Kibernetička sigurnost i ranjivosti: SSRF (Server-side Request Forgery)

Valentić, Lucija

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Lucija Valentić

KIBERNETIČKA SIGURNOST I RANJIVOSTI: SSRF (SERVER-SIDE REQUEST FORGERY)

Diplomski rad

Voditelj rada:
dr. sc. Goran Igaly

Zagreb, rujan, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Velika većina današnjih aplikacija su mrežne aplikacije koje se izvode na jednom ili više mrežnih poslužitelja. Korištenjem neke mrežne aplikacije te pregledavanjem njezina programskog kôda postoji mogućnost pronalaska raznih ranjivosti te aplikacije iako aplikacija možda na prvi pogled izgleda sigurna. Ovisno o ranjivosti, one mogu biti bezazlene ili imati malen utjecaj, ali mogu biti i katastrofalne što se tiče sigurnosti i privatnosti, poput recimo izvršavanja koda iz daljine (engl. *Remote Code Execution*, RCE). Pažljivim programiranjem i korištenjem znanja o kibernetičkoj sigurnosti aplikacije bi se mogle napraviti još sigurnijima. Na taj način, zlonamjernim korisnicima bi bilo teže ili nemoguće doći do osjetljivih podataka, ometanja rada aplikacije, i općenito iskorištavanja funkcija aplikacije koje nisu namijenjene biti iskorištene na taj način.

Jedna od ranjivosti koja se sve češće pojavljuje jest SSRF ranjivost. OWASP je 2021. godine uvrstio tu vrstu ranjivosti na svoj Top 10 popis svih najkritičnijih ranjivosti koje su se po mišljenjima stručnjaka pojavile u *web* aplikacijama. Iako se SSRF nije pojavio vrlo često u aplikacijama, radi se o vrlo bitnoj ranjivosti [13].

SSRF (engl. *Server Side Request Forgery*) je vrsta kibernetičke ranjivosti koja sama po sebi nije vrlo opasna, ali ponekad napadač može povezati još opasnijih ranjivosti za napad na aplikaciju iskorištavajući je. Koristeći SSRF ranjivost, napadač ima mogućnost natjerati ranjivu aplikaciju da kontaktira proizvoljnu lokaciju. U normalnim okolnostima, napadaču nije dostupna ta lokacija, ali ako je lokacija dostupna poslužitelju na kojoj se poslužuje ranjiva aplikacija, tada će on u određenim okolnostima moći poslati zahtjev prema ovoj lokaciji. Problem zapravo nastaje kad se koristi korisnikov input bez ikakve provjere ili filtriranja kako bi se napravio zahtjev prema nekoj lokaciji. Ako korisnik može proizvoljno mijenjati URL, putanju prema kojoj se šalje zahtjev, tada postoji mogućnost da je aplikacija ranjiva na SSRF.

U ovom diplomskom radu obrađuje se upravo ova vrsta ranjivosti. Cijeli diplomski rad je tematski podijeljen na četiri poglavlja. Svako poglavlje se sastoji od teksta koji opisuje tu tematsku cjelinu, te od konkretnih primjera koji pokušavaju što jasnije objasniti napisan tekst.

Prvo poglavlje ”Uvod u SSRF” služi, kao što i samo ime govori, kao uvod u ranjivost. Prije samog objašnjavanja ranjivosti, dana su su dodatna objašnjenja oko HTTP zahtjeva radi lakšeg shvaćanja na koji način SSRF može funkcionirati. Nakon toga, slijede tri primjera koja opisuju različite scenarije u kojima se može pojaviti SSRF ranjivost i na koji je se način može iskoristiti. Svaki scenarij je popraćen primjerom koji je napravljen upravo za njega. Isto tako, svaki scenarij sadrži i dijagram radi njegovog lakšeg shvaćanja.

Drugo poglavlje ”SSRF napadi” pobliže opisuje kako točno otkriti da je aplikacija ranjiva na SSRF. Dani su neki konkretni primjeri u kojima se opisuju posljedice iskorištavanja SSRF ranjivosti. Za neke scenarije su napravljeni konkretni primjeri koji pokazuju kako bi se mogla iskoristiti SSRF ranjivost u njemu.

Treće poglavlje ”Zaštita od SSRF napada” pobliže opisuje na koji način se mogu aplikacije zaštititi od SSRF ranjivosti. Budući da neka od opisanih rješenja nisu savršena, nakon njih je opisan i način zaobilaženja tih rješenja.

Na kraju, četvrto i zadnje poglavlje ”SSRF scenariji: aplikacijski kod” je poglavlje u kojem se opisuje aplikacija napravljena za ovaj diplomski rad. Aplikacija se sastoji od svih primjera koji su bili prikazani u prethodnim poglavljima, ali su svi integrirani u jednu aplikaciju radi njihovog lakšeg korištenja. Sama aplikacija je zamišljena kao mjesto na kojem se mogu samostalno isprobati iskorištavanja SSRF ranjivosti u raznim scenarijima, ali i kao neka vrsta edukacije kojom se pokušava pokazati na koji način je SSRF ranjivost opasna.

Poglavlje 1

Uvod u SSRF

Današnje *web* aplikacije koje se poslužuju na poslužiteljima (engl. *servers*) su vrlo često modularne. To znači da postoje različiti dijelovi aplikacije koji su posebno zaduženi za obavljanje samo jednog dijela posla. Primjerice, posebni dijelovi aplikacije su zaduženi za *frontend*, *backend* i rad s bazom podataka. Ovisno o veličini i potrebama organizacije koja stoji iza aplikacije, infrastruktura aplikacije može biti različita. Razvijatelji aplikacije mogu odlučiti o lokaciji pojedinih dijelova: svi dijelovi se mogu nalaziti na jednom poslužitelju, no u današnje vrijeme, vrlo često se dijelovi nalaze na više poslužitelja, gdje je svaki zadužen za jedan dio posla. Na svakom poslužitelju se nalaze servisi koji nude svoje usluge i pomoću njih se obavljaju poslovi. Jedan poslužitelj se brine o *frontendu*, jedan poslužitelj se bavi autentifikacijom [1] korisnika ako aplikacija ima neki dio gdje je potrebna autentifikacija, dok se neki drugi poslužitelj bavi dohvaćanjem informacija iz baze podataka.

Aplikacija u svakom trenutku mora moći brzo, sigurno i pouzdano doći do informacija iz baze podataka, ili mora biti u mogućnosti autentificirati i autorizirati [2] korisnika, te zato svi dijelovi moraju moći međusobno komunicirati. Glavni poslužitelj koji poslužuje aplikaciju, te ostali koji se bave samo jednim dijelom čine mrežu poslužitelja. Pristup toj internoj mreži poslužitelja i servisa je onemogućen vanjskim korisnicima pomoću vatrozida (engl. *firewall*). Upravo iz tog razloga, može se reći da postoji *veza povjerenja* (engl. *trust relationship*) među poslužiteljima.

Budući da se smatra da vanjski korisnici neće nikad moći pristupiti unutarnjim resursima, administratori infrastrukture relaksiraju autentifikaciju prema poslužitelja ili servisima, ili

[1] Postupak provjere identiteta, ne zamjenjivati s autorizacijom

[2] Postupak provjere razine pristupa korisnika, tj. postupak provjere ima li korisnik dopuštenje za tražene resurse

u nekim slučajevima, autentifikacije uopće nema - korisnik koji ima dozvolu za pristup unutarnjem poslužitelju, može se bez ikakvih vjerodajnica (engl. *credentials*) spojiti.
Kako bi aplikacija mogla dohvatiti resurse ili kontaktirati neki drugi servis ili poslužitelj, ona mora znati put do njega. Put se između ostalog može izraziti pomoću URL-a. Ovdje se može javiti ranjivost ako običan korisnik može utjecati na URL prema kojem se šalje HTTP zahtjev (engl. *HTTP request*).
Prije same definicije glavne kibernetičke ranjivosti o kojoj je riječ u ovom diplomskom radu, radi boljeg razumijevanja, u nastavku slijedi ukratko objašnjenje HTTP protokola - samo informacije koje su relevantne za ovaj diplomski rad.

1.1 HTTP/1.1 (engl. *Hypertext Transfer Protocol*)

HTTP [9] je protokol na aplikacijskoj razini. Služi za razmjenu podataka preko interneta. Svi podaci razmjenjuju se preko tzv. ”poruka”. Postoje dvije vrste HTTP poruka: zahtjevi (engl. *requests*) i odgovori (engl. *response*). Kada dva entiteta na internetu žele komunicirati, oni ostvare konekciju gdje jedan postane klijent, odnosno onaj koji šalje zahtjev i prima odgovor od poslužitelja, dok drugi postane poslužitelj koji prima zahtjev, te vraća odgovor.

Slika 1.1: Razmjena HTTP poruka

HTTP zahtjev

Svaki HTTP zahtjev započinje ”retkom zahtjeva” (engl. *request-line*), nakon koje slijedi nula ili više datoteka zaglavlja (engl. *header files*) koje se jednom riječju zovu zaglavlja. Zaglavlja završavaju praznim retkom koji označava njihov kraj, nakon čega može slijediti i tijelo poruke.
Redak zahtjeva uključuje metodu koja će biti korištena na resursu, zatim URI (engl. *Uniform Resourse Identifier*) resursa, i završava s verzijom protokola.

Slika 1.2 prikazuje jedan HTTP zahtjev. U ovom zahtjevu metoda je ”GET”, URI resursa je ”/admin.html?url=http://example.com”, a protokol zajedno s verzijom je ”HTTP/1.1”.

```
1  GET /admin.html?url=https%3A%2F%2Fexample.com HTTP/1.1
2  Host: localhost
3  sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
4  sec-ch-ua-mobile: ?0
5  sec-ch-ua-platform: "Linux"
6  Upgrade-Insecure-Requests: 1
7  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   Safari/537.36
8  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
   change;v=b3;q=0.9
9  Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Referer: http://localhost/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
```

Slika 1.2: Primjer HTTP zahtjeva

HTTP odgovor

Svaki HTTP odgovor započinje ”statusnim retkom” (engl. *status-line*), nakon kojeg slijedi nula ili više zaglavlja. Zaglavlja završavaju praznim retkom koji označava njihov kraj, nakon čega može slijediti i tijelo poruke.
Statusni redak uključuje verziju protokola koji se koristi nakon kojeg slijedi troznamenkasta vrijednost te niz znakova asociran s tom vrijednošću. Vrijednost pobliže označava klasu odgovora.

Slika 1.3 prikazuje jedan HTTP odgovor. U ovom odgovoru protokol zajedno s verzijom je `HTTP/1.1`, troznamenkasti kod je `200`, a tekst asociran s vrijednošću `200` je `OK`. Vrijednost `200` označava uspješan odgovor.

```
1  HTTP/1.1 200 OK
2  Date: Tue, 02 Aug 2022 14:59:16 GMT
3  Server: Apache/2.4.54 (Unix)
4  Last-Modified: Tue, 02 Aug 2022 11:16:24 GMT
5  ETag: "1e-5e5403f2af200"
6  Accept-Ranges: bytes
7  Content-Length: 30
8  Connection: close
9  Content-Type: text/html
10
11 <h1>
      Second Hello World!
   </h1>
12
```

Slika 1.3: Primjer HTTP odgovora

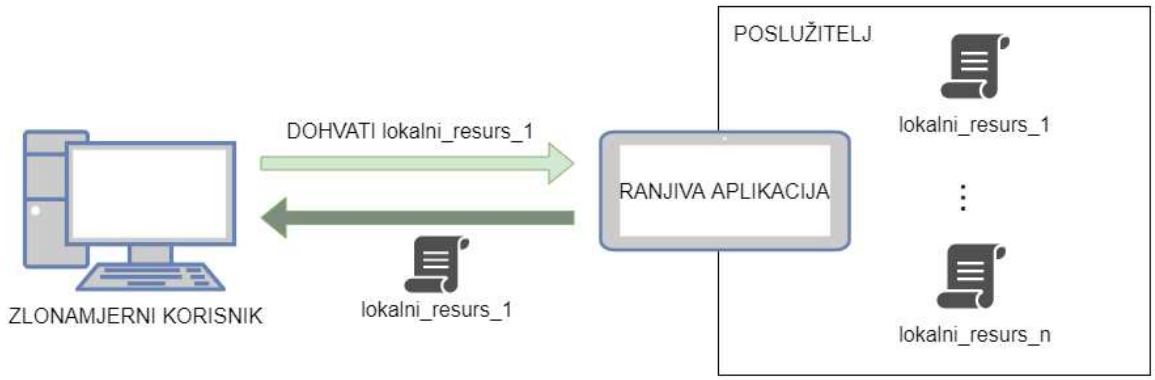
1.2 SSRF

SSRF [14] (engl. *Server-Side Request Forgery*) je kibernetička ranjivost gdje zlonamjerni korisnik uspije ”natjerati” ranjivu aplikaciju ili poslužitelj da napravi proizvoljan zahtjev (može biti HTTP zahtjev) prema nekoj proizvoljnoj lokaciji. Proizvoljna lokacija može biti javno dostupa, ali može biti i lokacija kojoj u normalnim okolnostima on (aka zlonamjerni korisnik) nema pristupa.
Sama SSRF ranjivost može biti bezopasna i imati mali utjecaj. No, zlonamjerni korisnik iskorištavanjem SSRF ranjivosti može imati mogućnosti iskoristiti i druge ranjivosti koje se potencijalno nalaze u aplikaciji ili poslužitelju. Tada on zapravo radi lanac ranjivosti (engl. *exploit chain*) gdje izvršava jednu ranjivost za drugom i na taj način dolazi do osjetljivih podataka, vjerodajnica, konfiguracijskih postavki servisa (iz kojih može iščitati postavke servisa, te potencijalno naći nove ranjivosti koje može iskoristiti), ili može doći i do udaljenog izvršavanja koda (*Remote Code Execution*, RCE).

Radi boljeg razumijevanja ove ranjivosti, u nastavku slijedi par scenarija u kojima se može javiti ranjivost, zajedno s dijagramima i primjerom koda ranjive aplikacije, te primjerom iskorištavanja ranjivosti.

Dohvaćanje lokalnih resursa

Neka postoji neka aplikacija koja se poslužuje na nekom poslužitelju. Aplikacija je vidljiva izvana i korisnik ima mogućnost njezina korištenja. Na poslužitelju se nalaze i lokalni resursi, koji uključuju sve datoteke i direktorije koji se na njemu nalaze. To se naravno odnosi i na osjetljive datoteke poput konfiguracijskih datoteka, ili primjerice datoteka koje sadrže vjerodajnice za neko admin sučelje.
U normalnim okolnostima, korisnik nema pristup lokalnim resursima poslužitelja. U slučaju da aplikacija ima SSRF ranjivost, postoji mogućnost da zlonamjerni korisnik uspije dohvatiti lokalnu datoteku i saznati osjetljive podatke koji bi mu mogli omogućiti daljnji napad aplikacije ili poslužitelja.
Aplikacija opisana u nastavku sadrži SSRF ranjivost koja zlonamjernom korisniku omogućuje dohvaćanje datoteke koja sadrži admin vjerodajnicu nakon što pošalje modificirani URL. Iskorištavanje SSRF ranjivosti na ovaj način prikazano je slikom 1.4.



Slika 1.4: Dohvaćanje lokalnih resursa poslužitelja iskorištavajući SSRF ranjivost

Za potrebe primjera, napravljen je poseban `Docker` [3] [3] kontejner koji u ovom slučaju predstavlja poslužitelj. U njemu se poslužuje jedna jednostavna aplikacija. Aplikacija se sastoji se od početne stranice (kao što je prikazano na slici 1.5) na kojoj korisnik može učitati sliku šaljući URL slike.
Nakon što korisnik pošalje URL slike, slika će se spremiti na poslužitelj te će se prikazati u nastavku početne stranice u umanjenom obliku.
Korisnik će zamijetiti kako se unutar HTTP zahtjeva šalje URL nad kojim on ima potpunu kontrolu. Zlonamjerni korisnik će probati poslati URL koji ne predstavlja URL slike, već putanju lokalnog resursa poslužitelja koji poslužuje aplikaciju.

[3] Docker je platforma koja pomaže pri pokretanju aplikacija

Slika 1.5: Početna stranica aplikacije

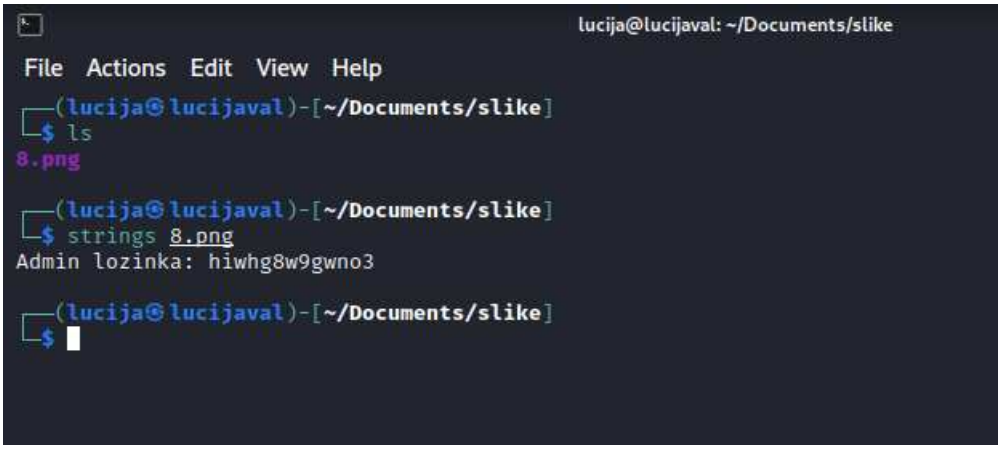
Osim aplikacije, na tom ”poslužitelju” je napravljen jedan poseban direktorij `secrets` koji sadrži datoteku `important_secret.txt` koja u ovom slučaju igra ulogu osjetljive datoteke s admin vjerodajnicom. Putanja osjetljive datoteke je `/secrets/important_secret.txt`. Ako zlonamjerni korisnik umjesto URL slike pošalje `file:///secrets/important_secret.txt`, prikazat će mu se slika s greškom. Nakon što je preuzme i prouči nizove znakova unutar te ”slike” pomoću naredbe `strings`[4], dobit će zapravo sadržaj datoteke `important_secret.txt`. Prethodno opisan korak se može vidjeti na slici 1.6

Što se zapravo događa?

Nakon slanja URL-a, njegov sadržaj se odmah preuzme u obliku niza znakova te se spremi na poslužitelj u posebno odabrani direktorij s `PNG` ekstenzijom. Nakon toga, ta novospremljena datoteka se prikaže u obliku slike. Ako slučajno poslani URL nije bio URL slike, sadržaj bi se i dalje preuzeo i spremio. Problem nastaje zbog toga što se poslani URL ne provjerava, već se sadržaj tog URL-a odmah preuzima. Sve se to može vidjeti iz sljedećeg koda[5]. Dakle, ako nema nikakve provjere poslanog URL-a, zlonamjerni korisnik može poslati URL koji označava putanju do neke datoteke na poslužitelju znajući da će ju aplikacija uspješno dohvatiti.

[4]Naredba omogućava čitanje svih nizova znakova duljine barem 4 znaka

[5]Napomena: kod je malo promijenjen radi bolje čitljivosti, ali logika je ostala ista

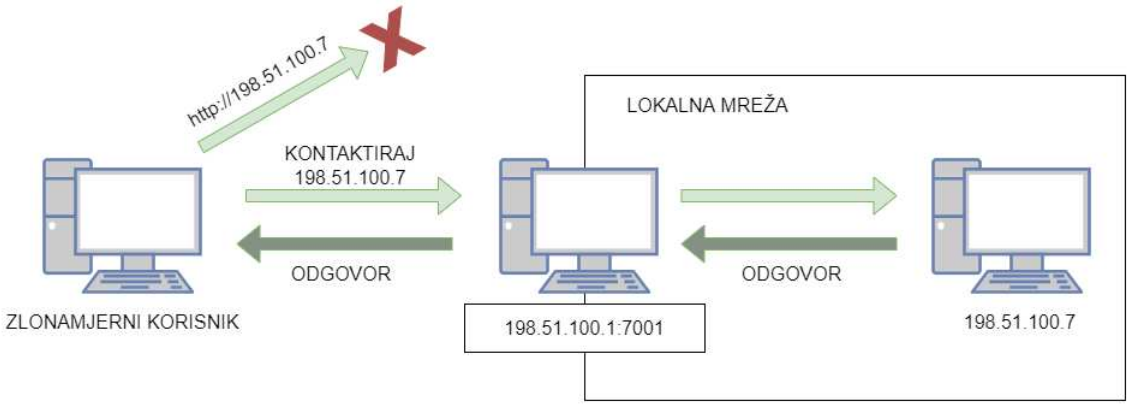


Slika 1.6: Otkrivanje sadržaja osjetljive datoteke

```
<form method="GET">
    <label >URL:</label>
    <input type="text"name="url" >
    <input type="submit" value="Submit">
</form>

<?php

if(isset($_GET['url'])){
    $url = $_GET['url'];
    $contents = file_get_contents($url);

    $new_file = "images/".rand(1, 100).".png";
    file_put_contents($new_file, $contents);

    $image = "<img src=\"".$new_file."\">";
    echo $image;
}
?>
```

Kontaktiranje ostalih poslužitelja u lokalnoj mreži

Neke od aplikacija se poslužuju na poslužiteljima koji su dio neke unutarnje mreže. Interna mreža se sastoji od više poslužitelja koji nisu dostupni izvana, tj. u normalnim okolnostima, korisnici izvan lokalne mreže ih ne mogu kontaktirati. Svatko od njih ima svoju IP adresu, i oni međusobno mogu komunicirati. Vanjski korisnik može jedino komunicirati s poslužiteljima ili aplikacijama koje se vide izvana. Unutarnji poslužitelji mogu sadržavati osjetljive informacije, ili čak sadržavati admin sučelje za aplikaciju koja jest vidljiva izvana. Problem nastaje ako zlonamjerni korisnik uspije kontaktirati jedan od unutarnjih poslužitelja i uspije saznati osjetljive podatke ili promijeniti ključne postavke. Primjerice, zlonamjerni korisnik bi mogao pristupom admin sučelju obrisati ostale korisnike aplikacije.
Aplikacija opisana u nastavku sadrži SSRF ranjivost koja omogućava zlonamjernom korisniku kontaktiranje unutarnjeg poslužitelja i dohvaćanje sadržaja njegove početne stranice. Iskorištavanje SSRF ranjivosti na ovaj način prikazano je slikom 1.7.



Slika 1.7: Kontaktiranje lokalnih poslužitelja iskorištavajući SSRF ranjivost

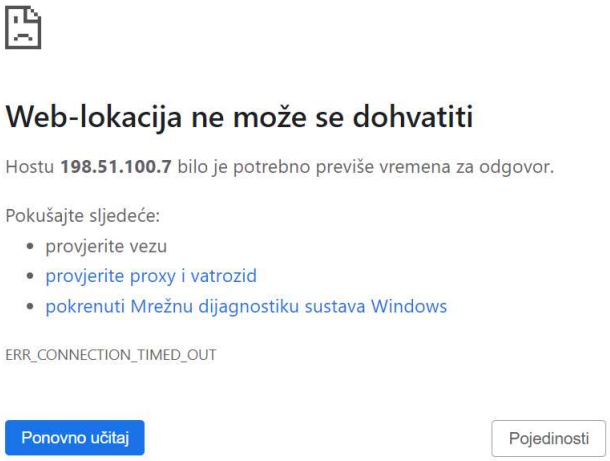
Za potrebe primjera korišten je `docker-compose` [4] [6] kako bi se definirala lokalna mreža. Napravljena su dva poslužitelja i svakom je dana jedna IP adresa unutar te lokalne mreže. Međusobno, oni imaju mogućnost komunikacije.

IP adresa `server 1`: `198.51.100.2`
IP adresa `server 2`: `198.51.100.7`

Jedan poslužitelj (`server 1`) je onaj koji poslužuje aplikaciju vidljivu izvana, dok drugi

[6] Alat za pokretanje više Docker kontejnera

poslužitelj (`server 2`) poslužuje aplikaciju koja u ovom primjeru glumi admin sučelje naše aplikacije.
Kako bi se prvi poslužitelj mogao kontaktirati izvana, napravljeno je preusmjeravanje priključka (engl. *port forwarding*).
Nakon toga, aplikacije je bila dostupna na IP adresi `198.51.100.2` i preko URL-a `http://localhost:7001`. Budući da nije napravljeno preusmjeravanje priključka za drugi poslužitelj, odnosno aplikaciju koja glumi admin sučelje, ona nije dostupna vanjskim korisnicima kao što se vidi na slici 1.8.

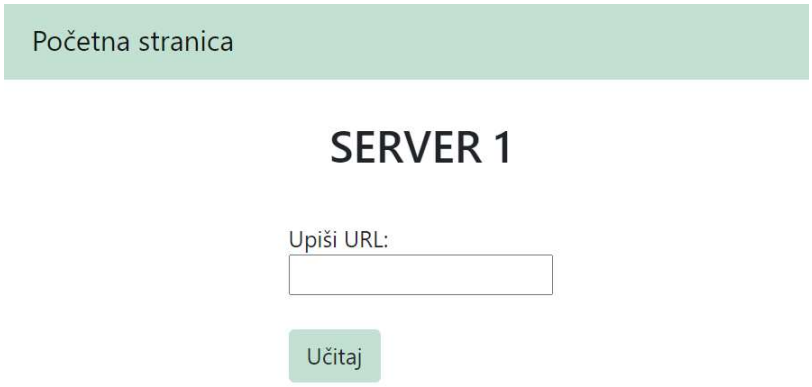


Slika 1.8: Admin sučelje nije dostupno vanjskim korisnicima

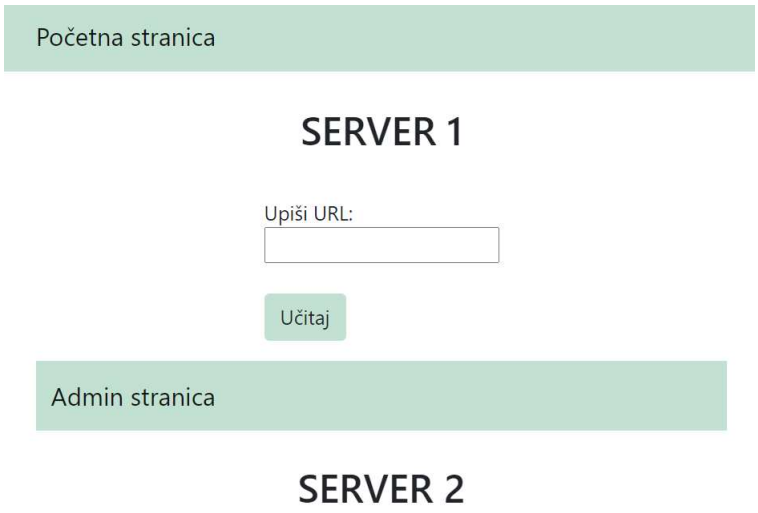
Kao što se vidi na slici 1.9 aplikacija se sastoji jedino od početne stranice na kojoj korisnik može poslati URL neke stranice.
Nakon toga, aplikacija će napraviti HTTP zahtjev prema tom URL-u i ispisati u nastavku njezin sadržaj. U slučaju da zlonamjerni korisnik slučajno zna privatne IP adrese unutarnjih poslužitelja i sazna da se na jednoj od njih nalazi admin sučelje trenutne aplikacije, on može poslati specijalno modificirani URL kako bi dohvatio sadržaj admin sučelja. Na slici 1.10 može se vidjeti da se sadržaj admin sučelja uspješno dohvaća nakon što se pošalje modificirani URL.

Što se zapravo događa?

Nakon što se pošalje URL, aplikacija dohvati sadržaj poslanog URL-a. Problem nastaje zato što aplikacija prije dohvaćanja sadržaja ne provjerava ni na koji način što je taj poslani



Slika 1.9: Početna stranica aplikacije



Slika 1.10: Zlonamjerni korisnik je uspio kontaktirati unutarnji poslužitelj

URL. U slučaju da zlonamjerni korisnik pošalje privatnu IP adresu koja se odnosi na neki poslužitelj u lokalnoj mreži, aplikacija koja se poslužuje na poslužitelju koji je dio te lokalne mreže će moći bez ikakvog problema kontaktirati poslanu IP adresu, odnosno vratiti sadržaj poslanog URL-a, iako vanjski korisnik ne bi smio moći pristupiti tom URL-u. U nastavku je dan programski kod iz kojeg se može vidjeti način na koji se dohvaća sadržaj poslanog URL-a, kao i manjak njegove provjere.

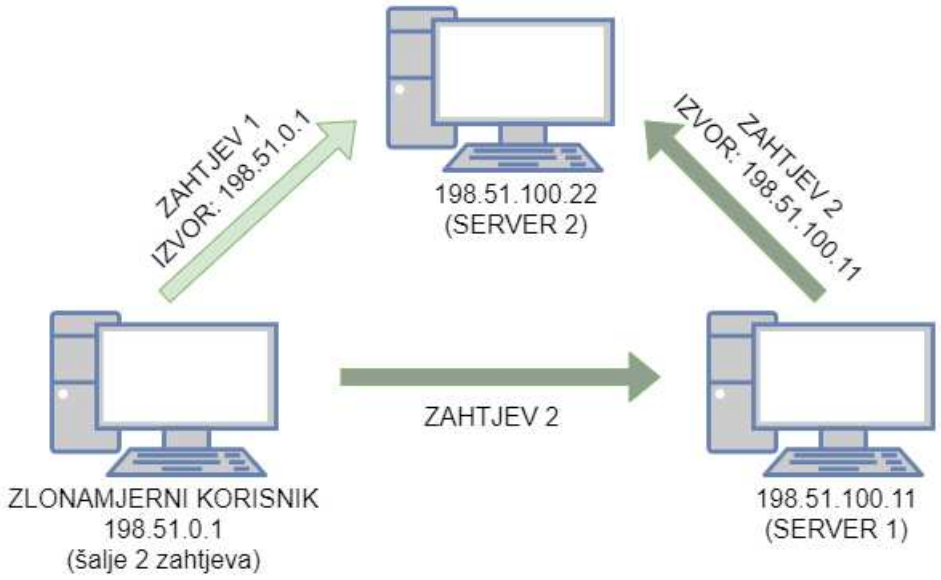
```
<form method="GET">
    <label > URL:</label>
    <input type="text"  name="url" >
    <input type="submit" value="Submit">
</form>

<?php

if(isset($_GET['url'])){
    $url = $_GET['url'];
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $result_data = curl_exec($ch);
    curl_close($ch);
    echo $result_data;
}
?>
```

Kontaktiranje vanjskih poslužitelja

Neka postoje dvije aplikacije, A i B, gdje se aplikacija A poslužuje na poslužitelju A, a aplikacija B se poslužuje na poslužitelju B. Obje aplikacije su dostupne izvana, odnosno vanjski korisnici mogu slobodno pristupiti i jednoj i drugoj aplikaciji. Svaki poslužitelj prati promet koji dolazi prema njemu. U datotekama dnevnika (engl. *logs*) se lako može iščitati s koje IP adrese je došao promet.
Uzmimo jednog korisnika čija je IP adresa `198.51.100.2`. Ako on posjeti aplikaciju A, tada će u datotekama dnevnika aplikacija biti zapis da je promet došao sa IP adrese `198.51.100.2`. Analogno se dogodi ako korisnik posjeti aplikaciju B.
Problem nastaje ako aplikacija A sadrži SSRF ranjivost. Ona omogućuje zlonamjernom korisniku da lažira zahtjev prema aplikaciji B. S posebno modificiranim URL-om, on uvjeri aplikaciju A da napravi zahtjev prema aplikaciji B. U datotekama dnevnika aplikacije B će biti zapisano da je promet došao upravo s IP adrese poslužitelja koji poslužuje aplikaciju

A, a ne s IP adrese zlonamjernog korisnika. Zlonamjerni korisnik bi mogao to iskoristiti da primjerice započne napad uskraćivanjem usluga (engl. *denial of service*) nad poslužiteljom B, ili može iskoristiti neku drugu ranjivost aplikacije B. Zbog zapisa u datotekama dnevnika, glavni krivac za napad će biti poslužitelj A.

Aplikacija (radi lakšeg snalaženja u nastavku `aplikacija A`) opisana u nastavku sadrži SSRF ranjivost koja omogućuje zlonamjernom korisniku kontaktiranje neke druge aplikacije (radi lakšeg snalaženja u nastavku `aplikacija B`) u ime `aplikacije A`. Tada će zapis u datotekama dnevnika `aplikacije B` ukazivati da je napad izazvala `aplikacija A`. Iskorištavanje SSRF ranjivosti na ovaj način je prikazano je slikom 1.11.



Slika 1.11: Kontaktiranje vanjskih poslužitelja iskorištavanjem SSRF ranjivosti

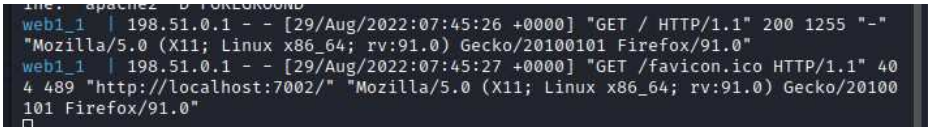
Za potrebe primjera, korišten je `docker-compose`. Kreirana su dva poslužitelja, u nastavku `server A` i `server B`. `server A` je posluživao `aplikaciju A`, dok je `server B` posluživao `aplikaciju B`. Definirana je lokalna mreža, te je svaki poslužitelj dobio jednu IP adresu.

IP adresa `server 1`: `198.51.100.11`
IP adresa `server 2`: `198.51.100.22`

`Aplikaciji A` se može pristupiti preko URL-a `http://localhost:7002`, kao i

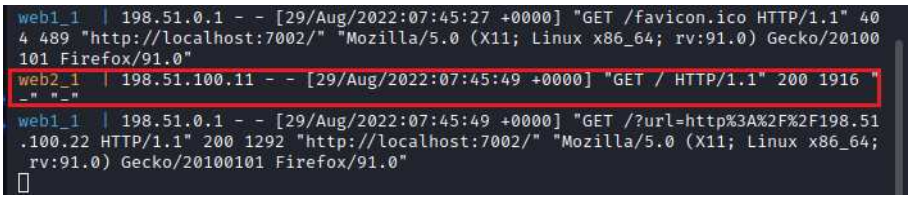
preko IP adrese `198.51.100.11`. Pristupom aplikaciji, unutar datoteka dnevnika će biti zapisano kako je promet došao sa IP adrese `198.51.0.1`.

Za potrebe primjera, uzimajući u obzir način rada `Dockera` IP adresu `198.51.0.1` ćemo smatrati IP adresom zlonamjernog korisnika.
`aplikacija A` jednako funkcionira i izgleda kao aplikacija opisana u prethodnom primjeru, pa u nastavku neće biti navedena slika početne stranice. Isto tako, aplikacija radi i datoteke dnevnika. Svaki put kad netko posjeti `aplikaciju A`, u datotekama dnevnika možemo vidjeti IP adresu korisnika koji je posjetio tu aplikaciju. Na slici 1.12 je prikazan zapis u datotekama dnevnika ako (zlonamjerni) korisnik, s IP adresom `198.51.0.1` posjeti `aplikaciju A`.



Slika 1.12: Datoteka dnevnika prikazuje IP adresu korisnika koji je posjetio aplikaciju

Na isti način funkcionira i `aplikacija B`, kao i njezine datoteke dnevnika.
`aplikacija A`, kao i u prethodnom primjeru, na početnoj stranici ima formu u koju se može upisati neki URL. Nakon što korisnik upiše URL, `aplikacija A` će napraviti HTTP zahtjev prema tom URL-u te vratiti sadržaj URL-a.
Problem nastaje ako zlonamjerni korisnik (IP adresa `198.51.0.1`) upiše URL `aplikacije B`. U datotekama dnevnika `aplikacije B` će tada biti zapisano da je promet došao sa IP adrese `198.51.100.11`. Na slici 1.13 se može vidjeti zapis u datotekama dnevnika `aplikacije B` nakon što je zlonamjerni korisnik upisao njezin URL u formu `aplikacije A`, izazivajući na taj način da `aplikacija A` napravi HTTP zahtjev prema `aplikaciji B`.



Slika 1.13: Datoteka dnevnika nakon što je zlonamjerni korisnik upisao modificirani URL

Što se zapravo događa?

Nakon što se pošalje URL, `aplikacija A` bez ikakve provjere pomoću funkcije `curl` napravi HTTP zahtjev prema URL-u. I ovo je ono što je ključno u ovom primjeru. Odredište će vidjeti da je zahtjev napravila `aplikacija A`, bez obzira na koji je način došlo do tog HTTP zahtjeva. Promatrajući izvana, korisnik koji je poslao URL zna da je on potaknuo aplikaciju da kreira i pošalje HTTP zahtjev. Ali `aplikacija B` to ne zna. Iz perspektive `aplikacije B`, HTTP zahtjev je napravila `aplikacija A`. U nastavku je dan programski kod kojim je napravljen prethodno opisan primjer.

```
<form method="GET">
    <label> URL:</label>
    <input type="text" name="url" >
    <input type="submit" value = "Submit">
</form>

<?php

if(isset($_GET['url'])){
    $url = $_GET['url'];
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $result_data = curl_exec($ch);
    curl_close($ch);
    echo $result_data;
}
?>
```

1.3 Vrste SSRF napada

Iako postoje različiti scenariji u kojima se može pojaviti SSRF ranjivost (kao što je viđeno u prethodnim primjerima) i različite posljedice SSRF napada (nešto više o tome će biti rečeno u sljedećem poglavlju), svi SSRF napadi se mogu podijeliti u dvije skupine: **obični SSRF napadi** (engl. *common SSRF attacks*) i **slijepi SSRF napadi** (engl. *blind SSRF attacks*).
U sljedećem poglavlju će isto tako biti detaljnije navedeni koraci za pronalazak SSRF ranjivosti, bez obzira bila ona običan SSRF ili slijepi SSRF.

Običan SSRF napad

Nakon što zlonamjerni korisnik izvrši običan SSRF napad, odnosno nakon što ranjiva aplikacija napravi HTTP zahtjev prema nekoj lokaciji, odgovor koji primi natrag će se prikazati zlonamjernom korisniku. Svi primjeri u ovom poglavlju pripadaju ovoj kategoriji, no budući da su bili napravljeni kako bi se na jednostavan i razumljiv način demonstrirala SSRF ranjivost, namjerno je isprogramirano prikazivanje odgovora zlonamjernom korisniku. U stvarnom svijetu, može se dogoditi da isti scenariji poput prethodno opisanih primjera ne reproduciraju nikakav odgovor.

Slijepi SSRF napad

Nakon što zlonamjerni korisnik izvrši slijepi SSRF napad, odnosno nakon što ranjiva aplikacija napravi HTTP zahtjev prema nekoj lokaciji, zlonamjernom korisniku neće biti prikazan odgovor. Upravo zbog tog razloga, slijepi SSRF je teže iskoristit od običnog SSRF-a, no on može voditi voditi do udaljenog izvršavanja koda na lokaciji prema kojoj je poslan HTTP zahtjev.

Poglavlje 2

SSRF napadi

Traženje ranjivosti u aplikaciji zahtijeva izvođenje raznih provjera, povezivanje znanja raznih sfera računalnog sustava poput mreža ili programiranja, i ponekad korištenja alata koji pomažu u pronalasku ranjivosti. Težina uočavanja SSRF ranjivosti ovisi o vrsti SSRF ranjivosti, te o načinu na koji je aplikacija napravljena.
Prvi dio ovog poglavlja će biti namijenjen opisu postupka pronalaska SSRF ranjivosti.

SSRF ranjivost nije nužno ozbiljna ranjivost aplikacije. Ako aplikacija ima SSRF ranjivost, ali napadač nema načina iskoristi je, tada je ta ranjivost bezopasna. Problem nastaje kad postoje ostale ranjivosti koje postanu iskoristive nakon iskorištavanja SSRF ranjivosti. Većina ovog poglavlja će biti posvećena upravo opisivanju raznih scenarija u kojima SSRF ranjivosti rezultira zapravo lancem ranjivosti. Njegovim iskorištavanjem se može naštetiti sigurnosti aplikacije i integritetu osjetljivih podataka.

2.1 Otkrivanje SSRF ranjivosti

Pronalazak ranjivosti može obavljati napadač ili osoba koja je zadužena za testiranje aplikacije ili nekog sustava. Osoba koja vrši testiranje se zove penetracijski tester, ili skraćeno pen tester. Pen tester ima dopuštenje korištenja metoda i alata koji služe za pronalazak ranjivosti. Testiranje koje pen tester obavlja može bit *white box* testiranje ili *black box* testiranje. Postoji razlika između *white box* i *black box* testiranja. *White box* testiranje sustava ili aplikacija je testiranje gdje osoba koja ih testira poznaje unutarnju strukturu, te ima pristup izvornom kôdu. S druge strane, kod *black box* testiranja, osobi koja testira nisu poznate nikakve dodatne informacije o sustavu ili aplikaciji. Tester pristupa aplikaciji kao običan korisnik s ciljem pronalaska ranjivosti.
Postupak pronalaska SSRF ranjivosti je u oba slučaja sličan, no pri *white box* testiranju, tester ima dodatnu mogućnost pregleda izvornog kôda. Pomoću njega, on može usmjeriti

svoju pozornost na neke dijelove aplikacije ili sustava kojima inače ne bi odmah pridavao pozornost.
Za temeljito otkrivanje ranjivosti, potrebno je najprije provjeriti sve funkcionalnosti aplikacije iz korisničke perspektive kako bi se dobio dojam na koji način aplikacija funkcionira. Konkretno, za otkrivanje SSRF ranjivosti, potrebno je probati naći neku funkcionalnost ili dio aplikacije koji bi mogao koristiti korisnički input za URL. Pregledom tog dijela te presretanjem zahtjeva bi se moglo vidjeti koristi li se samo dio inputa za stvaranje putanje prema resursu ili se koristi cijeli input kao putanja. Naravno, to ne mora značiti da zaista postoji SSRF ranjivosti, budući da se u tom trenutku ne zna kako funkcionira pozadinski dio aplikacije (engl. *backend*) i što on radi s inputom.
Tester koji ima pristup kôdu u tom trenutku može i provjeriti što se zapravo događa s tim inputom prateći tôk u programskom kôdu. Nakon što se locira mjesto na kojem bi aplikacija mogla biti ranjiva na SSRF, bez obzira ima li on pristup programskom kôdu ili ne, tester obavlja dinamičku analizu kako bi uspješno izazvao SSRF napad. U ovom slučaju *white box* testiranje može biti korisno jer tester može imati bolji uvid u način kako izazvati SSRF ranjivost.

To ne znači da ne postoje dijelovi u aplikaciji za koje nije odmah očito da mogu biti ranjivi na SSRF. U tome može pomoći analiza programskog kôda ili dinamička analiza. Općenito, iako možda dosadan i dugotrajan proces, pregled programskog kôda i shvaćanje kako *backend* aplikacije funkcionira može pomoći u pronalasku ranjivosti (ne samo SSRF) koje nisu odmah očite iz funkcionalnosti aplikacije.

Nakon što se otkrije SSRF ranjivost, potrebno je probati iskoristiti tu ranjivost za izazivanje nekakve štete. Šteta može uključivati dohvaćanje osjetljivih podataka, promjena nekakvih konfiguracija ili podataka, iskorištavanje neke ranjivosti, pristup lokacijama kojima inače nije dopušten pristup i ostalo.
Potrebno je napomenuti da tester koji pokušava pronaći i izvesti razne ranjivosti radi to s ciljem poboljšavanja sigurnosti aplikacije. Nakon što pronađe ranjivosti, njegova je dužnost prijaviti ih i savjetovati kako bi se te ranjivosti mogle riješiti. Napadač u drugu ruku otkriva i iskorištava ranjivosti radi sebe. U ovom dijelu, kad god se govori o osobi koja otkriva i iskorištava ranjivosti, misli se na testera, a ne na napadača. U slučaju da se misli na napadača, to će biti posebno naznačeno.

Konkretno, za iskorištavanje SSRF ranjivosti, potrebno je konstruirati URL kojim će se izvršiti napad i doći do krajnjeg cilja, bilo to dohvaćanje osjetljivih podataka ili mijenjanje nekih postavki.
Zlonamjeran sadržaj poruke (engl. *malicious payload*) može biti raznih oblika. U prvom poglavlju konstruiran URL je bio oblika

`file:///secrets/important_secret.txt`. Unutar njega se može primijetiti korištenje *file* šeme. Naime, postoje različite šeme koje se mogu koristiti kako bi se došlo do željenog cilja. Ovisno o tome kakav napad se želi pokrenuti i o dostupnosti usluga koje se žele napasti, potrebno je koristiti drugačije URL-ove. Šema zapravo govori koji će se protokol koristiti nad nekim resursom. Ponekad je ime šeme i protokola jednako, pa se koriste naizmjenično, iako tehnički nisu iste stvari.

Posebno kreirani URL-ovi
Iako se posebni URL-ovi za napad mogu kreirati ručno, putem interneta se mogu pronaći već kreirane liste i alati koji ih posebno generiraju. Ovisno o vrsti napada koji se želi pokrenuti, te koje su usluge dostupne, potrebno je odabrati pravi posebno kreiran URL.
Jedan od posebno kreiranih URL-a je i onaj koji iskorištava različito enkodiranje. Takvi URL-ovi pomažu u zaobilaženju filtra kojem je zadaća spriječiti SSRF napad. No o tome će biti više napisano u sljedećem poglavlju, stoga se neće ovdje posebno objašnjavati.

Iako najprije padaju na pamet, `http` i `https` šeme nisu jedine preko kojih se može izvršiti SSRF napad. Postoje i druge, i za svaku postoji drugačiji URL koji se može poslati kako bi se izvršio SSRF napad. Primjerice, `http` šema zapravo nalaže korištenje `http` protokola nad resursom, dok `https` šema nalaže korištenje http-preko-TLS-a.

Neke od drugih šema koje se mogu koristiti za izvođenje SSRF napada su: **file** šema, **Gopher** šema, **LDAP** šema, **dict** šema i **SFTP** šema.

2.2 Iskorištavanje SSRF ranjivosti

Skeniranje lokalne mreže

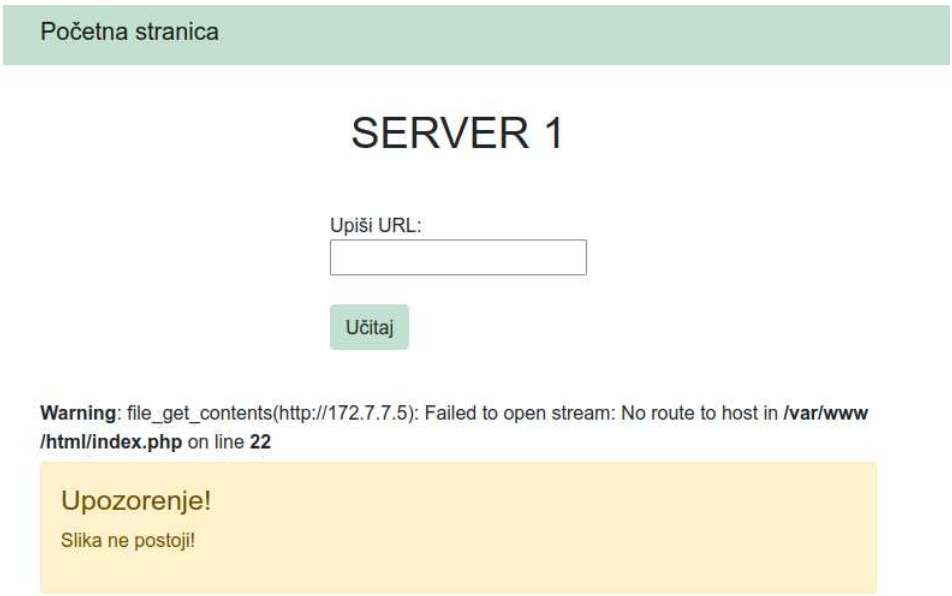
Neka se *web* aplikacija poslužuje na poslužitelju koji je dio neke veće mreže. Ta aplikacija je dostupna izvana, i ranjiva je na SSRF. Zamislimo da osoba izvana, napadač u ovom slučaju, ne zna ništa o strukturi mreže u kojoj se nalazi poslužitelj, ili o poslužitelju koji poslužuje aplikaciju. Dakle napadač ne zna što se nalazi na poslužitelju ili koje su usluge pokrenute na njemu. No, budući da je aplikacija ranjiva na SSRF, on može iskoristiti tu ranjivost kako bi saznao te stvari.
Naime, ovisno kako je aplikacija programirana, ona može davati različite odgovore na zahtjeve koji su uspješni, i na zahtjeve koji nisu uspješni. To znači da osoba može saznati opseg IP adresa koje se koriste u internoj mreži, ili servise koji se nalaze na poslužitelju. Korištenjem alata, napadač može slati zahtjeve jedan za drugim. Međusobno, ti zahtjevi su jednaki, osim što svaki ima drugačiji URL. Svaki od njih ima drugačiju IP adresu u nekom opsegu, gdje svaka može biti potencijalna IP adresa nekog unutarnjeg poslužitelja,

ili drugačiji priključak, gdje svaki može biti potencijalni priključak nekog servisa.
Osim što odgovori na zahtjeve mogu biti sadržajno drugačiji, isto tako promatranjem vremena odgovora može se odrediti koji priključak je otvoren, a koji nije. Nakon što se otkrije otvoreni priključak, napadač može na temelju toga probati odrediti koji servis se pokreće. Primjerice ako je otvoren priključak 22, tada je vjerojatno dostupan SSH servis. Jako je bitno napomenuti da se servisi mogu posluživati na bilo kojem priključku iako za svaki servis postoji priključak na kojem se on u prvotnoj konfiguraciji (engl. *default*) poslužuje.

Za potrebe primjera ove vrste iskorištavanja SSRF ranjivosti, uzet je primjer iz prethodnog poglavlja s malim preinakama. U mreži su dostupna dva poslužitelja, `poslužitelj A` koji je dostupan izvana i `poslužitelj B` koji nije dostupan izvana.

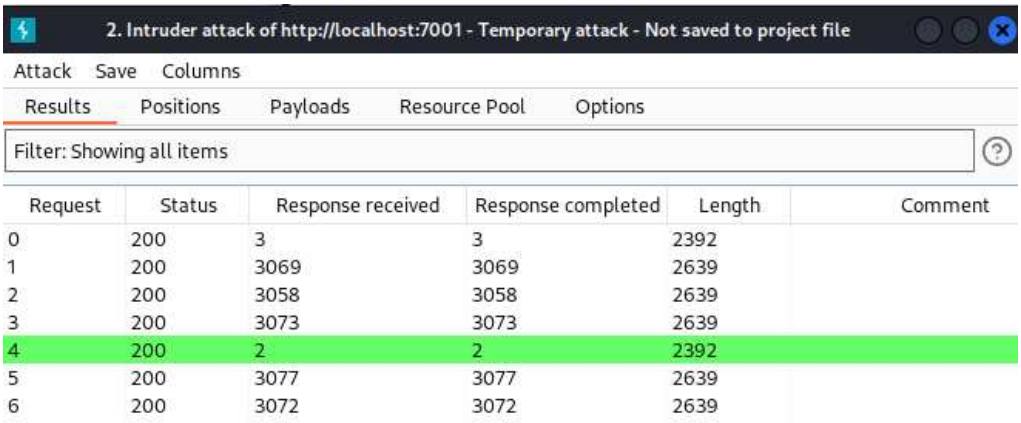
IP adresa `poslužitelja A` je: `198.51.100.2`.
IP adresa `poslužitelja B` je: `198.51.100.7`.

`poslužitelj A` poslužuje aplikaciju koja učitava slike s poslanog URL-a te ih nakon toga prikazuje korisniku. Ako se s poslanog URL-a ne može dohvatiti nikakav sadržaj, tada se javlja greška, što je prikazano na slici 2.1.



Slika 2.1: Neuspješno dohvaćanje sadržaja poslanog URL-a

Korištenjem alata, napadač može zapravo slati niz zahtjeva kako bi otkrio na kojim unutarnjim IP adresama se nalaze poslužitelji. Za potrebe primjera, poslano je samo šest zahtjeva, no u stvarnosti, lista IP adresa ili priključaka koji se isprobavaju je puno veća.
IP adrese koje su isprobane su bile u opsegu `198.51.100.4` do `198.51.100.9`.
Na slici 2.2 se može vidjeti razlika u odgovorima. Za IP adresu `198.51.100.7` odgovor na zahtjev je bio drugačiji nego za ostale IP adrese. Isto tako, može se vidjeti kako je vrijeme koje je bilo potrebno za odgovor manje nego za zahtjeve poslane na druge IP adrese.



Request	Status	Response received	Response completed	Length	Comment
0	200	3	3	2392	
1	200	3069	3069	2639	
2	200	3058	3058	2639	
3	200	3073	3073	2639	
4	200	2	2	2392	
5	200	3077	3077	2639	
6	200	3072	3072	2639	

Slika 2.2: Različiti odgovori

I zaista, ako se pogleda sučelje aplikacije nakon poslanog zahtjeva s URL-om `198.51.100.7`, ne pojavljuje se greška, te se korisniku, u ovom slučaju napadaču, prikaže ”slika”. Ta slika je zapravo sadržaj početne stranice aplikacije koja se poslužuje na `poslužitelju B`. Potrebno je napomenuti kako to neće uvijek biti odgovor. U ovom slučaju jest, jer je za potrebe primjera tako programirano.
Početna stranica aplikacije koja se poslužuje na `poslužitelju B` u ovom slučaju sadrži i administracijske vjerodajnice. One se mogu iskoristiti za daljnje napade. To samo pokazuje kako SSRF ranjivost može otvoriti vrata iskorištavanju drugih, opasnijih ranjivosti.

Korištenje funkcionalnosti koje izvorno nisu dostupne običnom korisniku

Dijelovi aplikacije se ne moraju nalaziti na istom poslužitelju. Dio koji se bavi bazama podataka, admin sučelje i ostali dijelovi se mogu nalaziti na posebnim poslužiteljima u

nekoj unutarnjoj mreži. Ako je aplikacija ranjiva na SSRF, te napadač iskorištavanjem te ranjivosti otkrije da ima pristup primjerice admin sučelju, ima mogućnost iskorištavanja funkcionalnosti koje su dostupne samo putem admin sučelja.

Radi boljeg objašnjavanja ovog scenarija, preuzet je primjer iz prvog poglavlja, te je napravljeno par preinaka. Prije objašnjavanja rada same aplikacije, potrebno je napomenuti kako je poanta ovog primjera pokazivanje SSRF ranjivosti, te način njenog iskorištavanja. Zato su svi dijelovi aplikacije isprogramirani na što jednostavniji način. U stvarnosti, aplikacije nisu napravljene na taj način, ali logika je ista, i naravno, SSRF ranjivost i način njenog iskorištavanja je isti.

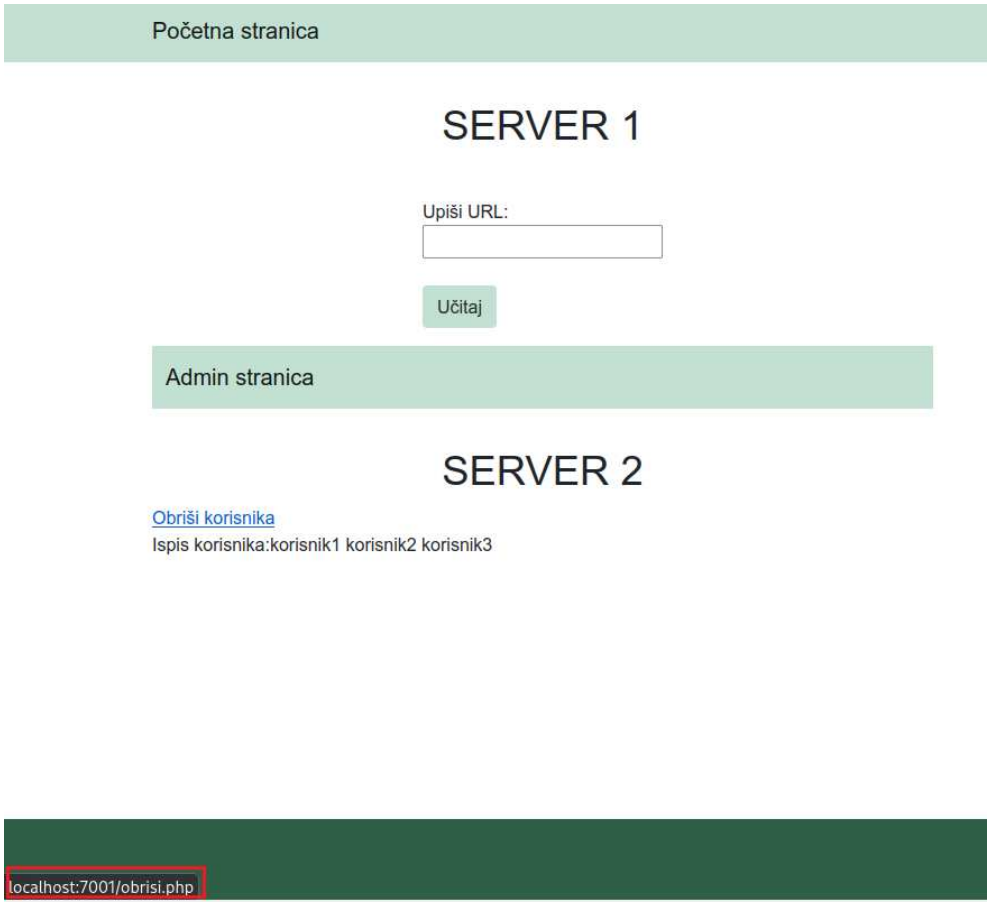
Napravljena su dva poslužitelja, `poslužitelj A` i `poslužitelj B`.
`poslužitelj A` je dostupan izvana, dok `poslužitelj B` nije.
`poslužitelj B` je dostupan `poslužitelju A`.

IP adresa `poslužitelja A` je: `198.51.100.2`.
IP adresa `poslužitelja B` je: `198.51.100.7`.

Na `poslužitelju A` se poslužuje aplikacija koja dohvaća sadržaj stranice poslanog URL-a te ga prikazuje na početnoj stranici. `poslužitelj B` u ovom primjeru igra ulogu admin sučelja aplikacije. Dodatno, neka aplikacija ima bazu podataka koja se nalazi na `poslužitelju B`, te se preko admin sučelja mogu brisati korisnici baze podataka.
U ovom slučaju tekstualna datoteka igra ulogu baze podataka, gdje svaka linija predstavlja jednog korisnika. To je dovoljno dobra implementacija za ovaj primjer, iako se baza podataka tako zapravo ne programira. Na početku su u bazi tri korisnika: **korisnik1**, **korisnik2**, **korisnik3**.
Admin sučelje je jednostavno. Sadrži gumb čijim pritiskom brišemo jednog korisnika iz baze podataka (**korisnik1**). Radi jednostavnosti, gumb je napravljen zapravo kao poveznica koja vodi prema datoteci `obrisi.php` koja obavlja brisanje jednog korisnika.
Time, URL koji je potreban za brisanje jednog korisnika je:
`http://198.51.100.7/obrisi.php`.
Ispod gumba se nalazi popis svih korisnika koji su trenutno u bazi podataka.

Za potrebe primjera, recimo da je napadač već napravio neki osnovni pregled aplikacije, pronašao lokaciju gdje bi mogao iskoristiti SSRF ranjivost. Isto tako, znao je da se aplikacija poslužuje na poslužitelju koji je dio neke mreže, te je napravio skeniranje IP adresa za neke unutarnje poslužitelje (kao što je objašnjeno u prethodnom primjeru). Na kraju, potrebno je vidjeti može li se što napraviti sada kada je pristup IP adresi `198.51.100.7` moguć.

Nakon što se pošalje samo URL `http://198.51.100.7`, ispiše se cijelo admin sučelje, no kao što se vidi, svi korisnici baze su još uvijek prisutni. Dakle, iako je iskorištena SSRF ranjivost, nije napravljena nikakva šteta.
Na slici 2.3 se može vidjeti ispis admin sučelja (budući da je to funkcionalnost aplikacije u ovom primjeru). Isto tako, budući da se ispisuje cijelo admin sučelje, može se vidjeti i sve funkcionalnosti i elementi admin sučelja koji su prethodno opisani.
Pristupom admin sučelju, može se vidjeti na koji način se obavlja brisanje korisnika. U ovom slučaju, zbog načina na koji je aplikacija napravljena, URL koji je potreban za brisanje korisnika može se dobiti prelaskom miša preko "gumba" (točnije je reći poveznice u ovom slučaju) u ispisu admin stranice. Prikaz URL-a će se pokazati pri dnu stranice, kao što je prikazano na slici 2.3.



Slika 2.3: Otkrivanje URL-a za brisanje korisnika

Sada, ako se u aplikaciji pošalje URL `http://198.51.100.7/obrisi.php`, zahtjev će biti poslan prema `poslužitelju B`, budući da je on dostupan `poslužitelju A`. Tada će on izvršiti brisanje korisnika. Nakon ponovnog provjeravanja stanja baze podataka, jedini korisnici koji će biti dio nje su **korisnik2** i **korisnik3**.

SSRF i DOS

Iskorištavanje SSRF ranjivosti može dovesti i do uskraćivanja usluga (engl. *denial of service*, DOS).
Naime, neka se na nekom poslužitelju poslužuje aplikacija koja je ranjiva na SSRF. Aplikacija je dostupna izvana. Neka je taj poslužitelj dio neke mreže u kojoj se nalaze i neki drugi poslužitelji koji nisu dostupni izvana. Poslužitelj koji poslužuje aplikaciju dostupnu izvana ima mogućnost komunikacije s unutarnjim poslužiteljima.
Budući da unutarnji poslužitelji komuniciraju jedino međusobno, ne očekuju veliku količinu prometa, te imaju manji kapacitet propusnosti (engl. *bandwidth*). Ako napadač može iskorištavanjem SSRF ranjivosti kontaktirati neki unutarnji poslužitelj, on isto tako može kreirati i poslati niz zahtjeva prema tom unutarnjem poslužitelju. Time će zauzeti sav dostupan kapacitet propusnosti te će prekinuti funkcionalnost tog poslužitelja.

Dohvaćanje osjetljivih podataka

Kao što je već bilo prikazano i u primjerima, pomoću SSRF se može doći i do osjetljivih podataka. Ti podaci mogu nekad biti i vjerodajnice pomoću kojih se onda može dobiti admin pristup ili izvršavati neke funkcije koje ne bi trebale biti dostupne običnim korisnicima.
Ovisno o tome kako je aplikacija koja je ranjiva na SSRF napravljena, te što se nalazi na poslužitelju koji je poslužuje, do vjerodajnica se može doći pristupom lokalnim datotekama. Isto tako, unutarnji poslužitelji i usluge koje se mogu nalaziti na unutarnjim poslužiteljima ne očekuju komuniciranje s korisnicima izvana, pa je ponekad autorizacija izostavljena. Upravo zbog toga, napadač ih može kontaktirati i doći do osjetljivih podataka.
U zadnje vrijeme, aplikacije se mogu pokretati i u oblaku (engl. *cloud*). Jedan od takvih oblaka je i AWS (engl. *Amazon Web Services*) [1]. Ako osoba koja postavlja AWS okruženje ne promijeni prvotne postavke, vjerodajnice za AWS se nalaze uvijek na istom mjestu. Osim vjerodajnica, tamo se nalazi još osjetljivih podataka. Kako bi se pristupilo tim podacima, potrebno je posjetiti `http://169.254.169.254` ili `http://instance-data`. Naravno, obični korisnici ne bi trebali moći doći do tih podataka. No, ako postoji SSRF ranjivost u aplikaciji koja se poslužuje u AWS okruženju i lokacija AWS vjerodajnica nije promijenjena, tada napadač može doći do njih, i iskoristiti

ih dalje u napadu.

U srpnju 2019., kompanija Capital One je objavila kako je došlo do curenja podataka više od 100 milijuna osoba u SAD-u, i oko 6 milijuna u Kanadi [15]. Napadač (ili napadači) su uspjeli doći do osjetljivih podataka kao što su bankovni računi i SIN-ovi (engl. *Social Insurance Number*) iskorištavajući SSRF ranjivost. Pomoću nje, napadač je izvukao AWS pristupne ključeve (engl. *access key*). Njih je iskoristio kako bi sinkronizirao svoj lokalni disk sa S3 AWS skladištem gdje su se nalazili osjetljivi podaci.

SSRF i RCE

RCE (*Remote Code Execution*) je vrlo opasna ranjivost. Kao što i samo ime govori, napadač ima mogućnost izvršavanja bilo koje naredbe (*code execution*) iz daleka (*remote*) bez izravnog fizičkog pristupa tom sustavu. Putem RCE-a, žrtva može čak i izgubiti kontrolu nad računalom. RCE isto tako može dovesti i do krađe osjetljivih podataka.

SSRF ranjivost u nekim slučajevima može omogućiti i dobivanje pristupa ljusci (engl. *shell*) na poslužitelju koji poslužuje aplikaciju ranjivu na SSRF, ili na nekom unutarnjem poslužitelju. Jednom kad to uspije, napadač može izvršavati bilo koju naredbu na poslužitelju.

Poglavlje 3

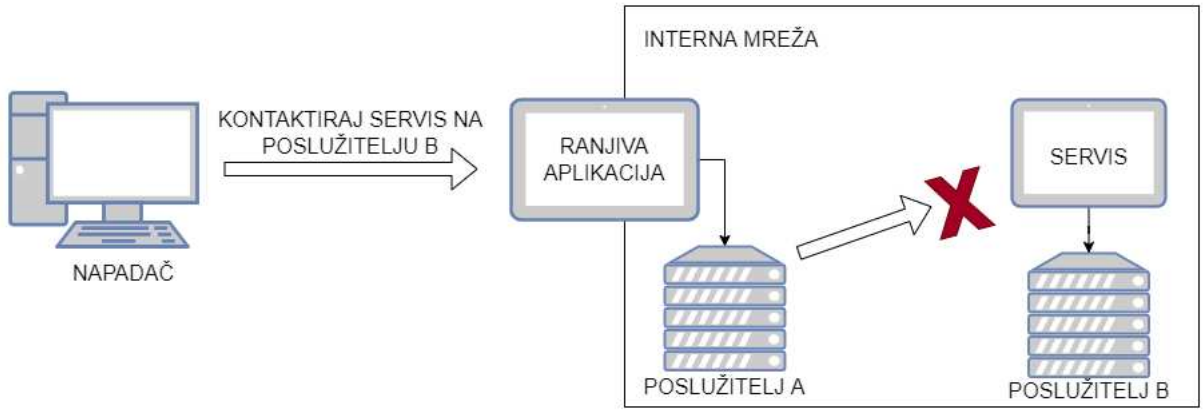
Zaštita od SSRF napada

SSRF ranjivost nije toliko poznata poput napada umetanjem SQL koda (engl. *SQL injection*) ili XSS napada (engl. *Cross Site Scripting attack*). Diverto je 2021. godine objavio izvješće "o stanju informacijske sigurnosti u Hrvatskoj" [2] u kojem se vidi kako je samo 1 % aplikacija ranjivo na SSRF napad. Usprkos tome, ona može biti početak vrlo opasnog lanca ranjivosti.
Kao što je već spomenuto, ako je neka aplikacija ranjiva na SSRF napad, nije nužno da napadač može napraviti ikakvu štetu. No, ovisno o infrastrukturi sustava i o ranjivostima uređaja koji su dio te infrastrukture, zbog SSRF ranjivosti napadač može doći do osjetljivih podataka, ili preuzeti kontrolu nad nekim uređajima.
Naravno, postoje mjere kojima se sprječava ili barem smanjuje opasnost od SSRF napada. Nažalost, nekad ni mjere ne mogu spriječiti iskorištavanje SSRF ranjivosti. U nastavku će biti navedene mjere koje bi pomogle u zaštiti od SSRF-a, kao i načini na koje se mogu zaobići mjere zaštite.

3.1 Onemogućavanje pristupa bez vjerodajnica

Aplikacije koje se poslužuju na poslužiteljima koji su dio neke interne mreže mogu biti sredstvo kojim napadač izvršava napad na poslužitelj koji poslužuje aplikaciju, ali i na poslužitelje i servise u internoj mreži.
Kao što je već spomenuto, između poslužitelja u internoj mreži postoji veza povjerenja, te je vrlo često na njima, kao i na servisima koji se na njima nalaze, autentifikacija i autorizacija relaksirana. Kao mjeru predstrožnosti, pristup unutarnjim servisima i poslužiteljima bi trebao biti zaštićen na način da samo korisnici ili servisi koji posjeduju vjerodajnice imaju omogućen pristup. Na taj način, čak i ako napadač pošalje modificiran zlonamjerni URL, pristup prema nekim unutrašnjim resursima će biti onemogućen, budući da su za pristup potrebne vjerodajnice.

Zamislimo primjer kao na slici 3.1. Na `poslužitelju A` se poslužuje aplikacija ranjiva na SSRF. `poslužitelj A` je dio interne mreže u kojoj se nalazi i `poslužitelj B`. `poslužitelj B` sadrži jedan servis kojemu je pristup omogućen jedino pomoću vjerodajnica. `poslužitelj A` ih nema budući da on nema potrebe imati pristup tom servisu. Ako napadač slučajno dobije pristup servisu koji se nalazi na `poslužitelju B`, tada bi on mogao doći do osjetljivih podataka.
Napadaču su poznate sve interne IP adrese poslužitelja koji su dio te interne mreže. On tada šalje HTTP zahtjev s modificiranim URL-om s ciljem spajanja na servis putem ranjive aplikacije. `poslužitelj A` pokuša napraviti zahtjev prema servisu na `poslužitelju B`. Budući da nema potrebne vjerodajnice, zahtjev je neuspješan, kao i napad.



Slika 3.1: Neuspješno kontaktiranje servisa na unutarnjem poslužitelju

Kao što se vidi iz primjera, kako bismo smanjili mogućnost iskorištavanja SSRF ranjivosti, potrebno je onemogućiti pristup bez vjerodajnica poslužiteljima ili servisima kojima bi se moglo pristupiti iz interne mreže.

3.2 Različite specifikacije URI-a

Kod aplikacije koja uzima i koristi URL kao input korisnika bez provjere i filtracije mogu se javiti ranjivosti. Specifikacija koja definira što je URL i koja je njegova struktura se mijenjala tijekom godina. Sve je to opisano u RFC-ovima[1]. Zbog toga, čak i ako se postavi zaštita u obliku nekog filtra na input korisnika, pažljivim kreiranjem URL-a napadač može i

[1] *Request for Comments*, dokument koji opisuje teme vezane za Internet i računala

zaobići postavljene zaštite. Time se otvara put prema iskorištavanju SSRF ranjivosti budući da napadač pažljivim kreiranjem URL-a može zbuniti parsere koje se nalaze u aplikaciji i tako uspješno izvršiti napad.
U nastavku će biti detaljnije opisano što je to URL, te na koji način može doći do drugačijeg parsiranja među funkcijama koje bi zapravo trebale davati iste rezultate.
Bilo bi još dobro spomenuti kako je točnije reći URI. URL je zapravo manja klasifikacija URI-a. Zato, u nastavku ovog dijela će cijelo vrijeme biti korišten izraz URI.

URI

URI (engl *Uniform Resource Identifier*) pruža način na koji se definira neki resurs. To je samo identifikator koji se sastoji od niza znakova koji slijedi određena pravila. U nastavku je opisana specifikacija URI-a iz RFC-a 3986 [8].

Uniform - dopušteno je definiranje veće količine resursa u jednom kontekstu.

Resource - ne postoji limit nad time što može biti resurs. Dakle resurs može biti slika, dokument, servis ili nešto drugo.

Identifier - informacija koja pomaže razlikovati jedan resurs od svih ostalih resursa.

URI-i imaju globalni opseg (engl. *scope*) i interpretiraju se uvijek na jednak način, bez obzira na kontekst krajnjeg korisnika (engl. *end user context*), no operacija koja će se izvršiti nakon što se URI interpretira ovisi kontekstu krajnjeg korisnika. URI služi za identificiranje resursa, ali to ne znači da je postojanje resursa ili čak lokacija tog resursa garantirana URI-em.
URI se može još više klasificirati u manje klasifikacijske podvrste poput URL ili URN.
URL (engl. *Uniform Resource Locator*) je podvrsta URI-a koja osim što identificira resurs, pomaže u pronalasku lokacije tog resursa.
URN (engl. *Uniform Resource Name*) označava URI-e koji moraju ostati jedinstveni i perzistentni čak i ako resurs prestane biti dostupan ili prestane postojati.

Sintaksa URI-a je organizirana hijerarhijski. Svaki URI započinje **šemom** (engl. *scheme*). Postoji više šema, i svaka nameće svoja pravila.
Nakon šeme slijedi delimiter ":". Osim šeme, URI sadrži komponente koje su međusobno odvojene posebnim znakovima koji su označeni kao rezervirani. Komponente su redom: **autoritet** (engl. *authority*), **put** (engl. *path*), **upit** (engl. *query*), **fragment** (engl. *fragment*).
Slika 3.2 prikazuje općenitu strukturu URI-a. Inače, jedino što mora biti prisutno u URI-u

Slika 3.2: Struktura URI-a

jesu šema i put. Put može biti prazan. Sve ostale komponente su opcionalne, ali postoje definirana pravila kako mora izgledati URI ako nema neke komponente.

Autoritet započinje znakom "//", a završava znakom "/", znakom "#", znakom "?" ili jednostavno krajem URI-a. Kao što je već rečeno, autoritet ne mora biti prisutan u URI-u, ali se onda nameću posebna pravila za put, odnosno s kojim znakom mora započeti put.

Slika 3.3: Struktura autoritet komponente

Slika 3.3 prikazuje strukturu autoritet komponente. Autoritet se sastoji od **info korisnika** (engl. *user info*) koji ne mora biti prisutan, ali ako jest, onda on završava znakom "@", **domaćina** (engl. *host*) i **priključka** (engl. *port*) koji ne mora biti prisutan, ali ako jest, onda započinje znakom ":".

Put sadrži informacije. Ako unutar URI-a postoji autoritet, tada put započinje sa znakom "/" ili je prazan. Ako autoritet nije prisutan, tada put ne može započeti s "//". Put završava prvom pojavom znaka "?", znaka "#" ili krajem URI-a. Gramatika koja označava na koji način put može biti izgrađen ima više mogućih kombinacija.
Dodatno, postoji relativno referenciranje URI-a i relativna referenca. Relativna referenca označava resurs koji je dio neke strukture dokumenata.

Problem s parsiranjem

Kao što je rečeno, tijekom godina specifikacija URI-a se mijenjala. RFC 1738 [6], RFC 2396 [7] i najnoviji RFC 3986 daju specifikacije URI-a, no svaki na malo drugačiji način opisuje samu sintaksu, dovoljno različito da parseri koji parsiraju URI budu zbunjeni.
Razni programski jezici sadrže biblioteke koje su zadužene za mrežno komuniciranje. Kao dio biblioteke, napisane su funkcije koje su posebno zadužene za iščitavanje pojedinih dijelova URI-a. Napisani su parseri koji iz URI-a primjerice iščitavaju šemu ili put prema nekom resursu. Razni parseri slijede različite RFC specifikacije, a neki ih ni ne slijede.

Tu nastaje ozbiljan problem. Ako aplikacija u sebi sadrži neki zadatak koji uključuje rad sa URI-em, pojedine dijelove URI-a je potrebno parsirati. Ti dijelovi se parsiraju radi filtriranja, ili kako bi aplikacija dohvatila resurs s te putanje. Korištenje više različitih biblioteka može rezultirati drugačijim shvaćanjem URI-a budući da postoji mogućnost da te biblioteke slijede drugačije RFC specifikacije. Time može doći do neočekivanog ponašanja aplikacije i pojave nekih ranjivosti.

Zabuna parsera može biti zbog **šeme**, **znaka "/"**, **znaka "\"** i **URL enkodiranja**. Kako bi se objasnile sve vrste zabune, u nastavku slijede primjeri. U svakom primjeru je izabran neki jezik i biblioteke koje pokazuju drugačije ponašanje.

Zabuna šeme (engl. *Scheme confusion*)

Za potrebe primjera uzet je programski jezik Java i biblioteke *java.net.uri* [11] i *java.net.url* [12]. Za demonstraciju drugačijih rezultata parsera korišten je sljedeći URI: *example.com*.

```
1  public static void main(String[] args)throws Exception{
2
3      URI uri = new URI ("example.com");
4      try{
5          System.out.println("Host: " + uri.getHost());
6          System.out.println("Path: " + uri.getPath());
7      }catch(Exception e){
8          e.printStackTrace();
9      }
10
11     URL url = new URL("example.com");
12     try{
13         System.out.println("Host: " + url.getHost());
14         System.out.println("Path: " + url.getPath());
15     }catch(Exception f){
16         f.printStackTrace();
17     }
18 }
```

Specifikacijom URI-a prije RFC 3986 nije bilo naglašeno da šema **mora** biti sastavni dio URI-a. Tek u RFC-u 3986 je izričito rečeno kako je šema obavezni dio URI-a.
Korištenjem funkcije *getHost()* koja je dio biblioteke *java.net.uri* i iste funkcije, ali iz biblioteke *java.net.url* dobivaju se drugačiji rezultati.
Naime, funkcija iz prve biblioteke uspješno parsira dani URI, no *example.com* parsira kao

komponentu put, a ne kao komponentu autoritet.
S druge strane, korištenje iste funkcije, ali iz druge navedene biblioteke rezultira greškom. U nastavku je navedena greška.

```
1 Exception in thread "main" java.net.MalformedURLException: no
     protocol: example.com
2         at java.base/java.net.URL.<init>(URL.java:645)
3         at java.base/java.net.URL.<init>(URL.java:541)
4         at java.base/java.net.URL.<init>(URL.java:488)
5         at Java_example.main(java_example.java:20)
```

Greška *MalformedURLException* [10] označava kako parser nije mogao pronaći validan protokol unutar danog URI-a. Funkcija *getHost()* biblioteke *java.net.url* ne smatra dani URI validnim.

Zabuna znaka ”/” (engl. *Slash confusion*)

Za potrebe primjera, uzet je programski jezik Java i bibiloteka *java.net.url*, te softver *wget* [5].
Programski kod Java programa je identičan kao u prošlom primjeru. Softver *wget* je korišten pomoću komandne linije. Za demonstraciju drugačijih rezultata parsera korišten je sljedeći URI: *http:///example.com*.

Prema RFC-u 3986, nakon šeme slijedi znak ”:” i ”//” te nakon toga slijedi autoritet. Autoritet završava ili krajem URI-a ili nekim znakom koji je delimiter. Taj znak može biti i ”/”.

Parser koji slijedi RFC 3986 bi dani URI trebao shvatiti kao URI koji nema autoriteta, a za komponenta put bi uzeo */example.com*.
Funkcija *getPath()* biblioteke *java.net.url* slijedi RFC 3986 te prepoznaje točan string kao put, dok funkcija *getHost()* iste biblioteke ne isčitava autoritet, što je željeno ponašanje ako se slijedi RFC 3986.
S druge strane, rezultati korištenja softvera wget nad danim URI-em su drugačiji. Softver ne smatra dani URI validnim.
Iz gornjeg primjera se može vidjeti kako različiti parseri na drugačiji način parsiraju URI.

Zabuna znaka ”\\” (engl. *Backslash confusion*)

Za potrebe primjera uzet je programski jezik Javu i biblioteke *java.net.uri* i *java.net.url*. Za demonstraciju drugačijih rezultata parsera korišten je sljedeći URI: *http:\\example.com*.

Po RFC-u 3986 znakovi "/" i "\" su dva različita znaka. To znači da bi parseri koji slijede RFC trebali tretirati *http:\\example.com* i *http://example.com* kao dva različita URI-a.
Biblioteke koje su uzete za potrebe primjera doista rade razliku između ta dva znaka. Međutim, funkcije *getHost()* i *getPath()* iz obje biblioteke daju različite rezultate.
Funkcije iz biblioteke *java.net.uri* ne smatraju dani URI validnim. Parser nije u mogućnosti pravilo parsirati dani URI. U nastavku je prikazana greška koja se javlja nakon poziva funkcija.

```
1 Exception in thread "main" java.net.URISyntaxException: Illegal
     character in opaque part at index 5: http:\\example.com
2         at java.base/java.net.URI$Parser.fail(URI.java:2913)
3         at java.base/java.net.URI$Parser.checkChars(URI.java
             :3084)
4         at java.base/java.net.URI$Parser.parse(URI.java:3120)
5         at java.base/java.net.URI.<init>(URI.java:600)
6         at Java_example.confusion(java_example.java:21)
7         at Java_example.main(java_example.java:12)
```

Funkcije iz biblioteke *java.net.url* mogu parsirati dani URI, ali dio *\\example.com* je uzet kao komponenta put.
Kao što se vidi iz primjera, dvije različite funkcije na drugačiji način parsiraju dani URI.

Zabuna URL enkodiranjem (engl. *URL encoding confusion*)

Kao i u prethodnim primjerima, uzet je programski jezik Java i biblioteke *java.net.uri* i *java.net.url*. Za demonstraciju drugačijih rezultata parsera korišten je sljedeći URI: *http://%65%78%61%6d%70%6c%65%2e%63%6f%6d*.
URI predstavlja zapravo *http://example.com* u enkodiranom obliku. Naime po RFC-u 3986, svaki dio URI-a osim šeme može biti enkodiran. U nastavku je prikazano kako prethodne dvije biblioteke na drugačiji način parsiraju dani URI.
Funkcije *getHost()* i *getPath()* iz biblioteke *java.net.uri* smatraju URI validnim i parsiranje je uspješno, no *example.com* nije prepoznat kao komponenta domaćin kao što bi trebala biti.
S druge strane, funkcije pod istim imenom iz biblioteke *java.net.url* isto tako uspješno parsiraju URI. Ovaj put, niz kodova koji reprezentiraju *example.com* je prepoznat kao komponenta domaćin.

3.3 Stavljanje na crnu listu (engl. *Blacklisting*)

Jedan od načina zaštite protiv SSRF napada je korištenje crne liste. Programeri aplikacije koja kao input korisnika dobiva URL s kojim radi operacije stavljanjem određenih domena ili IP adresa na crnu listu mogu zabraniti napadaču njihovo kontaktiranje.
U svrhu primjera uzmimo za primjer dva poslužitelja, *poslužitelj A* i *poslužitelj B*. *poslužitelj A* je dostupan napadaču, tj. napadač ga može kontaktirati, dok *poslužitelj B* nije dostupan napadaču, ali *poslužitelj A* može kontaktirati *poslužitelj B*.

Neka je IP adresa *poslužitelja A 198.51.100.2*.
Neka je IP adresa *poslužitelja B 198.51.100.7*.

Programer će tada na crnu listu staviti upravo IP adresu *198.51.100.7*. Isto tako, u programski kod će uključiti filtar koji provjerava upisani input. Ako je input jednak elementima koji se nalaze na crnoj listi, on će izvršiti zadanu akciju koju je programer odredio te se traženi poslužitelj neće kontaktirati. U nastavku je dan programski kod koji pokazuje jedan od načina na koji se može isprogramirati aplikacija.

```
<?php
function filter($url){
        $blacklist = array("198.51.100.7", "http://198.51.100.7");
        $size = count($blacklist);
        for($i = 0; $i < $size; $i++){
                if(strcmp($url, $blacklist[$i]) == 0){
                        return false;
                }
        }
        return true;
}

if(isset($_GET['url'])){
    $url = $_GET['url'];
    if(filter($url)){
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
        $result_data = curl_exec($ch);
        curl_close($ch);
    }else{
        echo "Upisali ste URL koji nije dozvoljen";
    }
?>
```

Kao što se vidi u prethodnom kodu, za potrebe primjera niz pod nazivom *blacklist* igra ulogu crne liste. Nakon što korisnik pošalje URL, njegov sadržaj se uspoređuje sa svakim elementom crne liste. Ako je barem jedan element jednak poslanom URL-u, kontaktiranje poslužitelja se neće izvršiti, već će aplikacija javiti grešku.
U ovom primjeru se poslani URL uspoređuje sa čistim nizom znakova. Općenito, filtar može biti i u obliku regularnog izraza koji provjerava je li poslani URL možda nedopuštenog oblika.
Ova metoda zaštite nije najbolja budući da se vrlo lako može zaobići. U prethodnom dijelu ovom poglavlja bila je opisana struktura URL-a te je isto tako bilo pokazano kako se pojedini dijelovi URL-a mogu i enkodirati. Upravo će to biti jedna od metoda zaobilaska ovog načina zaštite.

Potrebno je napomenuti da se ovaj način zaštite može zaobići i preusmjeravanjem, ali u ovom trenutku to neće biti objašnjeno. Preusmjeravanje je isto tako metoda koja se koristi kako bi se zaobišla druga vrsta zaštite, stoga će ono kasnije biti objašnjeno.

Zaobilaženje crne liste enkodiranjem

U ranijim dijelu poglavlja rečeno je bilo kako se određeni dijelovi URL-a mogu enkodirati. Postoji više enkodiranja koja se mogu iskoristiti za zaobilaženje zaštite. Isto tako, osim korištenja jedne vrste enkodiranja, oni se mogu i kombinirati.

Heksadekadsko enkodiranje

Heksadekadskim enkodiranjem neki niz znakova se predstavlja brojevima zapisanim u bazi 16. Za enkodiranje IP adrese, dovoljno je prebaciti decimalne brojeve u heksadekadsku bazu, a između pretvorenih brojeva ostaviti točku. Za enkodiranje niza znakova poput imena neke domene može se i koristiti ASCII tablica za pretvaranje niza znakova u niz heksadekadskih brojeva.
Za zaobilaženje zaštite koja je definirana u primjeru iznad, potrebno je enkodirati IP adresu *198.51.100.7* u heksadekadski oblik. Na taj način, kod uspoređivanja enkodiranog URL-a i elemenata crne liste, filtar neće primijetiti kako oni predstavljaju zapravo istu IP adresu, no funkcija koja obavlja kontaktiranje IP adrese će biti u mogućnosti izvršiti svoju zadaću. Napadač koji na ovaj način enkodira poslan URL će uspješno zaobići postavljenu zaštitu.

IP adresa *198.51.100.7* nakon heksadekadskog enkodiranja izgleda ovako:
0xc6.0x33.0x64.0x7

Oktalno enkodiranje

Na isti način, oktalnim enkodiranjem niz znakova se može predstaviti brojevima zapisanim u bazi 8.
Za zaobilaženje zaštite definirane u primjeru iznad, potrebno je enkodirati IP adresu *198.51.100.7* u oktalni oblik. Kao što je bilo i u slučaju heksadekadskog enkodiranja, napadač koji pošalje URL koji je u oktalno enkodiran će izvršiti uspješan SSRF napad budući da filtar neće primijetiti kako poslani enkodirani URL i elementi na crnoj listi pokazuju na istu IP adresu.

IP adresa *198.51.100.7* nakon oktalnog enkodiranja izgleda ovako: *0306.063.0144.07*

Dword enkodiranje

IP adresa je zapravo 32-bitni broj koji je podijeljen na 4 okteta i svaki oktet je odvojen točkom. Kad bi se ta IP adresa napisala kao niz bitova, te kad bi se taj broj pretvorio u broj u bazi 10, dobio bi se dword, ili *double word*. Dword enkodiranje je pretvaranje IP adrese u odgovarajući dword. Napadač koji pretvori IP adresu koju želi poslati u dword će izvršiti uspješan SSRF napad nad aplikacijom budući da, kao i u prethodnim primjerima, filtar neće primijetiti kako poslani URL označuje IP adresu koja se nalazi na crnoj listi.

IP adresa *198.51.100.7* nakon dword enkodiranja izgleda ovako: *3325256711*

3.4 Stavljanje na bijelu listu (engl. *Whitelisting*)

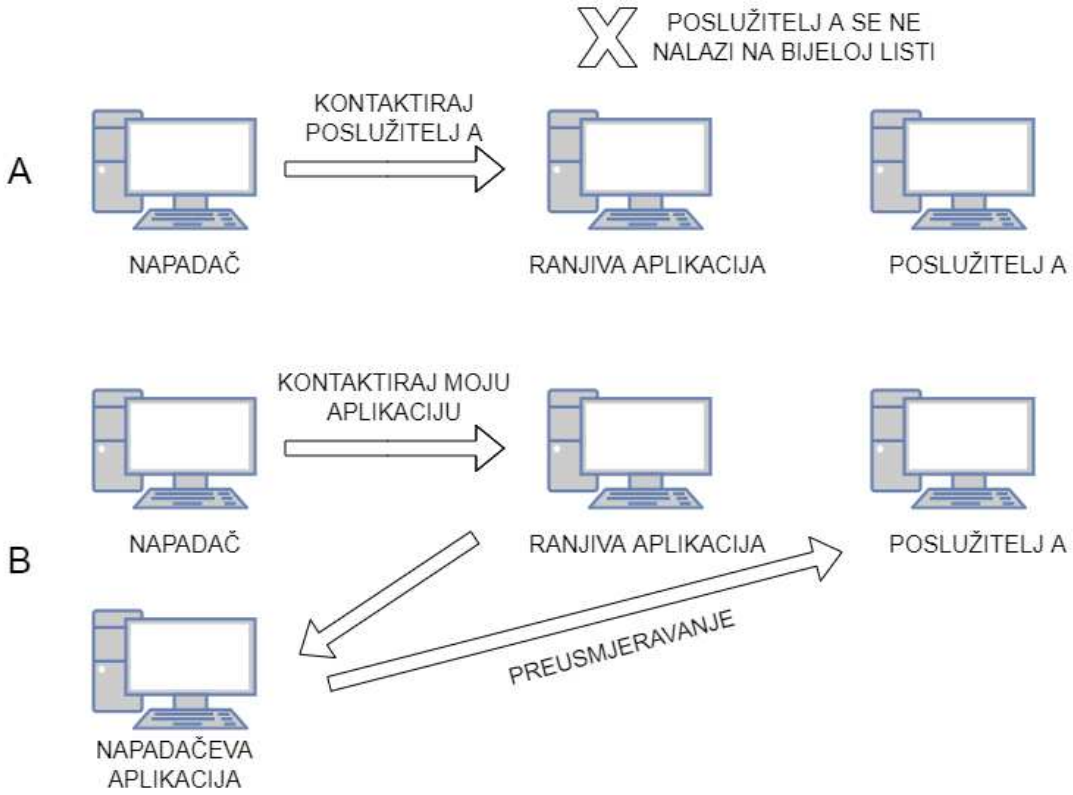
Malo bolja zaštita protiv SSRF napada od stavljanja na crnu listu jest stavljanje na bijelu listu. Na bijelu listu su stavljene domene, IP adrese prema kojima je dopušteno slati zahtjeve. Nakon što korisnik pošalje URL, aplikacija uspoređuje poslani URL s elementima bijele liste. Ako je poslani URL dopušten, tada se s njim nastavlja obavljati operacija koja je programirana u aplikaciji.
Za demonstraciju što to točno znači uzmimo tri poslužitelja, *poslužitelj A*, *poslužitelj B* i *poslužitelj C*. Svaki od tri poslužitelja poslužuje neku aplikaciju. Korisniku izvana je dostupan *poslužitelj A*. On može pristupiti aplikaciji na tom poslužitelju i tamo slati proizvoljne URL-ove kako bi kontaktirao druge poslužitelje. *poslužitelj B* je na bijeloj listi, dok *poslužitelj C* nije.
Ako korisnik pošalje URL koji pokazuje na aplikaciju na *poslužitelju B*, tada će aplikacija napraviti zahtjev prema toj aplikaciji budući da se nalazi na bijeloj listi. Ako korisnik pošalje URL koji pokazuje na aplikaciju koja se poslužuje na *poslužitelju C*, budući da on

nije na bijeloj listi, zahtjev neće biti napravljen.

Napadač koji želi pristupiti lokacijama koje su zaštićene na ovaj način će prvo poslati URL koji pokazuje na te lokacije. Naravno, one će mu biti nedostupne. Ali postoji način na koji se može zaobići i ova mjera zaštite.

Zaobilaženje bijele liste preusmjeravanjem

Napadač može iskoristiti mogućnosti aplikacije da slijedi preusmjeravanje. Proučavanjem aplikacije, kreiranjem i slanjem raznih URL-a i promatranjem odgovora, napadač može naslutiti neke od elemenata bijele liste. Ako aplikacija slijedi preusmjeravanje, napadač može iskoristiti ranjivost otvorenog preusmjeravanja (engl. *Open Redirect Vulnerability*). Napadač može ”natjerati” aplikaciju da napravi zahtjev prema njegovoj aplikaciji koja će ga preusmjeriti na IP adresu koju napadač želi napasti, ali ne može doći do nje zbog korištenja bijele liste.
Slika 3.4 prikazuje jedan takav slučaj. U A dijelu, napadač pokuša poslati poslati ranjivoj aplikaciji URL koji pokazuje prema *poslužitelju A*. Budući da *poslužitelj A* ne bi trebao biti dostupan napadaču, on je zaštićen korištenjem bijele liste. Ranjiva aplikacije jednostavno neće napraviti zahtjev prema *poslužitelju A*.
S druge strane, u B dijelu, napadač poslužuje svoju aplikaciju čiji je jedini zadatak preusmjeravati posjetitelje na IP adresu *poslužitelja A*. Tada, napadač pošalje ranjivoj aplikaciji zahtjev u kojoj joj nalaže kontaktiranje njegove aplikacije. Budući da aplikacija ima ranjivost otvorenog preusmjeravanja, ona će posjetiti napadačevu aplikaciju. No time će biti preusmjerena na *poslužitelj A*, te će napadač uspješno izvršiti svoj napad.



Slika 3.4: Zaobilaženje bijele liste

Poglavlje 4

SSRF scenariji: aplikacijski kod

Za kraj, svi primjeri prikazani u ovom diplomskom radu su spojeni u jednu aplikaciju. Aplikacija je zamišljena kao skupina manjih ranjivih aplikacija kojima se pristupa preko jedne glavne stranice. Aplikacija je *web* aplikacija, te joj se može pristupiti, jednom kad je pokrenuta, preko `http://localhost`.

4.1 Aplikacija

Aplikacija se sastoji od `Glavne aplikacije` i skupine manjih aplikacija. Te manje aplikacije su zapravo prethodno opisani primjeri. Iz glavne aplikacije se može pristupiti bilo kojem primjeru, te se iz svakog primjera može vratiti u glavnu aplikaciju. Za svaki primjer se u aplikaciji koristi naziv *zadatak*, ili *scenarij*. Razlog zašto se za svaki primjer u nekim dijelovima koristi riječ "zadatak" je taj što se aplikacija isto tako može shvatiti kao edukacijska aplikacija kojoj je cilj pokazati na koje se načine može iskoristiti SSRF ranjivost. Osim što će ovdje biti pobliže opisana struktura, zapravo nije važno napraviti razliku između `Glavne aplikacije` i primjera. Cijela aplikacija se može shvatiti kao jedna cjelina.

Na početnoj stranici je opisana SSRF ranjivost, kao i svrha aplikacije. Kako bi se pristupilo glavnom dijelu aplikacije, potrebno je pritisnuti na karticu `Zadaci`. Kao što se vidi na slici 4.1, prethodno opisanim primjerima, ili zadacima se može pristupiti odabirom željenog scenarija. Nakon toga, korisnik će biti prebačen na posebnu stranicu na kojoj se pobliže objašnjava taj scenarij. Na toj stranici se nalazi i rješenje tog zadatka.

Slika 4.1: Glavni dio aplikacije

Uzmimo za primjer prvi scenarij `Dohvaćanje lokalnih resursa` . Na slici 4.2 se može vidjeti sav prethodno opisani sadržaj.
Kao što se i vidi iz slike, pritiskom na gumb korisnik će biti preusmjeren na zadatak, u ovom slučaju prvi primjer. Svi primjeri su bili detaljno objašnjeni u prethodnim poglavljima, stoga će u ovom dijelu svaki primjer biti ukratko objašnjen. Na par primjera su napravljene neke preinake. U tom slučaju, primjer i preinake će biti detaljno objašnjene.

Slika 4.2: Primjer stranice koja vodi do primjera

Prvi scenarij: Dohvaćanje privatnih resursa

Ranjiva aplikacija se poslužuje na poslužitelju koji sadrži osjetljive datoteke. Datoteke sadrže admin vjerodajnice. Napadač može iskorištavanjem SSRF ranjivosti doći do tih vjerodajnica.
Kao što je i prethodno objašnjeno u prvom poglavlju, korišten je `Docker` kontejner u kojem je pokrenuta aplikacija. Isto tako, napravljena je osjetljiva datoteka te je prebačena u kontejner na posebno mjesto.

Ovdje nisu napravljene nikakve preinake.

Drugi scenarij: Kontaktiranje unutarnjih poslužitelja

Napravljene su dvije aplikacije. Jedna se poslužuje na poslužitelju koji jest dostupan izvana. Ta aplikacija je ranjiva aplikacija. Druga se poslužuje na poslužitelju koji nije dostupan izvana. Ta aplikacija igra ulogu admin sučelja. Poslužitelj koji nije dostupan izvana jest dostupan poslužitelju koji je dostupan izvana. Iako napadač ne može pristupiti aplikaciji koja se poslužuje na poslužitelju koji nije dostupan izvana, on može natjerati aplikaciju

da kontaktira ”admin” sučelje. Na taj način, napadač može saznati sadržaj admin sučelja. Kao što je i prethodno objašnjeno u prvom poglavlju, korišteni su `Docker` kontejneri u kojima su pokrenute obe aplikacije.

Ovdje nisu napravljene nikakve preinake.

Treći scenarij: Blacklisting

Napravljene su dvije aplikacije. Jedna se poslužuje na poslužitelju koji jest dostupan izvana. Ta aplikacija je ranjiva aplikacija. Druga se poslužuje na poslužitelju koji nije dostupan izvana. Ta aplikacija igra ulogu admin sučelja. Poslužitelj koji nije dostupan izvana jest dostupan poslužitelju koji je dostupan izvana. Napadač ne može poslati URL bez dodatnih modifikacija budući da je u aplikaciji isprogramiran i filtar koji odbacuje URL-ove koji u sebi sadrže riječi prethodno stavljene na crnu listu. Napadač modificiranjem URL-a može natjerati aplikaciju da kontaktira admin sučelje. Na taj način, napadač može saznati sadržaj admin sučelja.
Kao što je i prethodno objašnjeno u trećem poglavlju, korišteni su `Docker` kontejneri u kojima su pokrenute obe aplikacije.

Ovdje nisu napravljene nikakve preinake.

Četvrti scenarij: Izvršavanje funkcija namjenjenih samo adminu

Napravljene su dvije aplikacije, te jedna baza podataka. Jedna se poslužuje na poslužitelju koji jest dostupan izvana. Ta aplikacija je ranjiva aplikacija. Druga se poslužuje na poslužitelju koji nije dostupan izvana. Ta aplikacija igra ulogu admin sučelja. Poslužitelj koji nije dostupan izvana jest dostupan poslužitelju koji je dostupan izvana. Admin sučelje ima pristup bazi podataka u kojoj su upisani korisnici. Isto tako, na admin sučelju se može vidjeti popis korisnika, kao i gumb kojim se briše jedan korisnik. Napadač ne može pristupiti bazi podataka i obrisati korisnika, ali može natjerati admin sučelje da to napravi.

Ovaj primjer je pobliže objašnjen u drugom poglavlju, no tada je baza podataka napravljena na vrlo jednostavan način: korištena je tekstualna datoteka koja je igrala ulogu baze podataka. U ovoj aplikaciji je lažna baza podataka zamijenjena pravom bazom podataka. Naime, iskorišten je jedan dodatan `Docker` kontejner te je u njemu pokrenuta baza podataka. Njoj je moguće pristupiti preko određene IP adrese. Naravno, za pristup je potrebno imati i određene vjerodajnice kao što je *root* lozinka.

Peti scenarij: Skeniranje interne mreže

Napravljena je jedna aplikacija. Koristeći ju, korisnik može učitati bilo koju sliku s poslanog URL-a. Sadržaj poslanog URL-a se spremi na poslužitelja s `png` ekstenzijom, te se ta datoteka prikaže natrag korisniku unutar `image html` taga. Ako aplikacija ne može napraviti zahtjev prema poslanom URL iz bilo kojeg razloga, javlja se greška. Na taj način, korisnik (u ovom slučaju napadač) može skenirati internu mrežu.
Kao što je i prethodno objašnjeno u drugom poglavlju, korišten je `Docker` kontejner u kojemu je pokrenuta aplikacija. U ovom slučaju, pokrenuta je samo jedna aplikacija, budući da se ona već nalazi u mreži sa ostalim primjerima, te oni zapravo u ovom slučaju igraju ulogu ”drugog” poslužitelja.

Ovdje nisu napravljene nikakve preinake, osim što je pokrenuta samo jedna aplikacija (ona bitna/glavna).

4.2 Struktura

Aplikacija je napravljena koristeći `Docker`. Svaki primjer je dobio svoj kontejner. Isto tako, `Glavna aplikacija` se nalazi u posebnom kontejneru. Svi kontejneri su zapravo dio jedne mreže. Kako bi se sve to jednostavno postavilo, korišten je `docker-compose`. U nastavku je dan kôd pomoću kojeg su postavljeni svi kontejneri.

Potrebno je napomenuti kako je ovo zapravo skraćeni kôd, te su neki dijelovi izbačeni. Ti dijelovi su vrlo slični ostalim koji su prikazani, te nije ih bilo potrebe ostaviti. U kôdu se primjerice može vidjeti kako je glavna stranica, tj. sama aplikacija dostupna na `http://localhost`, a baza podataka je dostupna na IP adresi `198.51.100.100`.

```
version: '3.9'
services:
  glavna_stranica:
    build: glavna_stranica/
    ports:
      - "80:80"
    networks:
      network-glavni:
        ipv4_address: 198.51.100.2
    ...
  zadatak_2_server_1:
    build: zadatak_2/server1/
    ports:
```

```
      - "7002:80"
    networks:
      network-glavni:
        ipv4_address: 198.51.100.4

  zadatak_2_server_2:
    build: zadatak_2/server2/
    ports:
      - "80"
    networks:
      network-glavni:
        ipv4_address: 198.51.100.5
  ...
  db:
    build: zadatak_4/database/
    ports:
      - "3306:3306"
    networks:
      network-glavni:
        ipv4_address: 198.51.100.100

networks:
  network-glavni:
    ipam:
      config:
        - subnet: 198.51.100.0/24
```

Radi lakšeg i bržeg pokretanja aplikacije, napravljena je posebna skripta `run.sh`. U nastavku je dan kod skripte. Kontejneri se pokreću u *detach* stanju, kao što je vidljivo iz druge linije koda.

```
sudo docker-compose build
sudo docker-compose up --detach
```

Isto tako, za lakše zaustavljanje cijele aplikacije, tj. svih kontejnera, napravljena je posebna skripta `stop.sh`. U nastavku je dan kod skripte.

```
sudo docker-compose down --remove-orphans
```

Bibliografija

[1] Amazon, *Start Building on AWS Today*, https://aws.amazon.com/, pristupljeno 15.8.2022.

[2] Diverto, *Stanje informacijske sigurnosti u Republici Hrvatskoj 2021.*, 2021, https://www.diverto.hr/documents/diverto_stanje_informacijske_sigurnosti_2021.pdf, pristupljeno 10.8.2022.

[3] Docker, *Docker Documentation*, https://docs.docker.com/, pristupljeno 17.8.2022.

[4] _____, *Overview of Docker Compose*, https://docs.docker.com/compose/, pristupljeno 5.8.2022.

[5] Free Software Foundation, *GNU Wget*, https://www.gnu.org/software/wget/?, pristupljeno 5.8.2022.

[6] IETF, *Uniform Resource Locators (URL)*, 1994, https://www.rfc-editor.org/rfc/rfc1738, pristupljeno 10.8.2022.

[7] _____, *Uniform Resource Identifiers (URI): Generic Syntax*, 1998, https://www.rfc-editor.org/rfc/rfc2396, pristupljeno 10.8.2022.

[8] _____, *Uniform Resource Identifier (URI): Generic Syntax*, 2005, https://www.rfc-editor.org/rfc/rfc3986.html, pristupljeno 10.8.2022.

[9] _____, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, 2014, https://www.rfc-editor.org/rfc/rfc7231, pristupljeno 1.8.2022.

[10] Oracle, *Class MalformedURLException*, https://docs.oracle.com/javase/7/docs/api/java/net/MalformedURLException.html, pristupljeno 11.8.2022.

[11] _____, *Class URI*, https://docs.oracle.com/javase/7/docs/api/java/net/URI.html, pristupljeno 11.8.2022.

[12] _____, *Class URL*, https://docs.oracle.com/javase/7/docs/api/java/net/URL.html, pristupljeno 11.8.2022.

[13] OWASP, *OWASP Top Ten*, 2021, https://owasp.org/www-project-top-ten/, pristupljeno 5.8.2022.

[14] PortSwigger, *Server-side request forgery (SSRF)*, https://portswigger.net/web-security/ssrf, pristupljeno 5.8.2022.

[15] Riyaz Walikar, *An SSRF, privileged AWS keys and the Capital One breach*, 2019, https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af, pristupljeno 23.8.2022.

Sažetak

SSRF je kibernetička ranjivost gdje zlonamjerni korisnik uspije ”natjerati” ranjivu aplikaciju ili poslužitelj da napravi proizvoljan zahtjev prema nekoj proizvoljnoj lokaciji. Sama SSRF ranjivost ne mora biti opasna, ali njezinim iskorištavanjem se mogu potencijalno iskoristiti i neke druge ranjivosti koje jesu opasne.
Iskorištavanjem SSRF ranjivosti se može doći do osjetljivih podataka, kontaktirati poslužitelje ili usluge unutar interne mreže koji nisu dostupni izvana, ili se nakon nje mogu iskoristiti i neke druge ranjivosti, te je moguće čak dobiti pristup poslužitelju i izvršavati kod iz daljine (RCE).

Postoji više načina na koji se može aplikacija zaštititi od SSRF-a. Pažljivim programiranjem i korištenjem recimo kombinacije bijele liste i crne liste, te korištenjem vatrozida, može se smanjiti mogućnost da je aplikacija ranjiva na SSRF ranjivost.

OWASP je 2021. godine uvrstio SSRF na svoj popis *Top Ten Web Application Security Risks*. Kao što i sam naziv govori, popis sadrži ranjivosti koje su bile najkritičnije te godine po mišljenjima stručnjaka.

Summary

SSRF is a cybersecurity vulnerability where malicous user forces an application or server to make a request to arbitrary location. SSRF vulnerability is not dangerous by itself, but chained with other vulnerabilites it can create very dangerous exploit-chain that can lead to exploatation of the application.
Abusing SSRF vulnerability, an attacker can extract sensitive information (like passwords), contact servers or services that are not available from the outside, or, SSRF vulnerability can be used to perform other very dangerous attacks, such as Remote Code Execution (RCE).

There are many ways to combat this vulnerability. Using good programming practices and maybe combining whitelist, blacklist and firewall, application can be made more safe from this kind of vulnerability at least.

OWASP added SSRF vulnerability to their top ten list named *Top Ten Web Application Security Risks*. As it can be deducted from the name, the list consists of the most critical vulnerability that year based on consensus of experts.

Životopis

Lucija Valentić je rođena 7.2.1997. u Sisku. Pohađala je Gimnaziju Sisak, te je 2015. godine upisala Preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu Matematika (PMF Matematika) u Zagrebu. 2018. godine završava Preddiplomski sveučilišni studij Matematika, te upisuje diplomski studij Računarstvo i matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. 2021. godine se zapošljava na Fakultetu elektrotehnike i računarstva (FER) u Zagrebu kao junior inženjer za kibernetičku sigurnost. U sklopu posla provodi penetracijsko testiranje i pripremu CTF (engl. *Capture the Flag*) natjecanja i sigurnosnih dokumenata u suradnji sa Nacionalnim CERT-om.