciju da izvrši neželjene radnje. Na primjer, mijenjanje razine povlastica korisnika promjenom parametra koji specificira ulogu korisnika. Manipuliranjem vrijednostima parametara, napadač može zaobići ili onemogućiti te provjere valjanosti, dopuštajući mu da pošalje zlonamjerne podatke koji bi inače bili blokirani. Pristup neovlaštenim resursima mijenja parametre koji se odnose na identifikatore resursa, kao što su ID-ovi zapisa ili imena datoteka, napadač može pokušati pristupiti osjetljivim informacijama ili resursima za koje nisu ovlašteni pregledavati ili manipulirati njima. Lažno predstavljanje korisnika omogućuje neovlašteno mijenjanje parametara može se koristiti za lažno predstavljanje drugih korisnika mijenjanjem parametara koji identificiraju ili provjeravaju autentičnost korisnika. To može dovesti do neovlaštenog pristupa korisničkim računima, manipulacije korisničkim podacima ili izvođenja radnji u ime drugog korisnika.

Načini da se ublaže rizici povezani s napadima lažiranja parametara, važno je primijeniti sljedeće sigurnosne mjere, provjera valjanosti unosa je sigurnosna mjera koja implementira snažnu provjeru valjanosti unosa na strani poslužitelja kako bi osigurali da se parametri ispravno provjeravaju za njihove očekivane vrijednosti, formate i raspone. Potvrđuje se i dezinficira korisnički unos kako bi se spriječili obradu zlonamjernih podataka. Kontrola pristupa na strani poslužitelja provodi odgovarajuće kontrole pristupa kako bi se spriječio neovlašteni pristup resursima. Autentifikacija i autorizacija se primjenjuju da korisnik temeljem njihovih uloga i dopuštenja izvodi radnje i manipulacije osjetljivim informacijama. Sigurnosno upravljanje sesijom implementira sigurno upravljanje poput generiranje jedinstvenih ID-ova sesije, šifriranje podataka sesije i korištenje sigurnih kolačića, kako bi se spriječila otmica sesije ili napada neovlaštenog mijenjanja sesije. Bilježenje i praćenje koristi mehanizme bilježenja za bilježenje i praćenje aktivnosti korisnika, uključujući vrijednosti parametara, kakootkrit potencijalne napade i odgovorili na njih. Redovito pregledavanje i analiziranje zapisnika za bilo kakvu sumnjivu ili nepravilnu aktivnost. Implementacijom ovih sigurnosnih praksi, web

aplikacije mogu smanjiti rizik od napada krivotvorenjem parametara i zaštititi integritet, povjerljivost i dostupnost svojih resursa i korisničkih podataka.

3.2.3. Manipulacija HTTP zaglavlja

Manipulacija HTTP zaglavljem je web sigurnosna ranjivost u kojoj web aplikacija dinamički gradi zaglavlja od korisničkog unosa. HTTP radi na modelu zahtjeva/odgovora. Kada korisnik zatraži resurs od web poslužitelja, web poslužitelj odgovara. HTTP zaglavlja koriste se za traženje potrebnih resursa. Zaglavlja se mogu podijeliti u dvije skupine. Zaglavlja za zahtjev i odgovor. Kada je korisnički unos uključen u HTTP odgovor, javlja se ranjivost.[24]

HTTP zaglavlja dio su HTTP protokola i koriste se za prijenos dodatnih informacija zajedno sa zahtjevima i odgovorima. Zaglavlja pružaju pojedinosti kao što su vrsta sadržaja, upute za predmemoriju, vjerodajnice za provjeru autentičnosti, kolačići i još mnogo toga. Nekoliko primjera manipulacije HTTP zaglavljem:

- Kod manipulacije kolačićima napadač može modificirati zaglavlje "Cookie" kako bi se promijenilia vrijednosti kolačića sesije, dopuštajući im da oponašaju druge korisnike, izvrše otmicu sesije ili dobiju neovlašteni pristup zaštićenim resursima.

- Spoofing korisničkog agenta zahtjeva mijenjanje zaglavlja "User-Agent", napadači mogu prevariti web poslužitelj da povjeruje da zahtjev dolazi od drugog korisničkog agenta (npr. drugog preglednika ili uređaja). Najčešće se primjenjuje za zaobilaženje sigurnosnih kontrola preglednika ili uređaja.

- Manipulacijom Content-Type napadač može manipulirati zaglavljem "Content-Type" kako bi prevaro web poslužitelj da pogrešno postupa s dolaznim podacima. Na primjer, napadač može poslati zlonamjerni kod s lažnom vrstom sadržaja kako bi zaobišao ograničenja učitavanja datoteka ili izvršio kod na poslužitelju.

- Manipulacija kontrolom predmemorije se manipulirana zaglavlje "Cache-Control" ili "Expires" kako bi manipulirali ponašanjem predmemorije i prisilili klijenta ili proxy poslužitelje da predmemoriraju osjetljive ili zastarjele informacije.

- Utjecaj napada manipulacije HTTP zaglavlja može varirati ovisno o specifičnoj ranjivosti koja se iskorištava i kontekstu web aplikacije. Potencijalne posljedice uključuju neovlašteni pristup, manipulaciju podacima, curenje informacija, otmicu sesije, skriptiranje između stranica (XSS) i daljinsko izvršavanje koda.

Kako bi se spriječio napad manipulacije HTTP zaglavlja, bitno je implementirati sigurne prakse kodiranja u navedenim smjernicama. [25]

Provjerom valjanosti i dezinfekcijom unosa omogućuje ispravno provjeravanje i dezinficiranje korisničkog unosa, uključujući HTTP zaglavlja, kako bi se spriječilo uključivanje zlonamjernog sadržaja ili neočekivanih znakova. Sigurno upravljanje sesijom je tehnika u kojoj se koriste metode poput šifriranja podataka sesije, korištenje sigurnih kolačića i provjera valjanosti zaglavlja povezanih sa sesijom kako bi spriječilo manipulaciju sesije. Stroga provjera vrste sadržaja implementira stroge provjere zaglavlja "Content-Type" kako bi se osigurali da učitane datoteke ili podaci odgovaraju očekivanoj vrsti i spriječili izvršavanje zlonamjernog koda. Sigurnosna zaglavlja primjenjuju odgovarajuća sigurnosna zaglavlja, poput "Strict-Transport-Security" (HSTS), "X-Frame-Options" i "Content-Security-Policy," kako bi poboljšali sigurnost web aplikacije i ublažili određene vrste napada. Slijedom ove najbolje prakse i održavajući web aplikacije ažurnim sigurnosnim zakrpama, programeri mogu minimizirati rizik od napada manipulacije HTTP zaglavlja i održati integritet i sigurnost svojih sustava.

3.3. Napadi na baze podataka

3.3.1. Napad ugniježđenim SQL naredbama

Ugniježđeni SQL napad, poznat i kao napad SQL injekcije temeljen na podupitu, tehnika je koju napadači koriste za iskorištavanje ranjivosti u upitima baze podataka web aplikacije. U tipičnom napadu SQL injekcijom, napadač manipulira ulaznim parametrima kako bi ubacio zlonamjerni SQL kod u upit baze podataka aplikacije, što često dovodi do neovlaštenog pristupa, manipulacije podacima ili curenja informacija. Međutim, ugniježđeni SQL napad ide korak dalje umetanjem podupita unutar glavnog upita.[26]

Evo pojednostavljenog primjera za ilustraciju ugniježđenog SQL napada:
Ranjivi kod: Web aplikacija dinamički konstruira SQL upit spajanjem korisničkih unosa bez odgovarajuće provjere ili sanacije
query = "SELECT * FROM users WHERE username = '" + userInput + "';"
Ubacivanje podupita: Napadač manipulira unosom kako bi uključio podupit koji izvršavadodatne SQL naredbe unutar glavnog upita.
userInput = "' OR 1=1; SELECT * FROM sensitive_data;"

Manipulirano izvršavanje upita
SELECT * FROM users WHERE username = '' OR 1=1; SELECT * FROM sensitive_data;
Napadačev umetnuti podupit (SELECT * FROM sensitive_data) izvršava se unutar glavnog upita, potencijalno dovodeći do preuzimanja osjetljivih podataka iz baze podataka.

Ugniježđeni SQL napadi mogu imati ozbiljne posljedice, uključujući neovlašteni pristup podacima, otkrivanje osjetljivih informacija, manipulaciju podacima, au nekim slučajevima i potpuno ugrožavanje aplikacije ili temeljne baze podataka.

3.3.2. Slijepi napad ugniježđenim SQL naredbama

Slijepo SQL ubacivanje s ugniježđenim SQL izjavama, također poznato kao slijepo SQL ubacivanje drugog reda, napredna je tehnika koju koriste napadači za iskorištavanje ranjivostiu web aplikacijama koje nisu odmah vidljive ili reagiraju na napadača:

- Slijepa SQL injekcija s Booleovim vrijednostima

Vrsta napada gdje napadač šalje SQL upite koji vraćaju Booleovu (true ili false) vrijednost. Na temelju odgovora aplikacije napadač odlučuje je li upit bio uspješan.

- Slijepa SQL injekcija temeljena na vremenu

Napadači ubacuju SQL upite koji odgađaju izvršenje (na primjer, korištenjem metode sleep()). Ako aplikaciji treba više vremena da odgovori, to znači da je ubrizgavanje bilo uspješno. [27]

Ako se ne riješi na odgovarajući način, slijepa SQL injekcija može rezultirati neželjenim pristupom, gubitkom podataka, pa čak i daljinskim izvršavanjem koda. Da bi se izbjegla SQL injekcija, koriste se pripremljene izjave ili parametrizirani upiti. Za prepoznavanje i sprječavanje pokušaja injekcije SQL-a koristi se vatrozid web aplikacije (WAF).Potvrdite i očistite korisnički unos. Smanjuju se privilegije korisnika baze podataka na apsolutni minimum koji je potreban aplikaciji.

4. Timovi

Pojmovi Crveni tim, Plavi tim i Ljubičasti tim često se koriste u kontekstu web sigurnosti za opisivanje različitih pristupa procjeni i poboljšanju sigurnosti sustava organizacije. Dok se pojmovi Crveni tim, Plavi tim i Ljubičasti tim često koriste u web sigurnosti, pojmovi Žuti tim, Narančasti tim i Zeleni tim nisu tako široko prepoznati ili standardizirani. Međutim, u nekim se kontekstima ovi pojmovi mogu koristiti za označavanje dodatnih uloga i odgovornosti unutar područja web sigurnosti.[28]

4.1. Crveni tim

Ukratko crveni tim odgovoran je za provođenje napadačkih sigurnosnih procjena. Njihov glavni cilj je simulacija napada iz stvarnog svijeta i pokušaj probijanja obrane organizacije. Oni koriste različite tehnike, alate i metodologije za prepoznavanje ranjivosti i njihovo iskorištavanje. Crveni timovi imaju niz atributa koji ih odvajaju od ostalih napadačkih sigurnosnih timova. [28] Najvažniji među njima su: Emulacija TTP-ova koje koriste protivnici s kojima će se meta vjerojatno suočiti, npr. korištenje sličnih alata, eksploatacija, okretnih metodologija i ciljeva kao dani akter prijetnje. Testiranje temeljeno na kampanji koje traje dulje vremensko razdoblje, npr. više tjedana ili mjeseci oponašanja istog napadača. Cilj Crvenog tima je pružiti realističnu procjenu sigurnosnog položaja organizacije, identificirati slabosti i pomoći u poboljšanju obrane.[29]

4.2. Plavi tim

Plavi tim odgovoran je za obrambene sigurnosne mjere. Usredotočeni su na održavanje i poboljšanje sigurnosti sustava i mreža organizacije. Plavi tim prati i analizira mrežni promet, upravlja sigurnosnim kontrolama (vatrozid, sustavi za detekciju upada itd.) i reagira na sigurnosne incidente. Oni rade na sprječavanju, otkrivanju i odgovoru na kršenja sigurnosti te se aktivno brane od potencijalnih napada.[28] Postoji niz obrambeno orijentiranih InfoSec zadataka za koje se općenito ne smatra da su dostojni plavog tima npr. Tier-1 SOC analitičar koji nije obučen niti zainteresiran za napadačke tehnike, niti ga zanima sučelje koje traže, i bez kreativnosti u traženju novih metoda napada i obrane. Razlike između plavog i ostalih timova jest, proaktivnost nasuprot reaktivnom načinu razmišljanja, beskrajna znatiželjau pogledu stvari koje su neobične i stalno poboljšanje detekcije i odgovora. [29]

4.3. Ljubičasti tim

Ljubičasti tim je suradnički pristup koji kombinira elemente Crvenog i Plavog tima. To uključuje blisku suradnju i razmjenu informacija između crvenog i plavog tima. Ljubičasti tim koristi ofenzivne sposobnosti Crvenog tima kako bi testirao i potvrdio učinkovitost obrambenih mjera koje provodi Plavi tim. Ova suradnja omogućuje kontinuirano poboljšanje i optimizaciju sigurnosnih kontrola organizacije i procesa odgovora na incidente.[28]

4.4. Žuti tim

Žuti tim se često povezuje s proaktivnim i preventivnim pristupom sigurnosti. Oni su usredotočeni na prepoznavanje i rješavanje sigurnosnih rizika prije nego što se mogu iskoristiti. Žuti tim može provoditi procjene rizika, skeniranja ranjivosti i sigurnosne revizije kako bi osigurao da su odgovarajuće sigurnosne mjere na snazi. Cilj im je proaktivno ublažiti potencijalne ranjivosti i održati sigurno okruženje.[28]

4.5. Narančasti tim

Narančasti tim obično se odnosi na tim koji se fokusira na obavještavanje o prijetnjamai nadzor sigurnosti. Njihova glavna odgovornost je prikupljanje i analiza informacija o prijetnjama u nastajanju, ranjivostima i tehnikama napada. Narančasti tim je u tijeku s najnovijim sigurnosnim trendovima i daje uvid Plavom timu za učinkovite obrambene strategije. Oni također mogu biti uključeni u lov na prijetnje, odgovor na incidente i povećanje svijesti o sigurnosti organizacije.[28]

4.6. Zeleni tim

Zeleni tim često se povezuje s usklađenošću i upravljanjem unutar područja web sigurnosti. Osiguravaju da se organizacija pridržava relevantnih zakona, propisa i industrijskih standarda koji se odnose na zaštitu i sigurnost podataka. Zeleni tim može razvijati i provoditi sigurnosne politike i procedure, obavljati revizije usklađenosti i blisko surađivati s pravnim i regulatornim tijelima. Njihov fokus je na održavanju sigurnog i usklađenog okruženja.

Važno je napomenuti da se uloge i odgovornosti povezane sa žutim, narančastim i zelenim timovima mogu razlikovati ovisno o organizaciji i specifičnom kontekstu u kojem se koriste. Ovi pojmovi nisu tako univerzalno priznati ili standardizirani kao crveni, plavi i ljubičasti timovi, ali mogu pružiti dodatne perspektive i funkcije unutar cjelokupnog sigurnosnog okvira.[28]

5. OWASP

Open Web Application Security Project (OWASP) je neprofitna zaklada koja daje smjernice o tome kako razviti, kupiti i održavati pouzdane i sigurne softverske aplikacije. OWASP je poznat po svojoj popularnoj Top 10 listi sigurnosnih ranjivosti web aplikacija. OWASP je projekt zajednice otvorenog koda posvećen poboljšanju sigurnosti web aplikacija i softvera. OWASP pruža resurse, alate i smjernice za pomoć organizacijama i programerima u razumijevanju i rješavanju sigurnosnih problema web aplikacija.[30]

Jedna od ključnih misija OWASP-a je učiniti vidljivom sigurnost softvera i osnažiti organizacije i pojedince da donose informirane odluke o sigurnosti aplikacija. Projekt ima za cilj educirati, podići svijest i pružiti praktične smjernice o sigurnosti web aplikacija. OWASP pokreće globalna zajednica stručnjaka za sigurnost, programera, entuzijasta i volontera. Projekt potiče suradnju, razmjenu znanja i sudjelovanje putem različitih kanala, uključujući konferencije, lokalne podružnice, popise za slanje e-pošte, forume i suradničke projekte.[30]

OWASP-ova inicijativa koja je jedna od važnijih jest OWASP Top 10, popis najkritičnijih sigurnosnih rizika web aplikacija. OWASP Top 10 pruža prioritetni pregled uobičajenih ranjivosti, kao što su napadi ubrizgavanjem, skriptiranje na različitim mjestima (XSS), neispravna autentifikacija i upravljanje sesijom i više. [30] Organizacija pruža opsežnu biblioteku resursa, uključujući smjernice, varalice, najbolje prakse, prakse sigurnog kodiranja, alate za testiranje i sigurnosne okvire. Ovi resursi pokrivaju različite aspekte sigurnosti web aplikacija, uključujući arhitekturu, testiranje, modeliranje prijetnji i sigurne metodologije razvoja.

Ugošćuju brojne projekte i alate koje je razvila zajednica kako bi pomogli identificirati, ublažiti i upravljati sigurnosnim rizicima web aplikacija. Neki popularni projekti uključuju OWASP ZAP (Zed Attack Proxy) za testiranje sigurnosti web aplikacije, OWASP Dependency-Check za identificiranje poznatih ranjivosti u ovisnostima softvera i OWASP WebGoat za učenje o uobičajenim sigurnosnim ranjivostima putem namjerno nesigurne web aplikacije. OWASP surađuje s industrijskim organizacijama, vladinim agencijama i drugim sigurnosnim inicijativama za promicanje sigurnih praksi razvoja softvera i podizanje svijesti o sigurnosti web aplikacija. Projekt aktivno surađuje s programerima, stručnjacima za sigurnost i dionicima iz različitih sektora kako bi potaknuli usvajanje praksi sigurnog kodiranja. OWASP-ovi resursi i pristup vođen zajednicom učinili su ga vrijednim izvorom znanja i smjernica u području sigurnosti web aplikacija. Programeri i organizacije mogu iskoristiti resurse OWASP-a da

razumiju uobičajene ranjivosti, implementiraju najbolju sigurnosnu praksu i poboljšaju cjelokupno sigurnosno stanje svojih web aplikacija.[30]

6. Baza podataka o eksploataciji (EDB)

Baza podataka o eksploataciji (EDB) popularno je mrežno spremište koje služi kao sveobuhvatna zbirka eksploatacija i ranjivosti. Pruža platformu za sigurnosne istraživače, testere penetracije i hakere za dijeljenje i pristup informacijama o poznatim ranjivostima i povezanim iskorištavanjima.

Primarna svrha baze podataka o eksploataciji je osigurati centraliziranu lokaciju na kojoj sigurnosni istraživači i hakeri mogu podijeliti svoja otkrića koja se odnose na softverske ranjivosti i eksploatacije. Baza podataka sadrži ogromnu kolekciju eksploatacija, shell kodova, radova i povezanih izvora.[31] To uključuje kod za dokaz koncepta, okvire iskorištavanja i tehničke detalje o ranjivostima, što korisnicima omogućuje razumijevanje i potencijalno repliciranje iskorištavanja. Baza podataka o ranjivostima uključuje detaljne informacije o ranjivostima softvera, kao što su pogođeni softver, brojevi verzija, vrsta ranjivosti (npr. prekoračenje međuspremnika, ubacivanje SQL-a) i opis problema. Za mnoge ranjivosti navedene u bazi podataka o iskorištavanju, postoji popratni kod za iskorištavanje. Ovaj kod pokazuje kako napadač može iskoristiti ranjivost za postizanje neovlaštenog pristupa ili izvođenje zlonamjernih radnji na ciljanom sustavu. Baza podataka Exploit organizira svoj sadržaj pomoću sustava klasifikacije. Eksploatacije i ranjivosti kategorizirani su na temelju pogođenog softvera, platforme, vrste ranjivosti i drugih relevantnih atributa. To korisnicima omogućuje traženje specifičnih ranjivosti ili iskorištavanja od interesa. Exploit Database je projekt kojim upravlja zajednica, a oslanja se na doprinose sigurnosnih istraživača i hakera iz cijelog svijeta. Pojedinci mogu predati svoja otkrića u bazu podataka, proširujući njezinu širinu i dubinu pokrivenosti. Baza podataka Exploit potiče suradnju i razmjenu znanja među stručnjacima za sigurnost. Služi kao platforma za rasprave i razmjenu ideja o ranjivostima, iskorištavanjima i tehnikama koje se koriste u testiranju prodora i etičkom hakiranju. Exploit Database usko je povezana s Metasploit Frameworkom, popularnim alatom za testiranje penetracije otvorenog koda. Metasploit Framework često uključuje eksploatacije iz baze podataka o eksploataciji, omogućujući stručnjacima za sigurnost da automatiziraju testiranje i provjeru ranjivosti. Baza podataka Exploit podržava prakse odgovornog otkrivanja podataka. To znači da se istraživače potiče da obavijeste pogođene dobavljače o ranjivostima i daju im razumnu količinu vremena za razvoj zakrpa ili ublažavanja prije nego što javno objave pojedinosti. Dok je Exploit Database vrijedan resurs za sigurnosne istraživače i stručnjake, važno je napomenuti da se njezin sadržaj može koristiti zlonamjerno. Organizacije bi trebale nadzirati bazu podataka o iskorištavanju i druge slične resurse kako bi bile informirane o

novonastalim ranjivostima i poduzele proaktivne mjere za njihovo zakrpanje ili ublažavanje. Vrijedno je spomenuti da iako Exploit Database može biti vrijedan alat za sigurnosno istraživanje i podizanje svijesti, treba je koristiti odgovorno i unutar granica važećih zakona i etičkih razloga.

7. Ranjivosti nultog dana

Ranjivosti nultog dana odnose se na ranjivosti softvera koje su nepoznate dobavljaču softvera ili programeru, što ih čini nezakrpanima i podložnima iskorištavanju. Izraz "nulti dan" podrazumijeva da programeri imaju nula dana da poprave ili ublaže ranjivost nakon što se otkrije. Ove ranjivosti privlačne su napadačima jer se mogu koristiti za izvođenje zlonamjernih aktivnosti bez znanja dobavljača softvera ili pogođenih korisnika. Definicija o ranjivostima nultog dana:
„Ranjivost nultog dana je greška ili slabost softvera, hardvera ili firmvera koja je nepoznata dobavljaču ili programeru. Predstavlja sigurnosnu rupu koju napadači mogu iskoristiti za neovlašteni pristup, pokretanje napada ili kompromitiranje sustava.“[32]

Zero-day ranjivosti obično otkrivaju sigurnosni istraživači, neovisni hakeri ili kibernetički kriminalci. Otkriće se može dogoditi tijekom rutinskih sigurnosnih revizija, ciljanog istraživanja ranjivosti ili slučajno. To znači da nema dostupne službene zakrpe ili popravka kada se ranjivost inicijalno iskorištava. Napadači iskorištavaju ranjivosti nultog dana stvaranjem iskorištavanja ili specijaliziranih alata za napad koji iskorištavaju specifičnu slabost u softveru. Ove se eksploatacije mogu koristiti za isporuku zlonamjernog softvera, dobivanje neovlaštenog pristupa, povećanje privilegija ili kompromitiranje sustava. Ranjivosti nultog dana nude napadačima značajnu prednost jer su nepoznate i nezakrpane. Kao rezultat toga, napadi koji iskorištavaju ranjivosti nultog dana često su prikriveni i teško ih je otkriti korištenjem tradicionalnih sigurnosnih mjera. Nakon što se otkrije i iskoristi zero-day ranjivost, dobavljač softvera može izdati hitnu zakrpu ili sigurnosno ažuriranje za rješavanje problema. Ovo ograničava prozor izloženosti tijekom kojeg su sustavi ranjivi. Međutim, dok zakrpa nije dostupna, sustavi ostaju u opasnosti. Ranjivosti nultog dana posebno su vrijedne za napadače, budući da pružaju konkurentsku prednost u infiltraciji sustava prije nego što se obrana može ojačati. Vlade, obavještajne agencije i organizacije kibernetičkog kriminala često su zainteresirane za stjecanje i iskorištavanje ranjivosti nultog dana. Odgovorno otkrivanje: kada istraživač otkrije ranjivost nultog dana, ima mogućnost slijediti prakse odgovornog otkrivanja. To obično uključuje privatno obavještavanje pogođenog dobavljača ili programera, dopuštajući im razdoblje odgode za razvoj i izdavanje zakrpe prije nego što se pojedinosti o ranjivosti javno objave. Masovni "zero-day" napad usmjeren na popularni web-preglednik Google Chrome iznenadio je stručnjake za kibernetičku sigurnost i korisnike interneta 2022. Ova zero-day ranjivost, koja je u početku bila nepoznata javnosti, uzdrmala je internetsku zajednicu dopustivši neprijateljskim akterima da hakiraju preglednik kodirati i ugroziti korisničke podatke

i integritet sustava. Hakiranje je dano kao oštar podsjetnik prijetnji digitalnog doba koji se neprestano mijenja.[33]

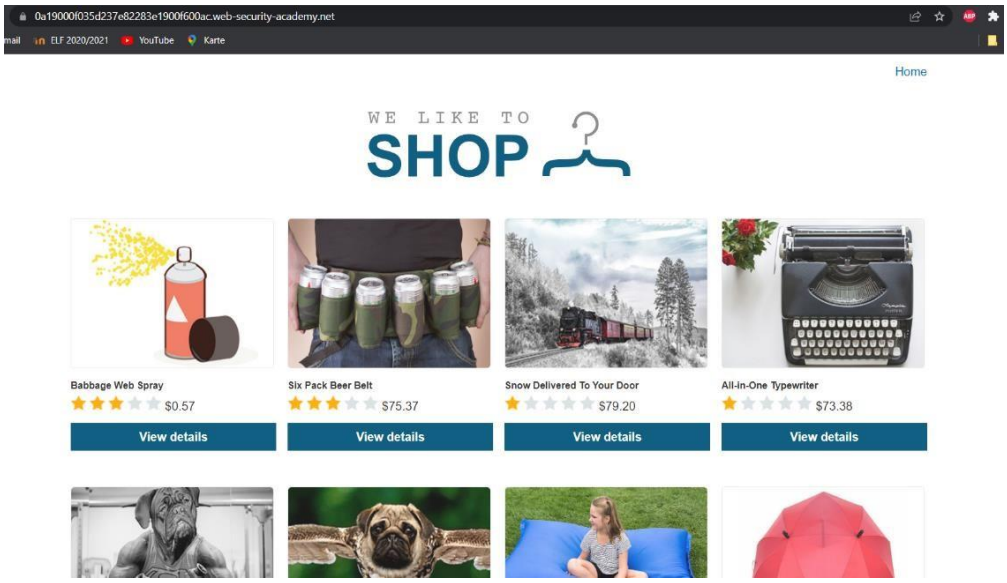
Postoji uspješno podzemno tržište za ranjivosti nultog dana, gdje hakeri i istraživači sigurnosti prodaju svoja otkrića najboljem ponuditelju. Ove ranjivosti mogu postići značajne cijene i od kriminalnih subjekata i od obavještajnih agencija. Za obranu od ranjivosti nultog dana, organizacije bi trebale usvojiti slojeviti pristup sigurnosti. To uključuje proaktivnu inteligenciju prijetnji, nadzor mreže, sustave za otkrivanje upada, analizu temeljenu na ponašanju i korištenje najboljih sigurnosnih praksi, kao što su redovita ažuriranja softvera i zakrpe. Važno je napomenuti da su ranjivosti nultog dana ozbiljan problem, ali predstavljaju mali dio ukupnog sigurnosnog okruženja. Redovita sigurnosna ažuriranja, jake sigurnosne prakse i informiranje o novim prijetnjama ključni su za održavanje sigurnog okruženja

8. Simulacije napada i zaštite

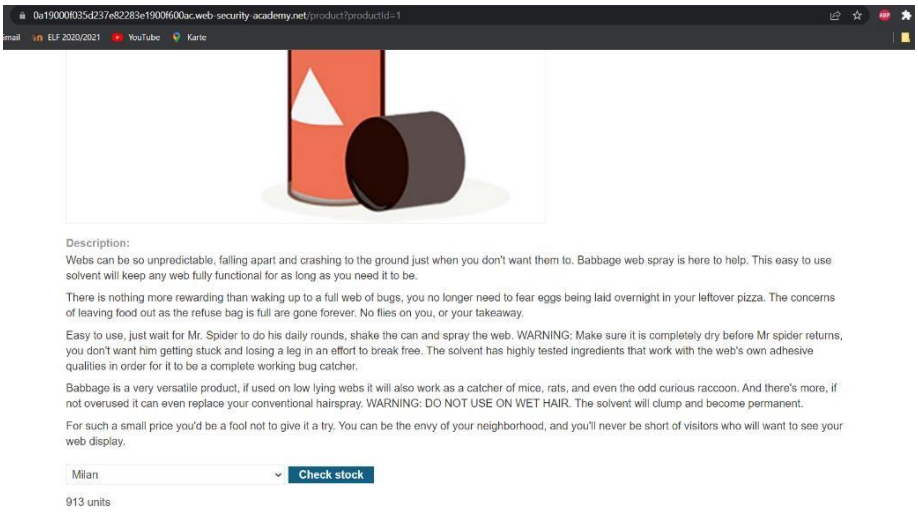
Demonstracije napravljene u ovome završnomu radu kao što su DDOS napad, SQL Injection, i XSS napad, prikazan je ovdje da takve malverzacije nisu tolika znanstvena fantastika koji se vidi u „hakerskim“ filmovima nego samo malo poznavanja alata i rada računala bilo to s lošom namjerom kako bi mogao iskorištavati slabosti web aplikacija u negativne svrhe. U navedenim primjerima su prikazani napadi ali na aplikacijama koje su striktno napravljene za vježbu cyber sigurnosti bez negativnih efekta. Primjer vrlo učinkovito iskorištenog napada bio kada su hakeri napali Yahoo! 2012 godine koristeći SQL injection, time uzeli više od pola milijuna email-ova i lozinki povezanih sa Yahoo!. Masivni DDoS napad bio je usmjeren na Dyn, glavnog pružatelja usluga DNS-a, u listopadu 2016. Ovaj je napad bio razoran i stvorio je poremećaj za mnoge velike stranice, uključujući Airbnb, Netflix, PayPal, Visa, Amazon, The New York Times, Reddit i GitHub koje je učinjeno pomoću zlonamjernog softvera pod nazivom Mirai.[34] Metode prikazane u radu su odabrane jer ih nije teško napraviti i to neka upozorava čitatelja da pri izradi web aplikacija, pogotovo onih koje drže osjetljive podatke treba paziti na puno mogućih vektora ulaza.

8.1. XXL IN

U ovom primjeru bit će prikazan XXE injection(XML External Entity). Ovaj XML napad je osmišljen tako da se pošalje POST zahtjev određenoj web stranici i u tijelu zahtjeva šalje zlonamjerni kod. Za ovaj primjer koristili smo pokusnu stranicu od proizvođaća PortSwigger. PortSwigger je tvrtka koja se bavi zaštitom web aplikacija. Ovaj primjer je izveden na njihovoj stranici koristeći njihov software „Burp Suite“ koji je napravljen za sigurnosnu penetraciju web aplikacija. Ovakav napad počinje tako da se u programu „Burp Suite“ uključi mogučnost presretanja zahtjeva koji putuju između odabranog servera i našeg Računala. Kada je hvatanje uključeno klikne se na bilo koji proizvod, pošalje se forma, čisto da Burp Suite uhvati par zahtjeva.

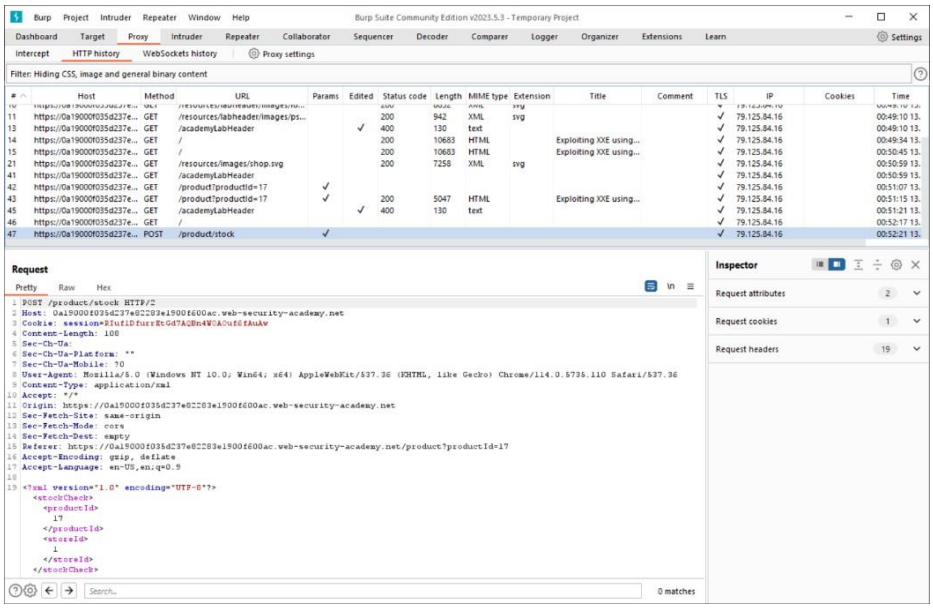


Slika 1 Pokusna stranica za testiranje



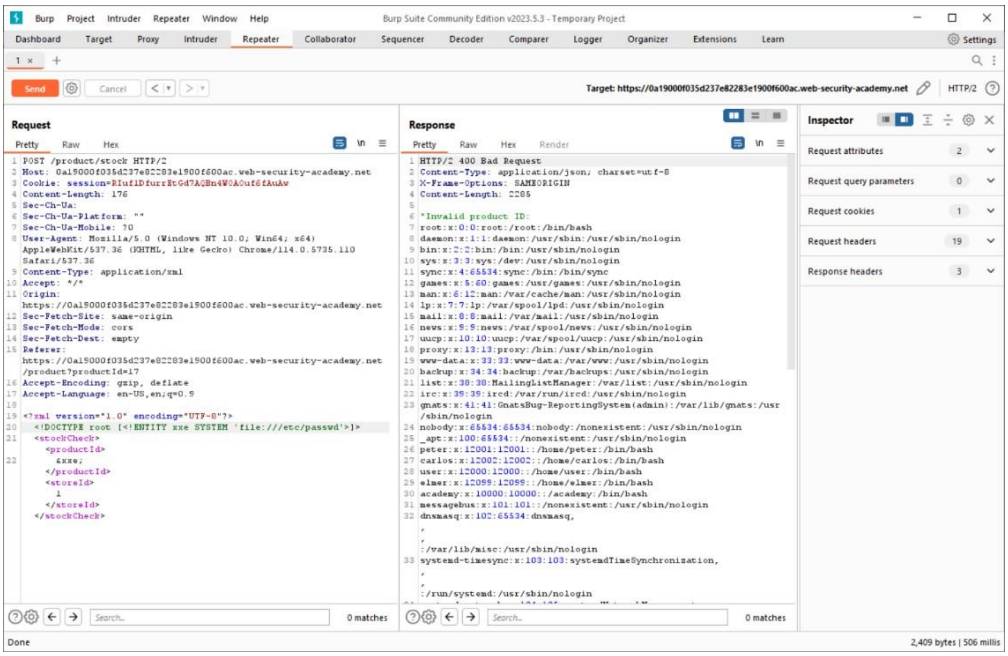
Slika 2 Opis procedure napada

Kada Burp Suite presretne komunikaciju između servera i računala, pronađe se POST zahtjev i njega se prebaci u Repeater pa se taj zahtjev može slati direktno iz programa umjesto da svaki put moramo replicirati početno ponašanje da napravimo promjenu na teretu od zahtjeva. U tijelu zahtjeva vidimo XML koji se šalje u tijelu i u koji ćemo dodati naš payload koji će sadržavati entitet sa naredbom po našoj želji, u ovom slučaju da vrati podatak.



Slika 3. Prikaz tijela post zahtjeva

Nakon što smo prebacili zahtjev u repeater, tamo vidimo cijelu glavu i tijelo zahtjeva. U tijelu zahtjeva vidimo podatke koji se šalju na server i napisani su u XML obliku. Teret, točnije sam Entitiy koji šaljemo na server se nadopisuje to jest nadodamo ga u tijelo zahtjeva. U ovome slučaju !DOCTYPE *tagu* definiramo što entitet „xxe“ radi, a to je da na server šalje file:///etc/passwd što vraća datoteku koja sadrži skup svih lozinki. Kada se napiše sav payload, stisne se „Send“ koji se nalazi u gornjem lijevom kutu. Time šaljemo naš payload na server i dobivamo odgovor nazad. U odgovoru se nalazi bad request što štiti podatke od prikazivana direktno na web stranici no oni su sadržani u tijelu odgovora. Uz to server vraća da atribut „ID“ nije valjan, a to se dešava jer smo u ID-ju poslali poslali naš entitet umjesto nekakve očekivanje cjelobrojne vrijednosti.



Slika 4 Server vraća grešku

8.2. SQL injekcija

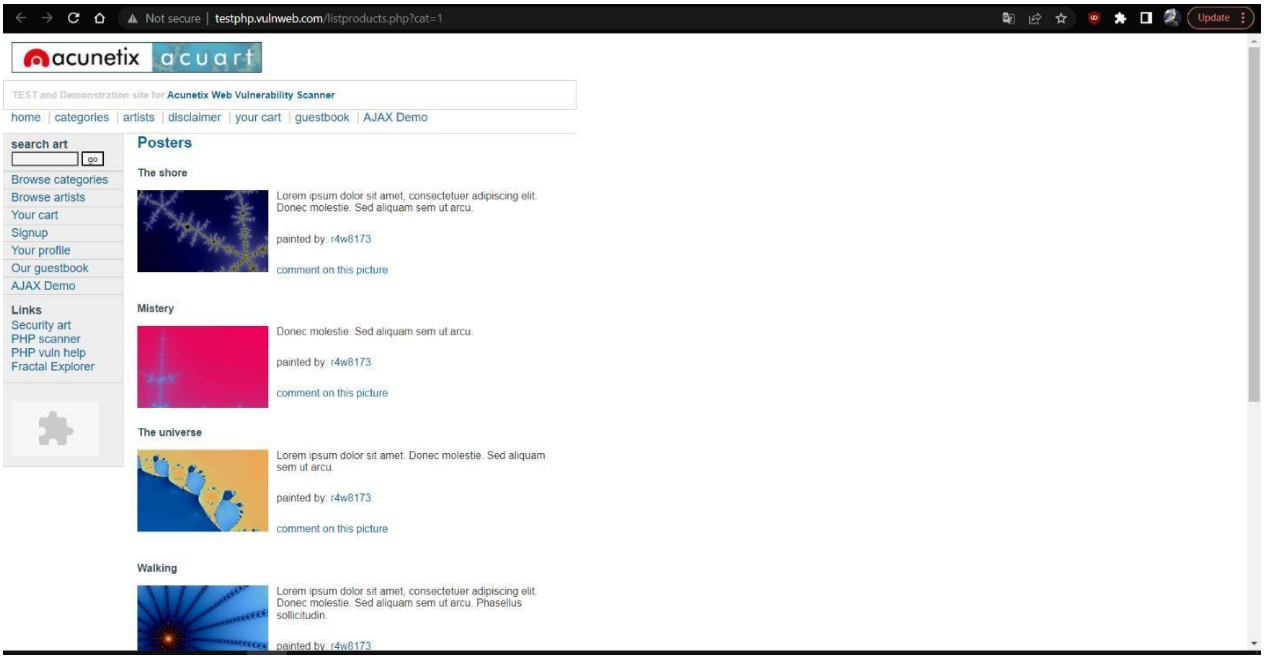
Što je to točno SQL injekcija? To je ranjivost koja je rezultat toga da date napadaču sposobnost da utječe na SQL upite koji kroz aplikaciju prolaze nazad u bazu podataka.[35] Ovaj propust se javlja kada web aplikacija ne provjerava ili ne filtrira korisnički unos prije nego što ga koristi za izvršavanje SQL upita. Kao rezultat toga, napadač može ubaciti SQL kôd koji se izvršava na bazi podataka, što može dovesti do krađe podataka, brisanja podataka ili drugih sigurnosnih problema. Procedura SQL injekcije je da napadač u polje za unos ili za pretraživanje unese niz znakova koji mijenja koji se SQL upit šalje na server i mijenja podatke koji se prikazuju. Tipični SQL upit izgleda ovako:

SELECT * FROM users WHERE username = 'korisnicko_ime' AND password = 'lozinka';

Napadač može iskoristiti ovu funkciju za SQL injekciju ako u polje za korisničko ime unese sljedeći niz: „' OR 1=1--“ što mijenja SQL upit koji se šalje na server pa server vrati podatke slušajući ovaj upit: „SELECT * FROM users WHERE username = '' OR 1=1--' AND password = 'lozinka';“. Ovaj upit će u web aplikaciji vratiti sve retke podataka što će prikazati osjetljive korisničke podatke.

Sql injekcija će biti prikazana preko acunetix acuart. Acunetix je firma koja se bavi izradom penetracijskog softvera I softvera za zaštitu aplikacija. Stranica http://testphp.vulnweb.com/index.php je stranica bazirana na php- koju je napravila firma

acunetix I napravljena je u svrhu demonstracije softvera **Acunetix Web Vulnerability** skenera. http://testphp.vulnweb.com/listproducts.php?cat=1 je link koji će se koristiti u program sqlmap i bit će prikazano kako je stranica ranjiva.



Slika 5. Acunetix Web Vanurabilitiy skener

Ovdje je prikazana komandna linija pokrenuta u terminalu za sqlmap. Dijelovi naredbe su python I sqlmap koje označavaju da se u python-u pokrene datoteka sqlmap.py, nakon nje slijedi link sa parametrom cat=1 koji se prosljeđuje na daljnju stranicu. Pokretanje naredbe sqlmap testira moguće načine penetracije koristeći sql I zapisuje rezultat na lokalno računalo. Prva faza se sastoji od prikupljanja informacija o ciljanoj web aplikaciji. To može uključivati otkrivanje URL-ova, identifikaciju parametara koji mogu biti podložni SQL injekciji i analizu strukture baze podataka ako je moguće. Zatim faza dva nakon prikupljanja informacija, Sqlmap koristi različite tehnike i heuristike kako bi automatski pokušao otkriti ranjivosti SQL injection. Alat šalje posebno oblikovane SQL upite u parametre web aplikacije i analizira odgovore kako bi pronašao znakove ranjivosti. Faza iskorištavanja dolazi kada je ranjivost pronađena, Sqlmap može automatski generirati i izvršiti SQL upite kako bi iskoristio ranjivost. To može uključivati dobivanje osjetljivih informacija, manipulaciju baze podataka ili čak izvođenje proizvoljnog koda na server.

Slika 6 Naredba za sql map

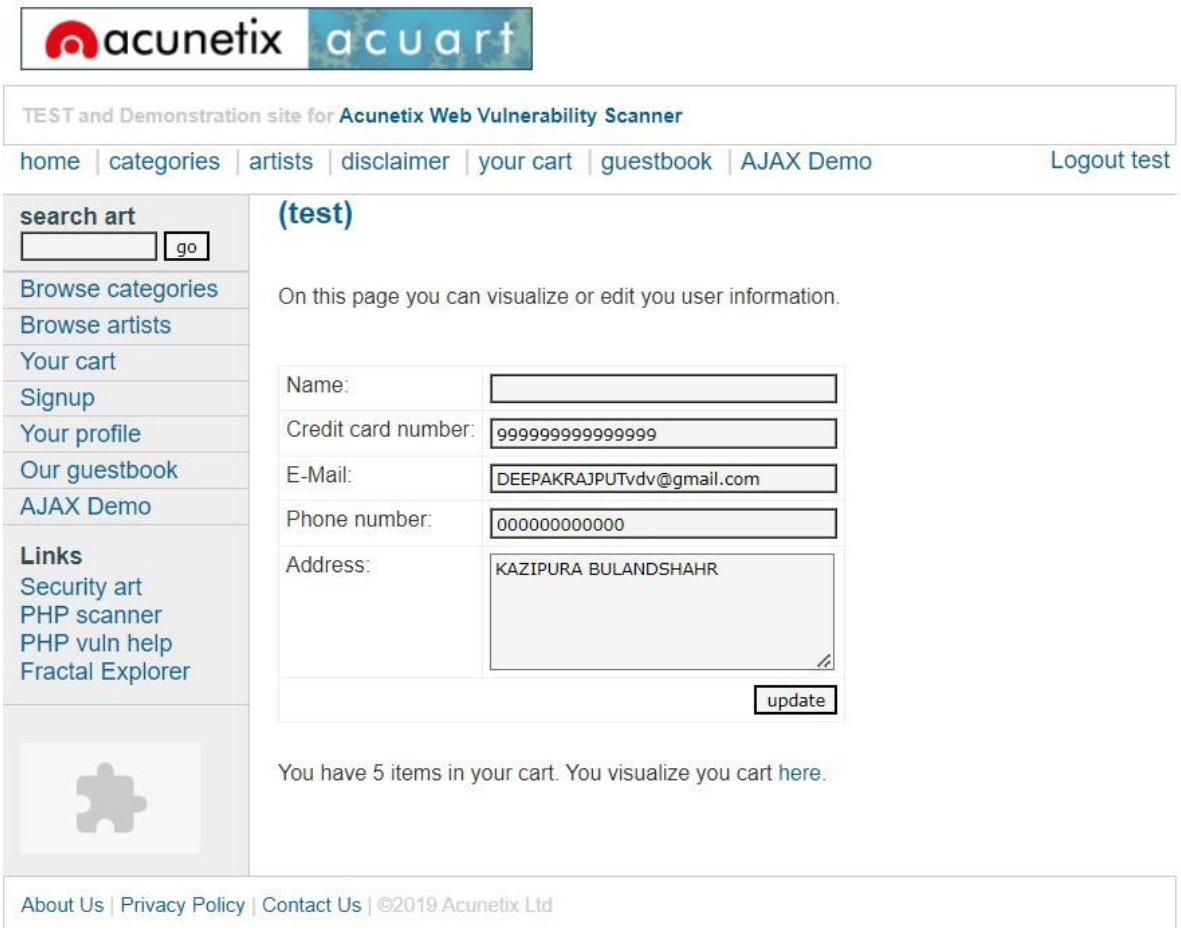
```
sqlmap identified the following injection point(s) with a total of 48 HTTP(s) requests:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 3331=3331

    Type: error-based
    Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
    Payload: cat=1 AND GTID_SUBSET(CONCAT(0x717a706b71,(SELECT (ELT(6923=6923,1))),0x7178787871),6923)

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: cat=1 AND (SELECT 6038 FROM (SELECT(SLEEP(5)))nDgH)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT NULL,CONCAT(0x717a706b71,0x784b7043576750427050 6d706a706f45724f68776f4974624367484a694a63716d57766d7174566d,0x7178787871),NULL,NULL,NULL,NULL,NULL,NULL
---
```

Slika 7 Output Sqlmapa-a

Na slici 7 vidljiv je output programa sqlmap i prikazuje moguće tehnike heuristike koje bi se mogle iskoristiti za manipulaciju bazama podataka i prikazivanje osjetljivih informacija. Budući da postoje te naredbe, to znači da je web aplikacija ranjiva na sql upite I dalje kako je navedeno može se napraviti sql injekcija .

Slika 8 Naredba or ‘1’=1

sNa login stranici u polja za unos, unosimo naredbu “’OR ‘1’=1˘” što vraća pozitivan rezultat poslužitelju I time kao što prikazano na slici to ubacuje u sustav.



Slika 9 Ulaz u bazu podataka

Zaštita od SQL injekcija se sastoji od par metoda koje ili djeluju na klijentskoj ili na serverskoj strani. Jedna od klijentskih načina zaštite od SQL injekcija je korištenje RegExa tako da se u unos u web stranici unese da traži i izbjegava određene setove simbola kao štosu = ili ‘. Primjer ovakve implementacije bi bio RegEx kod “/[\t\r\n]|(--[^\r\n]*)|(\/*[\w\W]*?(?=*)*\/)/gi” . Taj regularni izraz se brine da nema nikakvih komentara u input elementu na stranici. Jedan od načina je automatsko enkodiranje a to znači da Parametrizirani upiti koriste mehanizme enkodiranja koji automatski pretvaraju korisnički unosu siguran format prilikom izvršavanja upita. Na taj način se osigurava da korisnički unos bude tretiran samo kao vrijednost, a ne kao dio SQL koda. Način na koji se enkodiraju sql upiti je da se Parametri prenose odvojeno od SQL izjave, a baza podataka koristi te parametre za generiranje sigurnog SQL upita. To sprječava mogućnost umetanja zlonamjernog SQL koda putem korisničkog unosa. Takav način unosa dovodi do toga da se korisnički unos tretira samokao vrijednost, a ne kao dio SQL koda. To znači da čak i ako korisnik pokuša unijeti posebne znakove koji bi mogli utjecati na izvršavanje SQL izjave, oni će biti tretirani samo kao podaci,a ne kao dio izvršnog koda. Svi ti mehanizmi u Web aplikacijama se implementiraju koristeći nekakve ORM aplikacije i dodatne klase u kodu kao što su SQLAlchemy ili SQLLite. Ovakvi

ORM alati imaju već sve mogućnosti interakcije sa bazom podatak da korisnik ne mora sam ništa radit od početka i automatski primjenjuju sigurnosne korake.

8.2.1. Zaštita od SQL napada

Na slici je prikazan route „/SQLupit“ koji ima parametar id u URL-u. ID smo spremili u varijablu id i prikazana su tri načina zaštite od SQL Injekcije.

```
app.get('/SQLupit:id', (req, res) => {
    const id = req.params.id
    //Izrada parametriziranog upita pomocu ?
    const upit1 =  'SELECT * FROM baza WHERE id = ?'
    const podaci = db.all(upit1, id, (err, result) => {
        console.log(result)
    })      You, 4 days ago • Kolacice sam okrenu …
    //Provjera unesenih znakova pomoću regexa
    const regx = /^[A-Za-z]+$/;
    if(id.match(regx)) {
        console.log('Unos je validan')
    }
    //Korištenje AllowListinga
    const dopusteniUnosi = ["Lozinka1", "lozinka2", "lozinka 3"]
    if(dopusteniUnosi.includes(id)){
        res.send("Odgovor")
    } else {
        res.status(404)
    }
})
```

Slika 10 Prikaz tipova zaštite u kodu

Prvi način obrane je korištenje parametriziranog upita sa znakom ?. Sintaksa slanja takvog upita se razlikuje od biblioteke do biblioteke pa u ovome slučaju koristimo SQLite. SQLite radi parametrizirane upite tako da u glavnom objektu baze se koristi metoda all koja prima tri argumenta. Prvi argument je string s parametriziranim izrazima, drugi argument je polje ili varijabla koja sadrži podatke koje parametrizirani upit treba proslijediti da bi se došlo do rezultata i treći argument je callback funkcija koja diktira što se dešava s rezultatom i kako se taj rezultat procesuira. Drugi način obrane je provjera unesenih znakova pomoću regexa. Regex je tester stringova koji na temelju regularnih izraza i svog specifičnog jezika može tražiti i očekivati određene znakove u stringu. U ovome primjeru regex kod traži da string koji se šalje kao parametar sadrži samo slova što dovodi do toga da napadač ne može slati znakove kao

što su zvjezdica, točka-zarez jer regex u stringu dopušta samo slova. Treći način napada bibilo AllowListing što funkcionira tako da programer zna što se treba upisati u polje i već postoji određen set riječi koje se smiju upisati pa se unos uspoređuje sa WhiteListanim riječima.[36]

8.3. Cross-Site Scripting

Cross-Site Scripting (XSS) je vrsta sigurnosnog propusta koji omogućuje napadaču da ubaci zlonamjernu skriptu na web stranicu koja se kasnije izvršava u pregledniku korisnika. Jedna od najvećih prijetnji koju programeri web aplikacija moraju razumijeti je kako izbjeći XSS-napade. Iako je XSS relativno mali dio sigurnosti web aplikacija, za korisnika predstavlja najopasniji dio.[37] Napadač koristi ranjivost u web aplikaciji kako bi umetnuo zlonamjerni kod koji će se izvršiti na strani klijenta.

Kada je u pitanju Ajax (Asynchronous JavaScript and XML), koji je tehnika za izradu dinamičkih i interaktivnih web aplikacija, XSS napadi također mogu biti prisutni. Neki od načina kako se XSS može pojaviti u kontekstu Ajaxa:[*38*]

- Nepravilno rukovanje unosom: Web aplikacija ne uspijeva ispravno potvrditi valjanost ili očistiti korisnički unos prije nego što ga uključi u Ajax zahtjev.

- Generiranje dinamičkog sadržaja: Web aplikacija dinamički generira HTML ili JavaScript sadržaj na temelju korisničkog unosa ili drugih podataka dohvaćenih s poslužitelja.

- Ubacivanje zlonamjernih skripti: Napadač može iskoristiti XSS ranjivost ubacivanjem zlonamjernih skripti u generirani sadržaj. Ove se skripte mogu izvršiti unutar žrtvinog preglednika kada se obradi Ajax odgovor.

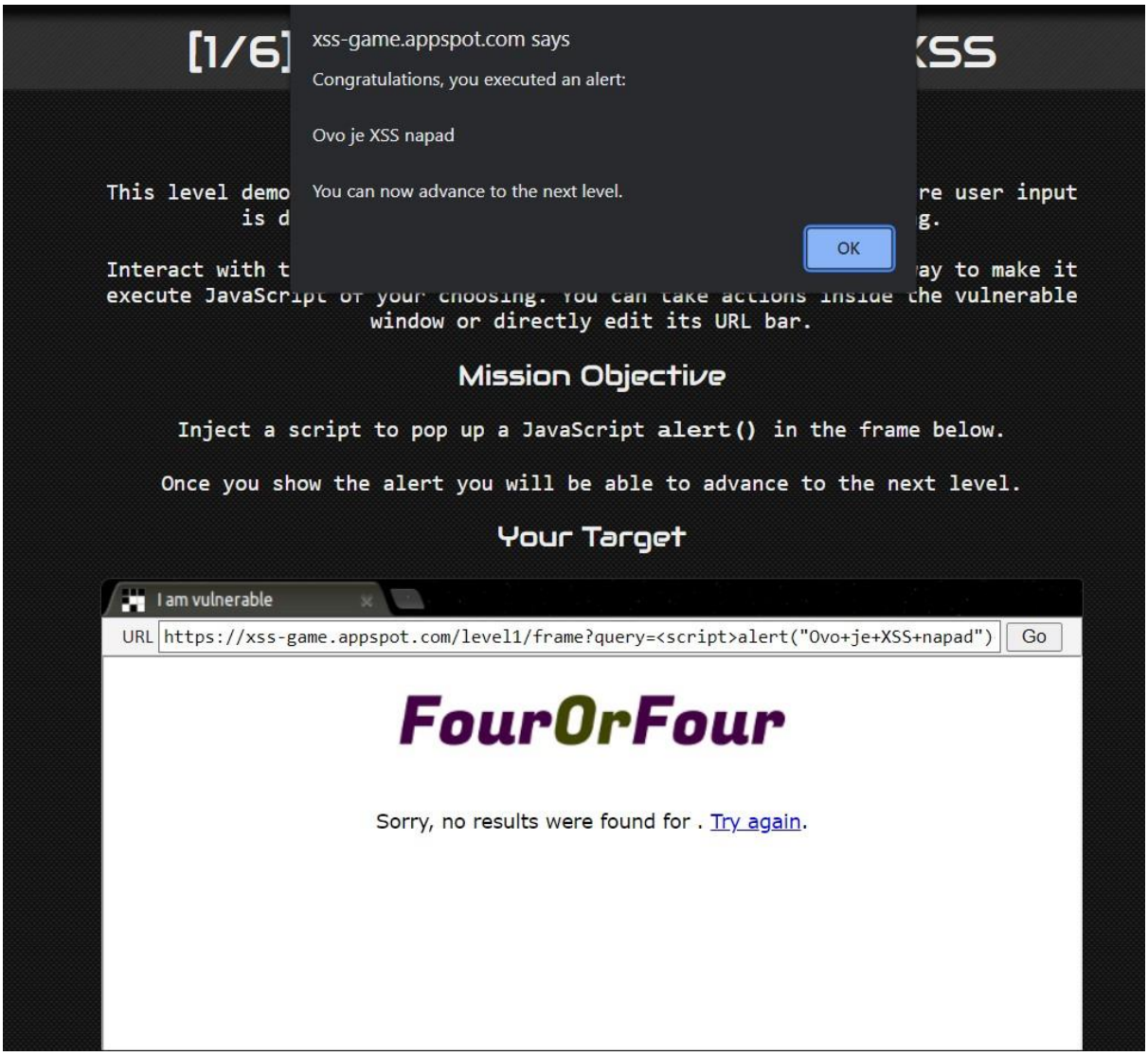
- Utjecaj na žrtvu: Zlonamjerne skripte mogu izvoditi različite radnje unutar žrtvinog preglednika, poput izmjene prikazanog sadržaja, krađe korisničkih podataka ili pokretanja daljnjih napada.

Za sprječavanje XSS napada u Ajax aplikacijama treba poduzeti sljedeće mjere.[39] Implementirati stroge rutine validacije i dezinfekcije unosa i na strani klijenta i na strani

poslužitelja. Osigurati da je unos korisnika ispravno provjeren i da su svi potencijalno zlonamjerni znakovi kodirani ili filtrirani. Prilikom generiranja dinamičkog sadržaja za Ajax odgovore, ispravno kodiranje svih podataka koje generiraju korisnici ili podaci na strani poslužitelja kako bi se spriječilo izvršavanje skripti. Upotrebavlja se tehnika izlaznog kodiranjas obzirom na kontekst kako bi se osiguralo da se podaci ispravno prikazuju unutar HTML ili JavaScript konteksta. Implementiranje jake politike sigurnosti sadržaja koja specificira dopuštene izvore sadržaja, uključujući skripte, tablice stilova i druge resurse. To pomaže u sprječavanju izvršavanja neovlaštenih skripti. Pridržavanje pravila istog podrijetla, koja ograničavaju interakcije između različitih domena. Upotrijebljavaju se odgovarajuće CORS (Cross-Origin Resource Sharing) konfiguracije za kontrolu zahtjeva s različitim izvorima i spriječava se neovlašteni pristup osjetljivim podacima. Izvršavanje temeljito sigurnosno testiranja, uključujući skeniranje ranjivosti i testiranje penetracije, kako identificirati i riješiti XSS ranjivosti unutar Ajax aplikacija. Implementacijom ovih sigurnosnih mjera, programeri mogu značajno smanjiti rizik od XSS napada u Ajax aplikacijama i osigurati integritet i sigurnost svojih web aplikacija.

Slika 11 Simulacijska web stranica XSS-game

U ovom primjeru XSS korištena je simulacijska web stranica xss-game. Tu stranicu je napravio google 2014. godine da bi pokazao kako je lagano iskoristiti XSS slabosti.



Slika 12 Prikaz XSS napada

U pozor unos je unešen niz znakova <script>alert("Ovo je XSS napad")</script>. Taj niz znakova je HTML oznaka za JavaScript koja u sebi sadrži naredbu alert, što nakon potvrde obrasca dovodi do pokretanja toga segmenta koda u nativnom pregledniku vidi kao naredba Alert.

8.3.1. Zaštita od XSS napada

Zaštita od XSS napada se sastoji od par načina obrane kao što su saniranje inputa, restriktiranje inputa korisnika i korištenje http vezanih kolačića.

```
import validator from "validator";
let userInput = `Jane <script onload="alert('XSS hack');"></script>`;
let sanitizedInput = validator.escape(userInput);
```

Slika 13 Primjer korištenja validatora u kodu

Ovo je primjer datoteke validator koja potvrđuje da korisnički unos nema nikakvih zlonamjernih skripti tako da parsira simbole u oblik koji nije opasan za korisnika i server. Kada se validira dobije se ispis ovakav: Jane onload="alert('XSS hack');">

Drugi način je validacija inputa na strani korisnika što znači korištenje html tagova i atributa unutar input tagova koji gledaju vrstu unosa koja mora biti. Ako se u određenom polju traži email može se namjestiti što automatski pazi i parsira da taj input ima sve atribute email adrese.

Treći način obrane od xss napada bi bio korištenje HTTPOnly kolačića što znači da tijekom izrade se programiraju kolačići kojima se ne može pristupiti pomoću nijedne skripte i tim kolačićima može pristupiti samo server.

```
app.use(express.session({
    secret: "secret",
    cookie: {
        httpOnly: true,
        secure: true
    }
}))
```

Slika 14 Prikaz konfiguriranja ekspresa

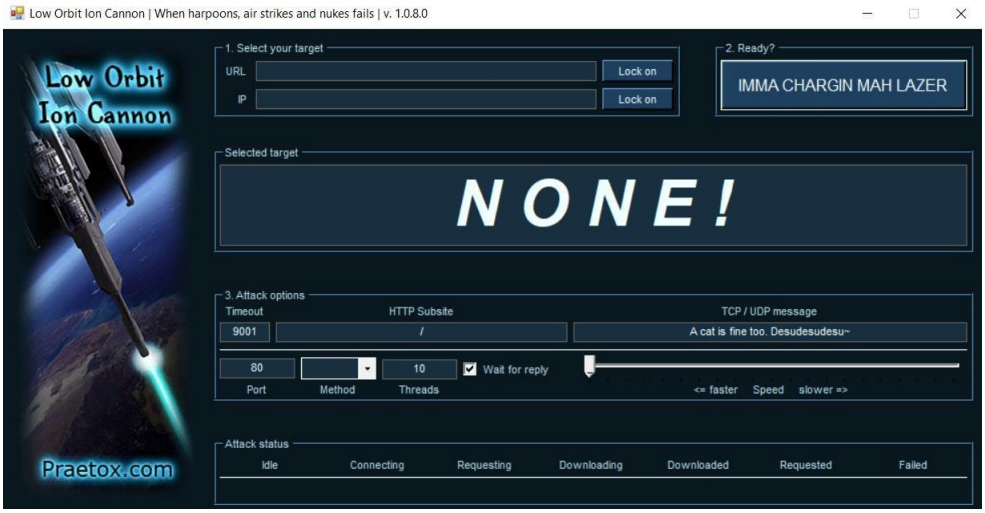
Primjer ovoga koda koristi app.use metodu što predstavlja metodu za konfiguriranje servera i unutra se upisuje httpOnly:true i secure:true da server i korisnička strana zna da se radi o kolačićima kojima može pristupiti samo server a ne neki vanjski korisnik koji bi mogao ukrasti podatke podvaljivanjem lažnog kolačića. [40]

8.4. DDoS napad

DDoS (Distributed Denial of Service) napad je vrsta napada na računalne mreže kojom se pokušava onemogućiti pristup legitimnim korisnicima ciljane web stranice tako što se generira veliki broj zahtjeva za pristupom ciljanoj web stranici. To je koordinarni napad koji koristi puno kompromitiranih poslužitelja.[41] Pretpostavimo da postoji online trgovina koja prodaje robu i usluge putem web stranice. Napadač može koristiti DDoS napad kako bi onemogućio pristup web stranici te tako spriječio prodaju robe i usluga. Napadač koristi veliki broj računala (kojasu zaražena malverom i postaju dio tzv. bot mreže) kako bi slali zahtjeve prema web stranici trgovine, što može uzrokovati preopterećenje servera i smanjenje dostupnosti web stranice. Ovakav napad može dovesti do toga da korisnici ne mogu pristupiti web stranici i obaviti kupnju, što može značajno utjecati na poslovanje trgovine. Ovaj napad je vrlo učinkovit jer zahtijeva minimalnu stručnost i znanje te se može provesti relativno jednostavno, a posljedice mogu biti vrlo štetne i dugotrajne. Zaštita od DDoS napada uključuje korištenje različitih mjera, uključujući primjenu zaštitnih programa, upotrebu usluge koja može filtrirati i blokirati sumnjive mrežne promet i uklanjanje zaraženih računala iz mreže.

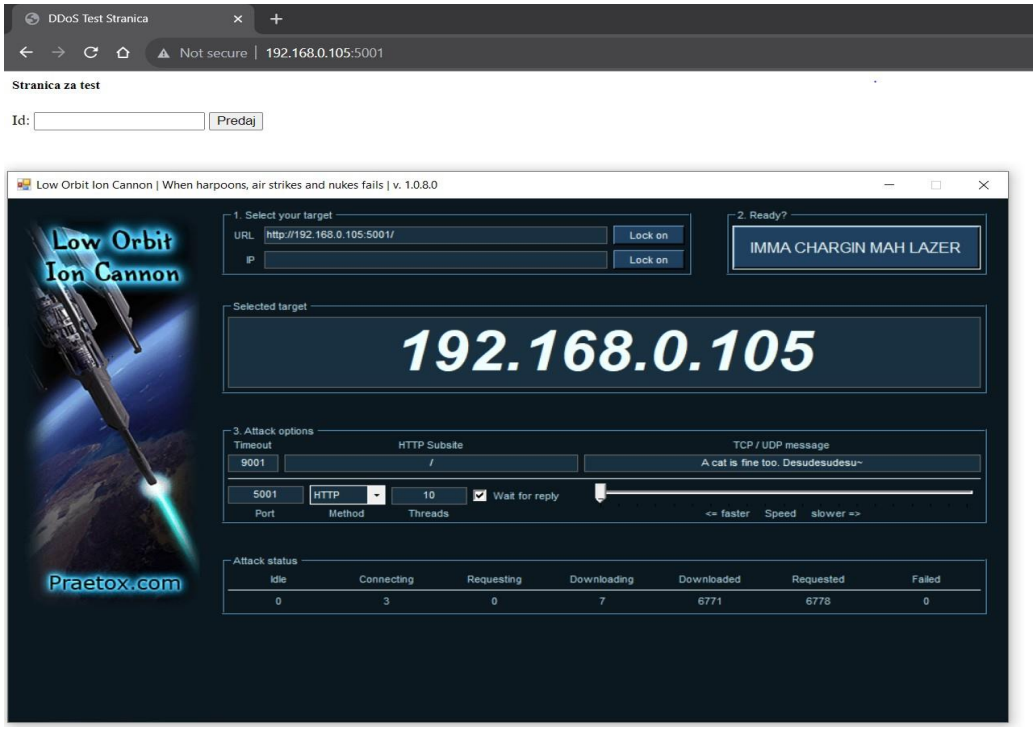
8.4.1. Demonstracija DDoS napada

DDos napad će biti demonstriran koristeći LOIC i virtualnu mašinu sa NodeJs serverom na lokalnoj mreži. Napad će se izvesti na lokalnoj mreži jer su DDoS napadi kažnjivi, čime napadač može biti progonjen na sudu. Za razumijevanje napada je potrebno znati da server koji će se napadati će biti upaljen na virtualnoj mašini na jednome uređaju na mreži dok će naistoj mreži biti spojen uređaj napadača. Demonstracijska stranica se izvodi na lokalnoj mašini da DDos napad ne bi usporio rad računala nego samo usporio virtulnu mašinu. Program za demonstraciju će biti LOIC iliti Low Orbital Ion Cannon. To je program napisan primarno u C#-u i on daje grafičko sučelje za izvođenje DDoS napada. Low Orbit Ion Cannon se koristi da se u url upiše url servera nad kojim se simulira DDoS napad i klikom na gumb lock on ta se adresa sprema u program. Nakon što se unese url u trećem koraku se unese port servera i vrsta konekcijskog protokola HTTP,UDP i TCP. Kada je upisan port i vrsta napada pritisne se gumb za izvršavanje napada.



Slika 15 Low Orbit Ion Cannon

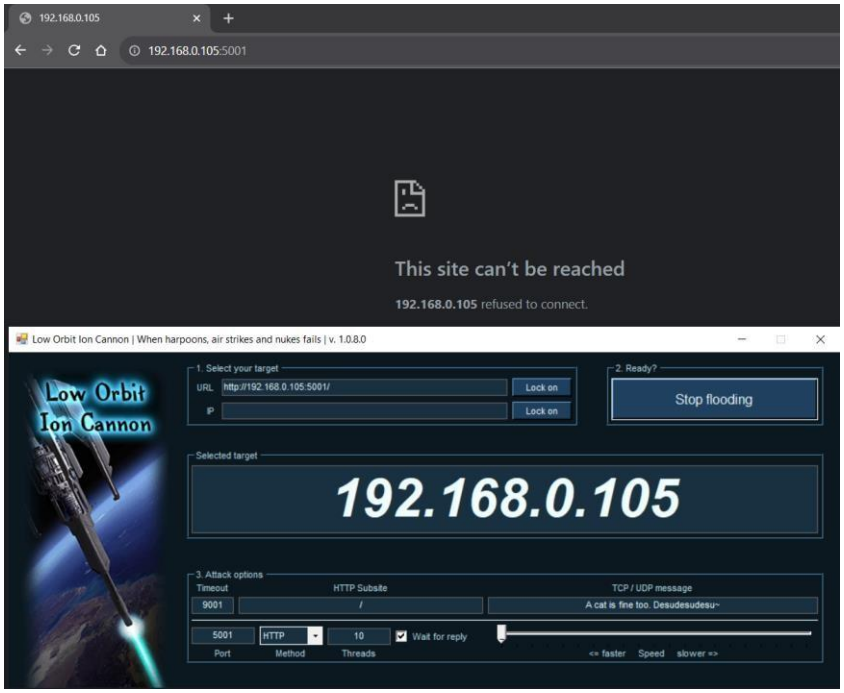
U ovom koraku izvođenja su upisani podaci mreže i servera na koji se spaja što uključuje URL, ip servera i port. U ovom primjeru IP je:192.168.0.105 a to je adresa drugog uređaja koji poslužuje server na lokalnoj mreži. Na slici uz program prikazana je web aplikacija s naslovom i obrascem koja je poslužena sa lokalne mreže.



Slika 16 Napad na testnu web stranicu

U zadnjem koraku pri izvršenom napadu se može primijetiti da kada se korisnik pokuša spojiti na svoj poslužitelja u lokalnoj mreži, dođe do pogreške koja je nastala zbog

preopterećenje servera koju su izazvana od strane LOIC-a, tako da je program poplavio poslužitelja velikim brojem zahtjeva.



Slika 17 Adresa web stranice i output napada

8.4.2. Obrana od DDoS napada

Obrana od DDoS napada se temelji na hardwareskoj i softwareskoj strani. Hardverska strana obrane bi se sastojala od povećanja prostora na serveru, povećanja propusnosti(bandwidtha) da server bude otporniji na veći promet i korištenje DDoS Cloud Based zaštitu što znači da prije nego što zahtjev dođe na server prođe kroz čistilište gdje se provjerava legitimnost. Kada smo kod softverske zaštite postoji par načina i primarno ćemo ih pokazati u NodeJS-u. Prvi primjer softverske obrane je limitiranje zahtjeva od iste IP adrese u određenom vremenskom intervalu. Na slici se vidi kako se u varijablu sprema objekt rate limiti u objekt limiter se poziva konstruktor objekta rate limit u koji se spremaju podaci o tome kako bi se trebali procesuirati zahtjevi. Kao argumenti konstruktora navedeni su windowMs što se odnosi koji bi bio intervala gledanja broja zahtjeva, drugi argument je max koji diktira koliko se zahtjeva može poslati u windowMs intervalu i treći je poruka koja se šalje kada se prekorači broj dopuštenih zahtjeva. Na kraju svega koristi se app.use(limiter) što sprema postavke iz objekta limiter.

```
const rateLimit = require("express-rate-limit");

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: "Too many requests from this IP, please try again in a few minutes",
});

// apply to all requests
app.use(limiter);
```

Slika 18 Primjer limitiranja zahtjeva u NodeJS-u

Druga softverska metoda zaštite od DDoS napada bi bila IP blokiranje. U ovome slučaju se koristi biblioteka IpFilter koja kao argumente uzima polje IP adresa od kojih se blokiraju zahtjevi pa kada se prepozna IP adresa koja radi DDOS napad može se direktno u kodu zablokirati svaki zahtjev s te adrese.

```
const ipfilter = require("express-ipfilter").IpFilter;

const ips = ["127.0.0.1", "::ffff:127.0.0.1"]; // add offending IP addresses here

const ipfilterMiddleware = ipfilter(ips, { mode: "deny" });

// apply to all requests
app.use(ipfilterMiddleware);
```

Slika 19 Kod filtriranja IP adrese

Posljednja metoda zaštite od DDoS napada je validacija zahtjeva. U ovome slučaju se koristi express-validator što je ustvari middleware(U NodeJS-u funkcija koja pokreče i obrađuje podatke zahtjeva prije glavne funkcije). U ovome slučaju se gleda da post zahtjevi na putanju „/login“ budu email i da lozinka bude minimalno 6 znakova duga. To vodi do toga da svi bezvezni zahtjevi koje napadač šalje neodgovaranju podacima traži middleware funkcija čime napadać nazad dobije status zahtjeva 400.[42]

```
const { body, validationResult } = require("express-validator");

app.post(
  "/login",
  // validate request body
  body("username").isEmail(),
  body("password").isLength({ min: 6 }),
  (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    // handle valid request here
  }
);
```

Slika 20 Prikaz validatora u kodu

9. Zaključak

Testiranje web aplikacije ključni je postupak koji osigurava njezin rad, upotrebljivost, sigurnost i ukupnu učinkovitost. Organizacije i programeri mogu dobiti značajan uvid u prednosti i nedostatke svojih aplikacija kroz rigoroznu procjenu, što im omogućuje donošenje obrazovanih odluka za promjenu.

Organizacije mogu procijeniti koliko dobro korisnici mogu raditi s aplikacijom provodeći rigorozno testiranje upotrebljivosti, ističući područja koja zahtijevaju jednostavnost ili poboljšanje. Testiranje performansi identificira potencijalna uska grla i osigurava da program može upravljati predviđenim korisničkim opterećenjem bez žrtvovanja brzine ili odziva.
U današnjem digitalnom okruženju, ispitivanje sigurnosti je kritično. Identificiranje ranjivosti putem testiranja prodora i sigurnosnih procjena ključno je za sprječavanje mogućih probojakoji bi mogli ugroziti osjetljive korisničke podatke ili integritet aplikacije. Testiranje sukladnostijamči da je aplikacija u skladu sa svim primjenjivim normama i standardima, stvarajući povjerenje i zakonsku usklađenost. Kontinuirana evaluacija neophodna je u današnjem svijetu brzih tehničkih poboljšanja. Redovite evaluacije i nadogradnje na temelju povratnih informacija korisnika, novih prijetnji i promjenjivih zahtjeva ključni su za relevantnost i pouzdanost aplikacije.

Evaluacija web aplikacija nije jednokratan pothvat, već trajna predanost izvrsnosti. Omogućuje organizacijama da isporuče web aplikacije koje nude iznimna korisnička iskustva, robusnu sigurnost i optimalne performanse. Kako se tehnologija nastavlja razvijati, tako se moraju razvijati i metode evaluacije i strategije koje se koriste, omogućujući web aplikacijama da napreduju u dinamičnom i konkurentnom digitalnom krajoliku.

Popis literature

[1] Website testing: A detailed guide. (2022, srpanj 21). *LambdaTest*. https://www.lambdatest.com/blog/website-testing/

[2] Weidman, G. (2014). *Penetration Testing: A Hands-On Introduction to Hacking*. https://openlibrary.org/books/OL26838292M/Penetration_Testing_A_Hands-On_Introduction_to_Hacking

[3] Petukhov, A., Kozlov, D. (2008.), Detecting Security Vulnerabilities in Web Applications,Using Dynamic Analysis with Penetration Testing

[4] Implementing authentication and authorization in web applications: Best practices. (2023, travanj 17). *CronJ*. https://www.cronj.com/blog/implementing-authentication-and-authorization-in-web-applications-best-practices/

[5] Dizdar, A. (2022, svibanj 29). Security testing: Types, tools, and best practices. *Bright Security*. https://brightsec.com/blog/security-testing\

[6] *How do you document and report the results and recommendations of your security testing?* (bez dat.). Preuzeto 06. rujan 2023., od https://www.linkedin.com/advice/0/how-do-you-document-report-results-recommendations-1f

[7] *What is web vulnerability scanning? A guide from portswigger*. (bez dat.). Preuzeto 24. kolovoz 2023., od https://portswigger.net/burp/vulnerability-scanner/guide-to-vulnerability-scanning

[8] *Wstg—Latest | owasp foundation*. (bez dat.). Preuzeto 24. kolovoz 2023., od https://owasp.org/www-project-web-security-testing-guide/latest/3-The_OWASP_Testing_Framework/1-Penetration_Testing_Methodologies

[9] Yasar, K. (2023). network vulnerability scanning. *Security*. https://www.techtarget.com/searchsecurity/definition/vulnerability-scanning

[10] Harper, A. (2023). What is the point of authenticated web application penetration testing? *Evalian®*. https://evalian.co.uk/what-is-the-point-of-authenticated-web-application-penetration-testing/#:~:text=An%20authenticated%20test%20assumes%20the,issued%20to%20a%20valid%20user

[11] Implementing authentication and authorization in web applications: Best practices. (2023, travanj 17). *CronJ*. https://www.cronj.com/blog/implementing-authentication-and-authorization-in-web-applications-best-practices/

[12] *Top 10 open source security testing tools for web applications(Updated)*. (bez dat.). Hackr.Io. Preuzeto 24. kolovoz 2023., od https://hackr.io/blog/top-10-open-source-security-testing-tools-for-web-applications

[13] Berge, A. (2023, travanj 12). *5 common threats to web applications and how to avoid them*. Geekflare. https://geekflare.com/common-web-application-threats/
[14]10 web application security threats and how to mitigate them. (bez dat.). StackHawk. Preuzeto 06. rujan 2023., od https://www.stackhawk.com/blog/10-web-application-security-threats-and-how-to-mitigate-them/
[15] *Brute force attack* | *owasp foundation*. (bez dat.). Preuzeto 24. kolovoz 2023., od https://owasp.org/www-community/attacks/Brute_force_attack/
[16] What is authentication attack? (2022, srpanj 10). GeeksforGeeks. https://www.geeksforgeeks.org/what-is-authentication-attack/
[17] National Institute of Standards and Technology [NIST] (bez dat.). Replay attack—Glossary | csrc. Preuzeto 06. rujan 2023., od https://csrc.nist.gov/glossary/term/replay_attack
[18] Venafi. (bez dat.). *What are timing attacks and how do they threaten encryption?* | *venafi*. Preuzeto 06. rujan 2023., od https://venafi.com/blog/what-are-timing-attacks-and-how-do-they-threaten-encryption/
[19] Common Protocol Vulnerabilites (2023.) Preuzeto 7.9.2023. https://cqr.company/web-vulnerabilities/common-protocol-vulnerabilities/
[20] *Session management—Owasp cheat sheet series*. (bez dat.). Preuzeto 06. rujan 2023., od https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html
[21] *Forced browsing* | *owasp foundation*. (bez dat.). Preuzeto 8. rujan 2023., od https://owasp.org/www-community/attacks/Forced_browsing
[22] *What is forced browsing*. (2020, listopad 12). Acunetix. https://www.invicti.com/learn/forced-browsing/
[23] *What are parameter tampering cyber attacks?* (bez dat.). Security. Preuzeto 02. rujan 2023., od https://www.techtarget.com/searchsecurity/definition/parameter-tampering
[24] Limited, S. L. (2022, prosinac 22). Http header injection. Medium. https://infosecwriteups.com/http-header-injection-4ba857fb9a16
[25] [*Preventing attacks using http headers*. (bez dat.). Twilio Blog. Preuzeto 08. rujan 2023., od https://www.twilio.com/blog/preventing-attacks-using-http-headers
[26] *Sql injection*. (bez dat.). Preuzeto 02. rujan 2023., od https://www.w3schools.com/sql/sql_injection.asp
[27] *Blind sql injection* | *owasp foundation*. (bez dat.). Preuzeto 10. rujan 2023., od https://owasp.org/www-community/attacks/Blind_SQL_Injection
[28] Sara Jelen, Cybersecurity Red Team Versus Blue Team — Main Differences Explained, 14.10.2021, https://securitytrails.com/blog/cybersecurity-red-blue-team

[29] Miessler, D. (bez dat.). *The difference between red, blue, and purple teams*. Unsupervised Learning. Preuzeto 24. kolovoz 2023., od https://danielmiessler.com/p/red-blue-purple-teams
[30] *About the owasp foundation | owasp foundation*. (bez dat.). Preuzeto 29. kolovoz 2023., od https://owasp.org/about/
[31] Penetration testing services. (bez dat.). OffSec. Preuzeto 03. rujan 2023., od https://www.offsec.com/penetration-testing/
[32] What is a zero day vulnerability exploit? (bez dat.). Security. Preuzeto 06. rujan 2023., od https://www.techtarget.com/searchsecurity/definition/zero-day-vulnerability
[33] *What is a zero-day exploit? | ibm*. (bez dat.). Preuzeto 02. rujan 2023., od https://www.ibm.com/topics/zero-day
[34] Contributor. (2020, rujan 19). The 5 most famous DDoS attacks in history—TechTalks. https://bdtechtalks.com/2020/09/19/top-5-ddos-attacks-in-history/
[35] Clarke, J. (2012). *SQL injection attacks and defense*. Elsevier.
[36] *How to prevent SQL injection attacks in Node.js | PlanetScale*. (2022, ožujak 3). https://planetscale.com/blog/how-to-prevent-sql-injection-attacks-in-node-js
[37] Fogie, S., Grossman, J., Hansen, R., Rager, A., & Petkov, P. D. (2011). *XSS attacks: Cross Site Scripting Exploits and Defense*. Elsevier.
[38] *Ajax tutorial*. (bez dat.). Preuzeto 08. rujan 2023., od https://www.tutorialspoint.com/ajax/index.htm
[39] *Cross site scripting prevention—Owasp cheat sheet series*. (bez dat.). Preuzeto 08. rujan 2023., od https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
[40] Gathoni, M. (2022, lipanj 18). *How to prevent cross-site scripting in node. Js*. MUO. https://www.makeuseof.com/prevent-cross-site-scripting-in-nodejs/
[41] Bhattacharyya, D. K., & Kalita, J. K. (2016). *DDoS attacks: Evolution, Detection, Prevention, Reaction, and Tolerance*. CRC Press.
[42] Mitigation DDoS attack from nodejs server. (bez dat.). Preuzeto 08. rujan 2023., od https://www.linkedin.com/pulse/mitigation-ddos-attack-from-nodejs-server-vartul-goyal

Popis slika