VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IZRADA SUSTAVA POTPORE PENETRACIJSKOM TESTIRANJU

Karlo Kegljević

Zagreb, rujan 2020.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu,15.9.2020.

Predgovor

Ovim putem se želim zahvaliti profesoru Robertu Petruniću na ukazanom povjerenju pri izradi završnog rada. Profesor Petrunić je dao sve od sebe kako bi što prije pogledao novu verziju rada te bi mi svaki puta napisao detaljne upute kako i što ispraviti. Hvala mu što je usmjerio moje lude ideje završnog u jedan konkretan i koristan rad.

Hvala Danijelu Beleu za njegovo maksimalno ulaganje u predavanje svakog predmeta kojeg sam slušao kod njega. Programski jezici i predmeti ne bi bili zanimljivi da nije bilo njegovih šala i stručnog znanja. Ako me pitaju po čemu ću se sjećati fakulteta, odmah ću pomisliti na Milicu Krmpotić i Danijela Belea. Profesore hvala!

Hvala kolegi i prijatelju Tomislavu Kućaru za pokazivanje knjiga i projekata koji su potrebni kako bih ostvario neke od svojih ciljeva u životu. Bez Tomislava ne bih imao većinu znanja koje sam stekao u učenju kod kuće.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Kako bih olakšali proces svim penetracijskim testerima, izrađen je sustav koji spaja više različitih alata u jedan. Nazvali smo ga Dante. Njegova svrha je traženje otvorenih portova na sustavima, traženje ranjivosti za servise koji rade na tim portovima, te generiranju izvješća koje će uvelike olakšati proces pisanja sigurnosnog izvješća. Također su izrađena i 2 pomoćna alata koje Dante koristi. Jedan od njih je skener za direktorije na internet stranicama. U izradi skenera za direktorije fokusirali smo se na brzinu i fleksibilnost. Može se koristiti kao zasebni modul a i kao biblioteka. Njega smo nazvali dr.buster. Osim dr.bustera, izrađena je i omotnica oko *exploit-db* stranice. Posebnost kod toga je da je taj modul napravljen u jednoj datoteci, tako da smo ga lagano uvezli i koristili u Danteu te je poseban u formatiranju podataka za generiranje izvješća što je i krajnji rezultat Dantea.

Ključne riječi: Sigurnosno testiranje, sigurnosni alati, sigurnosno izvješće, Dante

To facilitate the penetration testing process for all penetration testers, we created a system that combines multiple tools into one. We called it Dante. The aim of this system is to scan for open ports on the systems, identify the exploits of the daemons working on these ports and to generate a report that would greatly expedite the process of writing a security report. Additionally, we created 2 auxiliary tools that Dante uses. One of these tools is a web path finder. While coding a web path finder, the focus was put on and flexibility. It is functional as a standalone script or as a library. we called him dr.buster. Besides dr.buster, we also created an exploit-db wrapper. The special feature of this wrapper is that it is created in a single file, meaning that we easily imported it and used it with Dante. The wrapper also has special data formatting suitable for generating reports, which is the end result of Dante.

Keywords: Security testing, Security tools, Security report, Dante

Sadržaj

1. Uvod

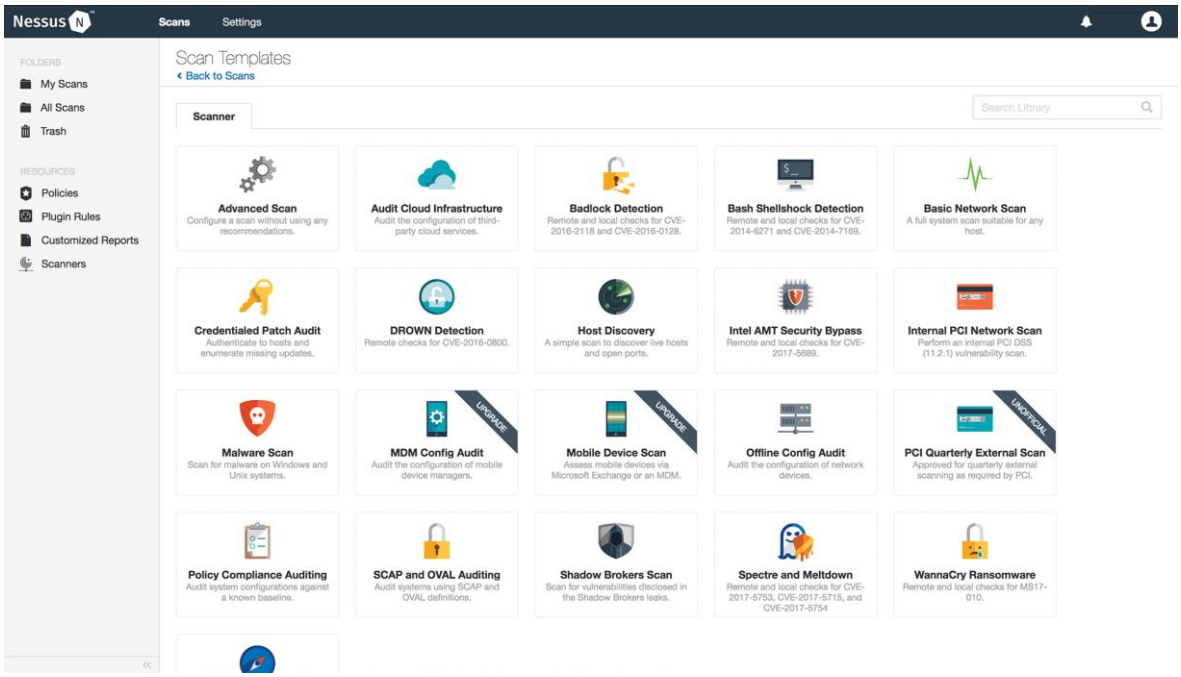
Prilikom sigurnosnih testiranja sustava koristimo mnoge alate. Korištenje svakog alata pojedinačno uzima vremena, a vrijeme je presudno u sigurnosnim testiranjima. Već postoje neki alati tipa *Nessus*[1] koji spajaju više alata u jedan. No oni imaju puno različitih opcija, parametara itd. Sustav koji ćemo izraditi bit će intuitivan i lagan za korištenje penetracijskom testeru, a davat će potrebnu količinu informacija kako bi penetracijskom testeru omogućio sažeto i korisno izvješće. On tada može koristiti to izvješće na više načina. Na primjer, može ga koristiti kao kostur izrade krajnjeg sigurnosnog izvješća. Također u izvješću (ako ranjivosti postoje) bit će linkovi na stranice gdje će se moći preuzeti kôd koji će iskorištavati te ranjivosti. Ovaj sustav bit će otvorenog kôda, što znači da će se moći nadograđivati po potrebi korisnika. Otvorenog kôda bit će i svi moduli koje sustav koristi što znači da krajnji korisnik ima uvid u kompletan kôd sustava, te može prilagođavati što god poželi.

[1] https://www.tenable.com/products/nessus

2. Usporedba sličnih alata

2.1. Usporedba *Dantea* s *Nessusom*

Kada bi pogledali korisničko sučelje nekih sličnih alata primjerice Nessus vidjeli bi kako je ono opširno.



Slika 2.1 Prikaz korisničkog sučelja *Nessus*-a

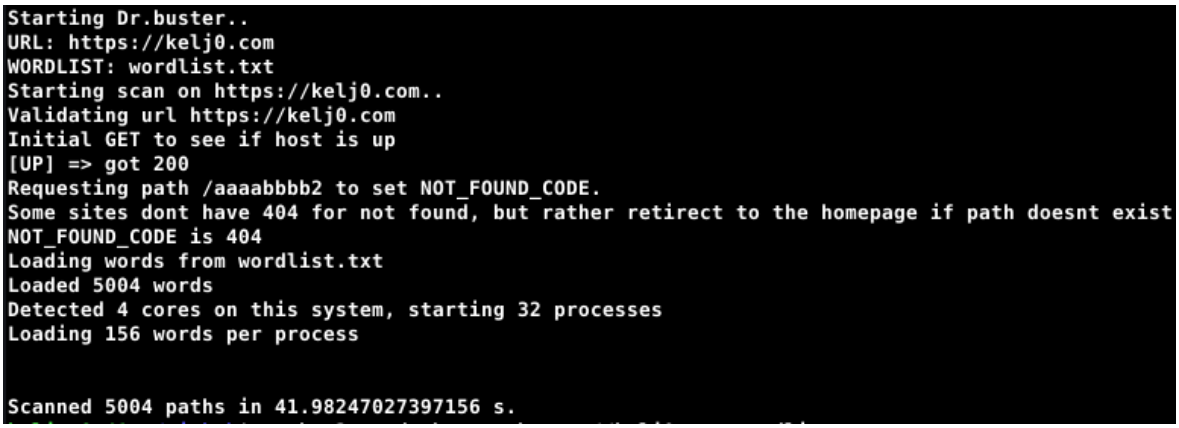
Iz Slika 2.1 Prikaz korisničkog sučelja *Nessus*-aSlika 2.1 vidimo kako *Nessus* nudi puno opcija, no po pitanju brzine pokretanja i stvarnog početka procesa testiranja ovakvom alatu potrebno je više vremena. Osim toga, potrebno je poznavati to sučelje što iziskuje dodatan trud eksperta. Dante se može pokrenuti odmah nakon preuzimanja i instalacije potrebnih paketa te je za njegovo pokretanje potrebna samo adresa računala koje se testira.

```
keljo@0x41:~/github/dante(master)$ python3 dante.py -h
usage: dante.py [-h] ip

positional arguments:
  ip          IP address of server you want to test

optional arguments:
  -h, --help  show this help message and exit
```

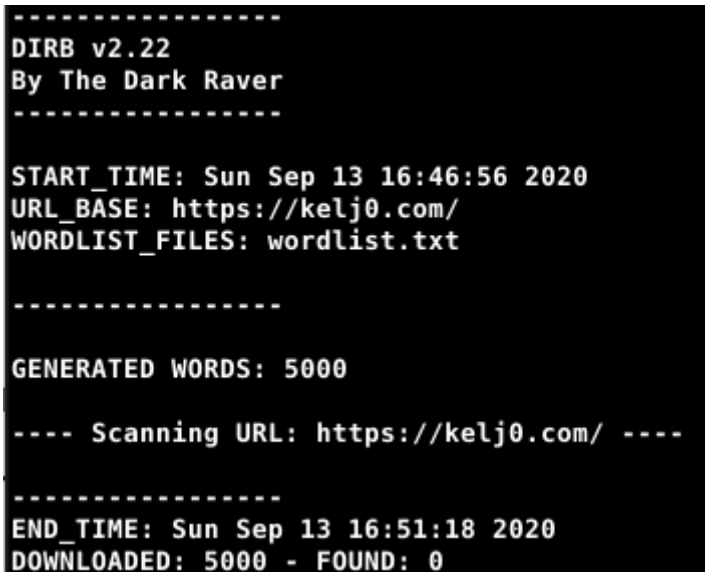
Slika 2.2 Prikaz informacijske poruke Dantea

2.2. Usporedba *dr.bustera* s *dirb*

Kada uspoređujemo dva skenera za pretragu direktorija na web aplikacijama, najbitnija nam je stvar brzina. U ovoj usporedbi pokretali smo *dirb*[2] i dr.buster s istom listom riječi, na istom računalu. Rezultati su sljedeći:



Slika 2.3 Prikaz dr.buster skena adrese https://kelj0.com



Slika 2.4 Prikaz *dirb* skena adrese https://kelj0.com

Iz prikazanih slika vidimo kako je dr.busteru bila potrebna 41 sekunda kako bi napravio sken adrese. Koristeći istu listu riječi, isto računalo te istu adresu *dirbov* sken trajao je preko 4 minute (točnije 262 sekunde). Što znači da se dr.buster pokazao 6.39 puta efikasnijim.

[2] https://tools.kali.org/web-applications/dirb

3. Osnovna terminologija vezana uz izradu sigurnosnog sustava za potporu sigurnosnom testiranju

3.1. Penetracijsko testiranje

Penetracijsko testiranje je proces provjere sigurnosti informacijskog sustava u kojoj ispitivač (etički haker engl. *white hat*[3] *hacker*) nakon potpisivanja ugovora o neotkrivanju podataka (engl. *non disclosure agreement*) glumi napadača (zlonamjernog hakera engl. *black hat*[4] *hacker*) kako bi pronašao ranjivosti u sustavu. Nakon obavljene provjere, etički haker piše izvješće koje se dostavlja vlasniku sustava koji tada (ukoliko hoće) poduzima potrebne mjere nadogradnje sustava.

Važna stvar prije samog testiranja je definiranje opsega (engl. *scope*) i cilja (engl. *objective*)testiranja. To se definira na više načina ovisno što se testira. Na primjer ukoliko se testiraju uređaji u nekoj unutarnjoj mreži opseg testiranja može biti definiran na sljedeći način:

- Strojevi za testiranje nalaze se u 192.162.1.0/28
- u opseg također ulazi Domenski sustav imena - DNS[5] (engl. *Domain Name System*) na adresi 192.162.1.156

Tek nakon definiranja opsega i cilja, te potpisivanja ugovora i NDA[6] dokumenta, ispitivač može početi s testiranjem sustava.

[3] https://en.wikipedia.org/wiki/White_hat_(computer_security)
[4] https://en.wikipedia.org/wiki/Black_hat_(computer_security)
[5] https://en.wikipedia.org/wiki/Domain_Name_System
[6] https://en.wikipedia.org/wiki/Non-disclosure_agreement

Slika 3.1 Pojednostavljeni prikaz procesa penetracijskog testiranja

3.2. Ranjivosti

Ranjivost je programski propust koji omogućuje korisniku korištenje usluge na nepredviđen način. Zlonamjerni napadač može korištenjem ranjivosti narušiti sigurnost nekog informacijskog sustava. Najčešće ranjivosti se događaju radi ljudske nepažnje. Ranjivosti mogu biti fizičke, programske, a i psihološke (socijalni inženjering, engl. *social engineering*[7]). Danas najpoznatiji sustav koji omogućuje uvid u javno poznate ranjivosti je CVE[8] (engl. *Commong Vulnerabilities and Exposures*).

Primjer ranjivosti u *Python* kôdu:

```
c.execute("SELECT * FROM tablica WHERE stupac = '%s'" %
varijabla)
```

U ovom kôdu moguć je *SQL Injection*[9]. Ukoliko varijabla postane „test OR 1=1--“ izlaz *SELECT-a* bit će svi redovi ove tablice.

```
c.execute('SELECT * FROM tablica WHERE stupac=?', (varijabla,
))
```

Na ovaj način smo zaštitili kôd od *SQL Injectiona*. *Python* će u navedenom slučaju pažljivo raditi *SELECT* komandu tako da pronađe primjerice navodnike, i pripremi ih na poseban način koji neće slomiti upit na bazu.

[7] https://en.wikipedia.org/wiki/Social_engineering_(security)
[8] https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures
[9] https://en.wikipedia.org/wiki/SQL_injection

3.2.1. Zero-day (0-day)

Zero-day[10], poznat kao 0-day je poseban tip ranjivosti za koji je specifično da još nije poznat proizvođaču. Taj tip ranjivosti je najopasniji jer za njega nema zakrpi. Srećom takvi tipovi su rijetki i ne pronalaze se tako lagano. Ime Zero-day je originalno bilo korišteno za dane od kako je novi program izašao u javnost, znači zero-day program je program koji je bio preuzet od strane zlonamjernih hakera neovlaštenim provaljivanjem u servere gdje se nalazio kôd programa prije njegova originalnog datuma objavljivanja. Danas se taj termin koristi i kod ranjivosti, i odnosi se na broj dana koje je proizvođač programa imao da popravi tu ranjivost.

Među najpoznatijim virusima koji koriste 0-day ranjivosti je *Stuxnet,* virus pretpostavljeno napravljen od strane Amerike i Izraela kako bi napravili veću štetu Iranskom nuklearnom programu[11]. *Stuxnet* je koristio čak 4 0-day ranjivosti, što bi značilo da je on najkompleksniji virus napravljen do tada[12].

Obrane od 0-day ranjivosti u periodu od kad se sazna do perioda izdavanja zakrpi postoje. Potrebno je stručno znanje računalnih eksperta da pronađu privremene načine smanjenja rizika. Primjerice ako pogledamo *Shellshock[13]* ranjivost, *Red Hat* je izdao posebna pravila u vatrozid-u kojima se sprječava iskorištavanje te ranjivosti[14].

3.3. Skeneri

Općenito pojam „skener“ u digitalnom svijetu predstavlja neki program čija je svrha automatiziranim načinom doći do potencijalno važnih informacija. Danas se najčešće skeneri programiraju u skriptnim jezicima kao na primjer:

- *Python*
- Perl[15]
- BASH (the Bourne Again Shell)[16]

[10] https://en.wikipedia.org/wiki/Zero-day_(computing)
[11] https://en.wikipedia.org/wiki/Stuxnet
[12] https://arstechnica.com/tech-policy/2012/06/confirmed-us-israel-created-stuxnet-lost-control-of-it/
[13] https://en.wikipedia.org/wiki/Shellshock_(software_bug)
[14] https://access.redhat.com/articles/1212303
[15] https://en.wikipedia.org/wiki/Perl
[16] https://www.gnu.org/software/bash/

Postoji puno vrsta skenera no za potrebe izrade ovog sustava bit će potrebno poznavati:

- Port skenere
- Skenere za pretragu direktorija na web aplikacijama

Slika 3.2 Primjer skenera za pretragu direktorija na web aplikacijama (engl. Web path finder) skenera

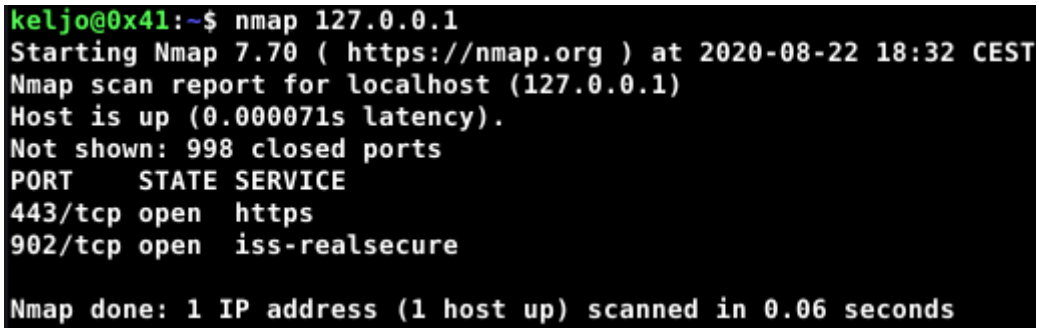
3.3.1. Port skeneri

Port skener je program namijenjen detektiranju otvorenih portova. Detektira otvorene portove tako da na primjer šalje minimalni TCP zahtjev na ciljani uređaj i port, te gleda povratne pakete iz kojih dolazi do informacije o statusu porta.

Najpoznatiji port skener je *nmap* koji je implementiran uz pomoć sljedećih programskih jezika[17]:

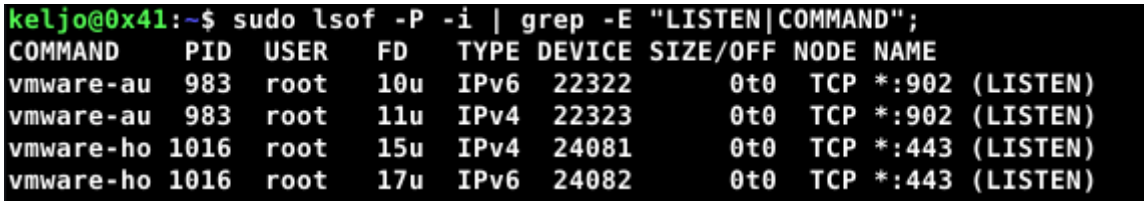
- *Python*
- C[18]
- C++[19]
- Lua[20]

[17] https://github.com/*nmap*/*nmap*
[18] https://en.wikipedia.org/wiki/C_(programming_language)
[19] https://en.wikipedia.org/wiki/C%2B%2B
[20] https://en.wikipedia.org/wiki/Lua_(programming_language)



Slika 3.3 Primjer *nmap* skena osobnog računala

Iz slike (Slika 3.3) možemo vidjeti da je *nmap* skenirao 1000 najčešće korištenih portova te je pronašao 2 otvorena porta.



Slika 3.4 Prikaz ispisa komande *lsof* koju prosljeđujemo u komandu *grep* čime izvlačimo otvorene portove

Slika 3.4 prikazuje da imamo otvorene portove 902 i 443 što potvrđuje *nmap*-ov sken.

3.3.2. Skeneri za pretragu direktorija na web aplikacijama

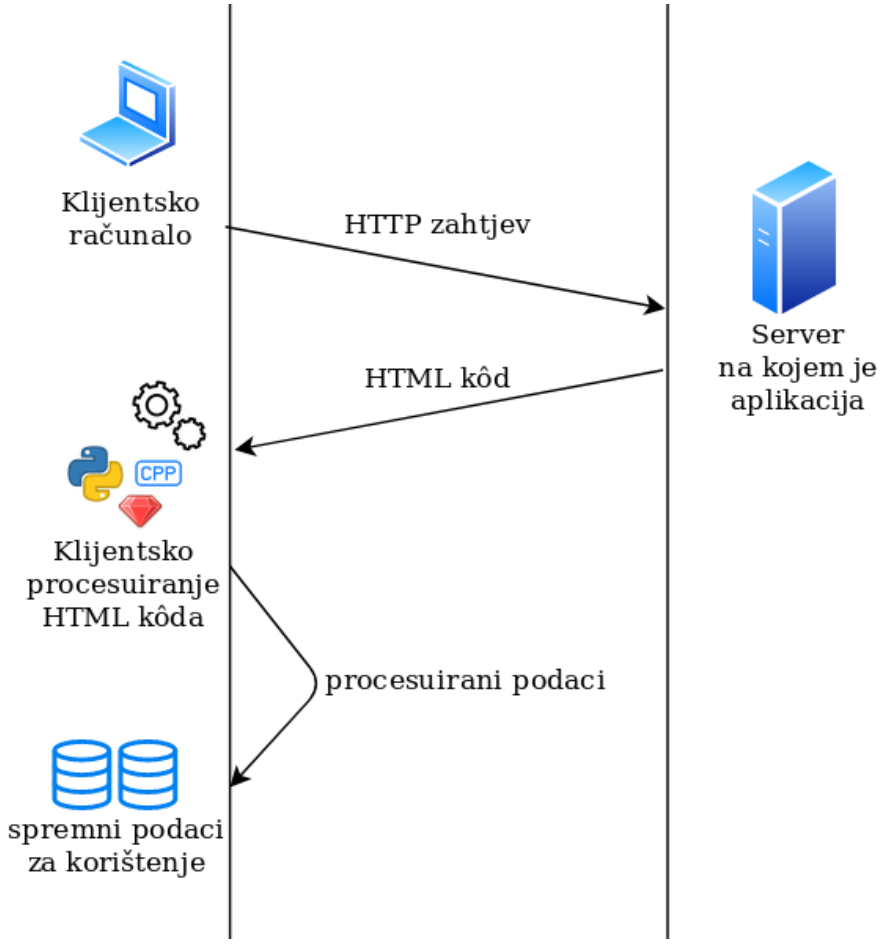
Ova vrsta skenera odnosi se na skenere čija je glavna namjena traženje ruta neke web aplikacije. Jednostavne implementacije se najčešće rade u skriptnim jezicima dok se ozbiljnije implementacije takvih skenera rade u C/C++ programskom jeziku radi brzine. Takvi skeneri najčešće koriste liste najčešćih riječi (engl. *wordlists*) za provjere ruta web aplikacije. Najlakša metoda provjere postoji li neka ruta je dohvaćanje statusnog koda iz zaglavlja odgovora (engl. *response header*). Ukoliko je statusni kod 404 ili riječima „nije pronađeno“ (engl. *not found*) , možemo zaključiti da ta ruta ne postoji ili nam ona nije dostupna. Neke implementacije web aplikacija koriste statusni kod 404 za rute koje zapravo trebaju vraćati 403 zabranjeno (engl. *forbidden*). Programeri to rade iz potencijalno sigurnosnih razloga, no to nije dobra praksa[21].

[21] https://en.wikipedia.org/wiki/Security_through_obscurity

3.4. Izvlačenje podataka s interneta (engl. *web scraping*)

Web scraping je termin koji se koristi kada neki program dohvaća bilo kakve podatke s neke web stranice. Podaci se mogu izvlačiti iz programskog sučelja (engl. *application programming interface*), ali se također može obraditi HTML[22] kôd stranice.

Najčešći tok programa koji radi izvlačenje podataka je sljedeći



Slika 3.5 Tok „web scrapinga"

[22] https://en.wikipedia.org/wiki/HTML

4. Priprema za implementaciju sustava

Kako ne bi instalirali nepotrebne pakete na sustav domaćina (engl. *host system*) koristit ćemo *Python* virtualno okruženje. Virtualna okruženja pružaju odvojene izvršne datoteke (engl. *executables*). Također možemo dodavati razne programske biblioteke (engl. *library*) koji se instaliraju direktno u naše virtualno okruženje. Postavljanjem virtualnog okruženja pružamo našem sustavu uvijek iste verzije alata koji omogućuju stabilnost i pouzdanost pokretanja sustava.

Za izradu sustava koristit ćemo *Python* verziju 3 ili više. Uz standardne *Python* biblioteke koristit ćemo također i:

- *requests*[23]
- *Beautiful Soup*[24]
- *python-nmap*[25]
- *Jinja2*[26]

4.1. Komponente sustava

Sustav se sastoji od 4 glavne komponente:

- Skener za direktorije na web aplikaciji
- Port skener
- Generator izvješća
- Jezgra samog sustava (kôd koji spaja sve komponente u jedan centralizirani sustav)

4.1.1. Kôd sustava

Glavni dio kôda sustava (jezgra samog sustava) je organizirana u jednu datoteku koja uvozi metode iz biblioteki. Sustav je implementiran da se koristi jedino u terminalu. Kod

[23] https://requests.readthedocs.io/en/master/
[24] https://crummy.com/software/BeautifulSoup
[25] https://xael.org/pages/*Python-nmap*-en.html
[26] https://jinja.palletsprojects.com/en/2.11.x/

pokretanja sustav prima samo jedan parametar, a to je IP adresa usluge koju testiramo. Koristeći *Python* *argparse* biblioteku, dohvaćamo taj argument kako bi ga koristili dalje u sustavu.

```
1 def main():
2     p = argparse.ArgumentParser()
3     p.add_argument("ip", help="IP address of server you want to test")
4     if len(sys.argv) != 2:
5         p.print_help()
6         sys.exit(1)
7     a = p.parse_args()
8     if ':' in a:
9         print("Provide an IP address without port.. il parse it myself")
10        a = a.split(":")[0]
11        print("Continuing with %s" % (a,))
12    ip = parse_ip(a.ip)
13    start_scan(ip)
```

Slika 4.1 Prikaz *main* funkcije sustava u kojem se vidi korištenje *argparse* biblioteke za dohvaćanje argumenata kod pokretanja sustava

Kôd sustava je napravljen na način da je svaki korisnički unos pokriven *try-catch* ili *if-else* blokom kôda tako da krajnji korisnik ne vidi nikakve iznimke (engl. Exceptions) kod krivog unosa, već predefinirane instrukcije kako ispravno unijeti neki podatak.

Za potrebe provjere IP adrese koju je korisnik unio, koristimo *inet_aton* metodu i *error* tip iznimke iz *socket* biblioteke.

```
1 from socket import error as WrongIpException
2 from socket import inet_aton as check_ip
```

Slika 4.2 Prikaz uvoza potrebnih metoda iz biblioteke „socket“

Error i *inet_aton* uvezli smo s posebnim imenima, *inet_aton* je uvezen kao *check_ip*, dok je *error* uvezen kao *WrongIpException*. Koristili smo ove dvije metode jer nismo morali implementirati svoje metode provjera IP adrese. Ovakva provjera radi jer metoda *inet_aton* baca *error* iznimku ukoliko primi nevažeću IP adresu.

```
1 def parse_ip(ip):
2     try:
3         check_ip(ip)
4     except WrongIpException:
5         print("ip %s is not valid" % (ip,))
6         sys.exit(1)
7     return ip
```

Slika 4.3 Prikaz funkcije koja provjerava ispravnost unesene IP adrese

4.1.2. Pomoćne skripte za lakšu pripremu okruženja

Korištenje vanjskih (ne standardnih) biblioteka zahtjeva instalaciju istih na naš sustav ili u naše virtualno okruženje. Kako bi sustav napravili pristupačnijim koristit ćemo *Makefile*[27]. *Makefile* je datoteka koja u sebi ima niz direktiva koje se mogu pokrenuti jednom komandom. U našem slučaju, *Makefile* će nam omogućiti dohvaćanje i instaliranje potrebnih vanjskih biblioteka jednim pokretanjem *Makefile*-a.

```
keljo@0x41:~/github/dante(master)$ make pypack
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (from -r requirements.txt (line 1)) (2.21.0)
Requirement already satisfied: bs4 in /home/keljo/.local/lib/python3.7/site-packages (from -r requirements.txt (line 2)) (0.0.1)
Requirement already satisfied: python3-nmap in /home/keljo/.local/lib/python3.7/site-packages (from -r requirements.txt (line 3)) (1.4.8)
Requirement already satisfied: Jinja2 in /home/keljo/.local/lib/python3.7/site-packages (from -r requirements.txt (line 4)) (2.11.2)
Requirement already satisfied: beautifulsoup4 in /usr/lib/python3/dist-packages (from bs4->-r requirements.txt (line 2)) (4.7.1)
```

Slika 4.4 Prikaz pokretanja *Makefile*-a koristeći *pypack* parametar

Unutar *Makefile*-a definirali smo *pypack* na sljedeći način

```
pypack:
	@bash -c "pip3 install -r requirements.txt";
```

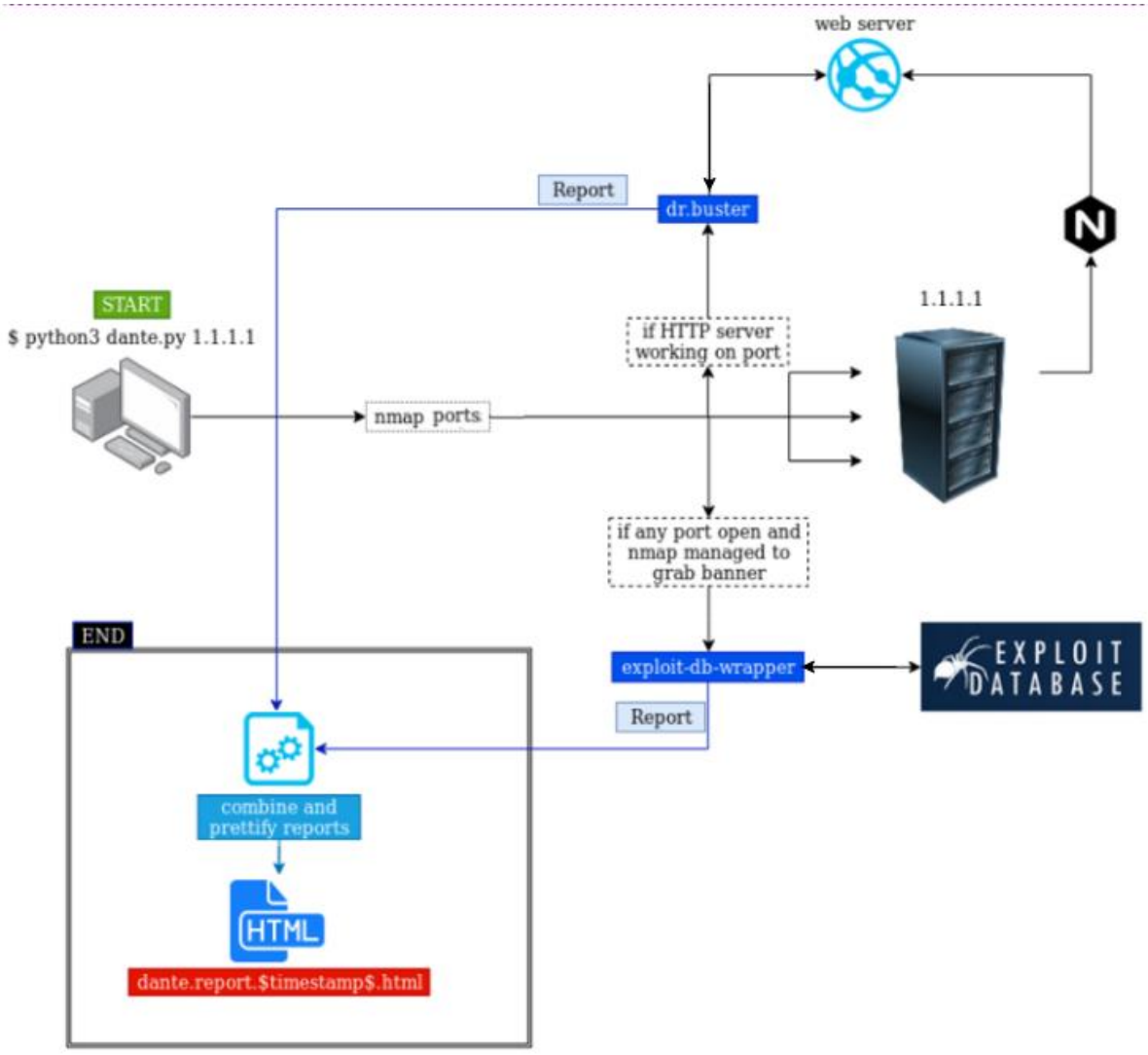
Što znači da *pypack* izvrši naredbe putem *BASH-a*. Konkretno naredbu *pip3 install -r requirements.txt* koja tada instalira sve pakete navedene *requirements.txt* datoteci

4.1.3. Tijek rada sustava

Dante je kompleksni sustav koji koristi razne biblioteke kako bi ispravno funkcionirao. Pruža vrlo intuitivno i lagano sučelje za krajnjeg korisnika, no razvijanje sustava je kompleksan proces te je potrebno znanje iz više dijelova iz područja računalnog inženjerstva.

Kako bi lakše objasnili proces rada Dantea prikazati ćemo njegov tijek rada kroz dijagrame

[27] https://en.wikipedia.org/wiki/Makefile



Slika 4.5 Prikaz dijagrama rada Dantea

Na Slika 4.5 vidimo kako se inicijalno pokretanje Dantea čini vrlo jednostavno. Dante je potrebno pokrenuti samo s jednim parametrom, a to je IP adresa sustava kojeg testiramo.

Dante prvo kreće sa skeniranjem portova, ovisno jesmo li odabrali skeniranje samo top 1000 najčešće korištenih portova ili ćemo skenirati sve portove, Dante pokreće *nmap* s tim parametrima.

Nakon skeniranja portova, ukoliko postoje otvoreni portovi na kojima rade HTTP serveri[28], Dante pokreće sken direktorija na tom portu koristeći dr.buster. Paralelno s time Dante provjerava postoje li kakve ranjivosti za pronađene servise koji poslužuju na tim portovima. Za navedeno je zadužen exploit-db-wrapper. Nakon prikupljenog izvješća dr.bustera i

[28] https://en.wikipedia.org/wiki/Web_server

exploit-db-wrapper-a Dante generira finalno izvješće koristeći *Jinja* pogonski sustav za razvoj predložaka (engl. *templating engine*). Takvo izvješće možemo otvoriti putem omiljenog internet preglednika

5. Implementacija sustava

Koristeći samo jednu datoteku, Dante je i dalje napisan urednim i čistim kôdom. Penetracijskom testeru je ključno vrijeme, i što prije odradi testiranje, to prije može naručitelju potvrditi da je sustav siguran ili da je potrebno napraviti zakrpe na servisima. Korištenje jedne datoteke (uz izuzetak korištenja vanjskih biblioteka koje se moraju uvesti) omogućuje brzo i lagano pokretanje.

5.1. Integracija Python programskog jezika s nmap-om

Programski jezik *Python* nema uključenu podršku za *nmap* u standardnim bibliotekama. Radi lakše i jednostavnije implementacije ovakvog sustava koristit ćemo *python-nmap* biblioteku koja nam omogućuje korištenje *nmap*-a kroz programski kôd u *Python*-u. *Python-nmap* nam omogućuje puno funkcionalnosti i fleksibilnosti. Dante koristi *PortScanner* objekt iz biblioteke. Detaljnije, Dante koristi metodu *scan* kako bi započeo skeniranje. Skenirati može na 2 načina:

- Najpoznatiji portovi – koristi sljedeće opcije kod pokretanja *nmap*-a
  - -*T4* kako bi *nmap* koristio 4 dretve za skeniranje
  - -*sV* kako bi *nmap* dohvatio više informacija o servisima koji rade na otvorenim portovima
  - --*version-intensity=1* govori kojim intenzitetom *nmap* treba skenirati verzije
- Svi portovi
  - -*T4* kako bi *nmap* koristio 4 dretve za skeniranje
  - -*sV* kako bi *nmap* dohvatio više informacija o servisima koji rade na otvorenim portovima
  - --*version-intensity=1* govori kojim intenzitetom *nmap* treba skenirati verzije
  - -*p*- što govori *nmap*-u da skenira sve portove

Dante te rezultate sprema, te ih koristi u daljnjoj obradi.

5.2. Implementacija skenera za direktorije

Implementacija skenera za direktorije može biti trivijalna stvar ukoliko koristimo vanjske biblioteke, no napraviti skener koji radi kvalitetno i brzo bez vanjskih biblioteka zahtjeva određenu razinu znanja o programiranju i internetskim protokolima.

Skripta (skener) koju ćemo mi implementirati naziva se dr.buster.

Dr.busteru su potrebna 2 parametra kada se poziva u samostalnom modu a to su adresa računala i put do datoteke koja sadrži rječnik za testiranje. Kako bi radio što je brže moguće dr.buster prvotno rasporedi određeni broj riječi svakom procesu. Način na koji određuje broj procesa je sljedeći: Ukoliko je broj jezgri na procesoru na kojem pokrećemo dr.buster manji ili jednak od 4 stvaramo 32 nova procesa, a ukoliko je veći od 4 stvaramo 64 novih procesa. Svaki taj proces onda šalje zasebne zahtjeve za svaku riječ, te javlja status na ekran ukoliko pronađe novu rutu. Također, prije samog pokretanja dr.buster provjeri ispravnost dane adrese tako da je raščlani na dijelove definirane u RFC-u 1738[29]. Nakon potvrde da je adresa dobra, dr.buster napravi inicijalni zahtjev kako bi utvrdio radi li računalo koju testiramo, te koji statusni kôd vraća ukoliko tražena ruta ne postoji. To radi na način da pošalje zahtjev na rutu koja sigurno neće postojati na serveru, znači nasumični niz znakova te uzima i sprema statusni kôd koji je dobio.

Dr.buster koristi biblioteke *socket* i *ssl* kako bi radio HTTP zahtjeve. Najvažnija funkcija u dr.busteru je *get_gode* koja „troši“ najviše vremena jer se poziva za svaku riječ iz rječnika. Ona provjerava radi li HTTPS na serveru ili je HTTP, te sprema to u varijablu. Tu varijablu kasnije koristi kako bi utvrdila koji tip zahtjeva treba slati, te treba li omotati naš zahtjev SSL omotnicom. Nakon slanja zahtjeva, prima minimalni broj znakova 12 jer se iz njih može iščitati statusni kôd koji je server vratio za taj zahtjev.

Koristeći maksimalne optimizacije u kôdu ova skripta je u rangu s ostalim skriptama za te svrhe, no ona je potpuno prilagođena našem sustavu, te Dante može maksimalno iskoristiti dr.busterove sposobnosti.

[29] https://tools.ietf.org/html/rfc1738

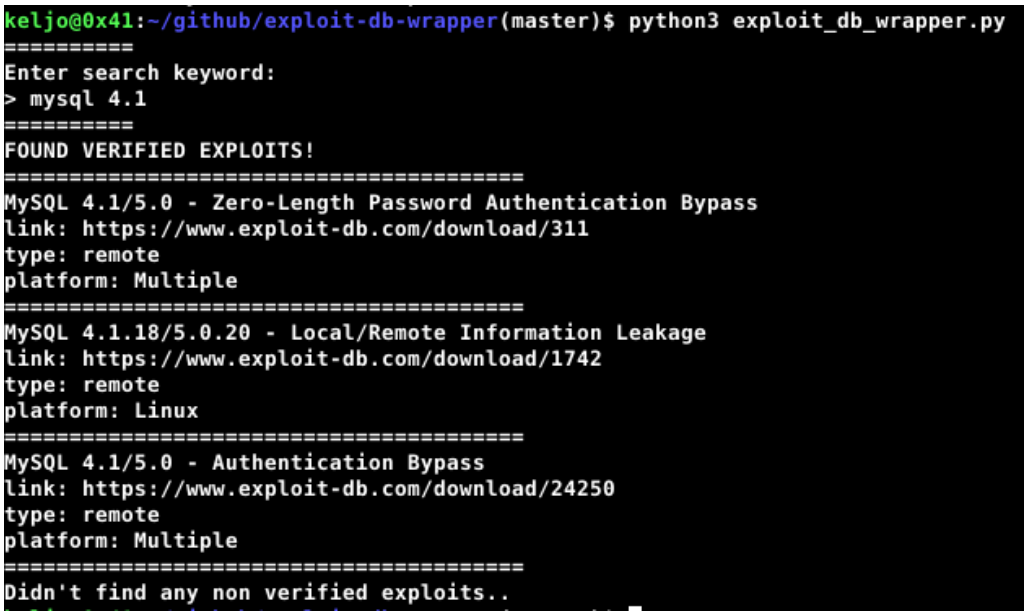
5.3. Implementacija omotnice oko exploit-db web stranice pomoću Pythona

Dobiveni podaci pomoću *nmap*-a ne znače nam mnogo. Verzije servisa i imena bi morali sami tražiti po internetu, te bi morali tražiti postoje li ranjivosti za to. Automatiziranje tog procesa omogućio nam je alat imena *searchsploit*[30]. Služi kao tražilica za ranjivosti.

Problem kod *searchsploit*-a je taj da bi morali lokalno imati skinutu cijelu *exploit*-db bazu. To znači da bi prilikom prvog pokretanja Dantea morali napuniti tu veliku bazu i to bi trajalo dosta vremena. Radi lakšeg i bržeg korištenja našeg sustava implementiran je *explot-db-wrapper*. On služi kao omotnica oko *exploit*-db internet stranice. Može se koristiti kao samostalna skripta, ali i kao biblioteka. Za potrebe Dantea koristit ćemo je kao biblioteku.

Sama implementacija biblioteke svodi se na korištenje *requests* biblioteke za uspostavljanje konekcije i dohvaćanje JSON[31] sadržaja s HTTP servera. Nakon dohvaćanja traženih podataka, *explot-db-wrapper* procesuira te formatira i ovisno koristimo li ga kao biblioteku ili samostalnu skriptu, podatke ispisuje na ekran ili vraća kao rječnik. Podaci koji se procesuiraju sastoje se od informacija o servisima, potvrđenim ranjivostima, nepotvrđenim ranjivostima te linkovima na mjesta gdje se može preuzeti kôd koji iskorištava ranjivost. U daljnjem kontekstu „kôd koji iskorištava ranjivost“ nazivat ćemo *exploit* radi jednostavnosti čitanja.

[30] https://github.com/offensive-security/exploitdb/blob/master/searchsploit
[31] https://www.json.org/json-en.html



Slika 5.1 Prikaz korištenja *explot-db-wrapper*-a kao samostalne skripte

Na Slika 5.1 vidimo kako je korištenje *explot-db-wrapper*-a vrlo intuitivno i lagano. Potrebno ga je samo pokrenuti i dati mu ključnu riječ koju treba tražiti. Nakon toga on nam ispisuje pronađene *exploit*-e u formatu linka, zahvaćene verzije, tip *exploit*a i platforma koje su zahvaćene tim *exploit*-om.

5.4. Implementacija generatora izvješća

Kad govorimo o izvješćima, govorimo o širokom spektru metoda prikaza istih. To može biti pdf datoteka, može biti dokument generiran putem LaTeX-a[32], ali može biti i HTML datoteka. Pisanje samog HTML-a direktno iz *Python*a bi bilo vrlo nezgrapno i naš kôd bi postao vrlo neuredan i težak za čitanje.

Jednostavno rješenje je koristiti *Jinju*[33]. Ona nam pruža dinamične načine građenja HTML-a. Primjerice možemo imati *for* petlje koje kasnije generiraju listu od N redova.

```
{% for exploit in exploits['verified'] %}
  <ul class="exploits">
    <li>{{ exploit['title'] }}</li>
    <li>type: {{ exploit['type'] }}</li>
  </ul>
```

[32] https://www.latex-project.org/
[33] https://jinja.palletsprojects.com/en/2.11.x/

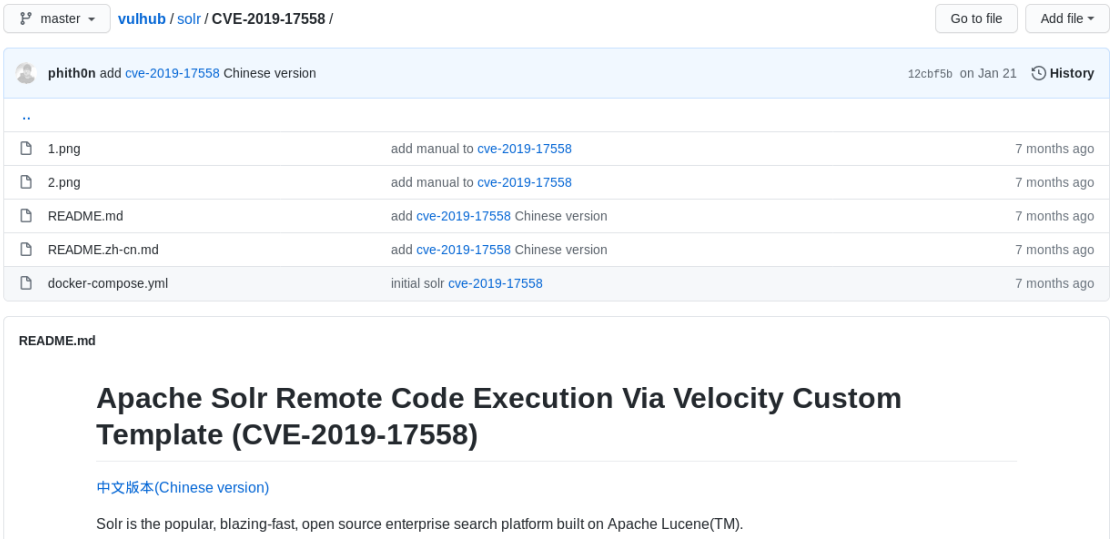
```
{% endfor %}
```

Ovaj kôd prikazuje jedan dio datoteke *template.html* koja sadrži potrebni *Jinja* kôd za generiranje krajnjeg Dante izvješća. Konkretno ovaj pojednostavljeni dio generira listu potvrđenih *exploit*-ova. To znači da koliko god *exploit*-ova mi dinamički dobijemo od *exploit-db-wrapper*-a, možemo na ovako lagani način generirati HTML kôd koji predstavlja formatirani ispis tog sadržaja.

6. Demonstracija rada sustava za potporu sigurnosnom testiranju

Testiranje Dantea ćemo provesti u lokalnom okruženju koristeći se kontejnerima. Koristit ćemo *Docker*[34] kao platformu za stvaranje i pristupanje kontejnerima. U kontejner ćemo instalirati ranjivu verziju *Apache Solr*[35] platforme. *Apache Solr* je platforma otvorenog koda koja pruža traženje riječi, indeksiranje, dinamično klasteriranje…

Ranjivu platformu skinuti ćemo s *Github* repozitorija *vulnhub*-a



Slika 6.1 Prikaz *Apache Solr* direktorija unutar repozitorija *vulnhub*-a

Nakon što smo klonirali repozitorij potrebno je pripremiti i pokrenuti *Apache Solr* u kontejner.

[34] https://www.docker.com/
[35] https://lucene.apache.org/solr/

Slika 6.2 Prikaz d*ocker-compose* naredbe

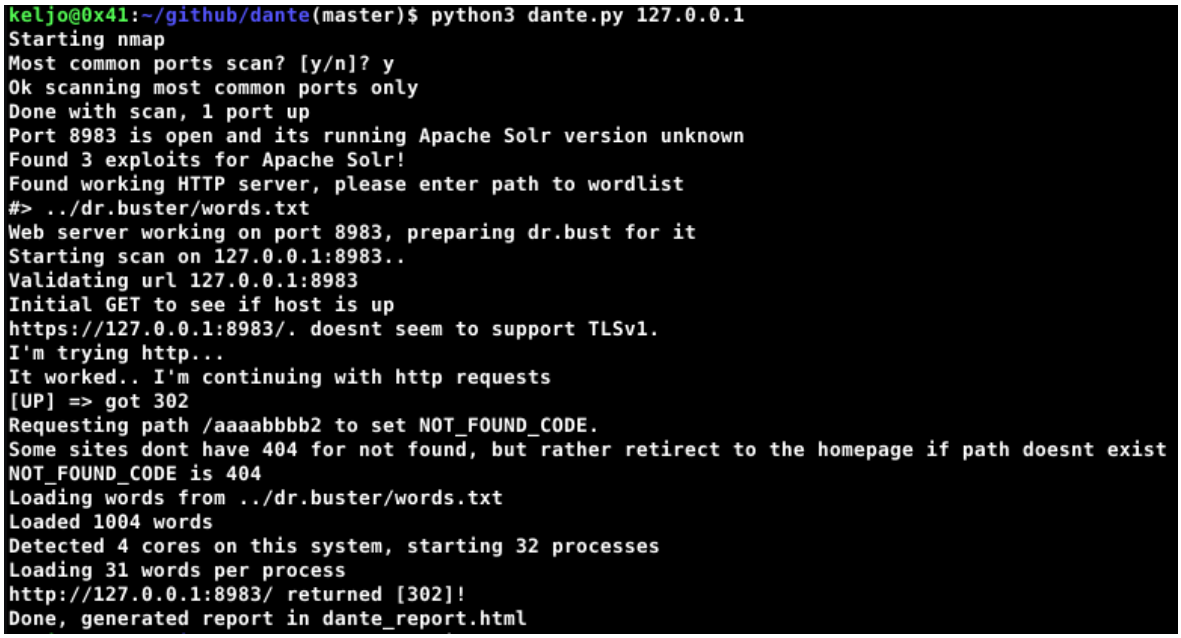
Na Slika 6.2 možemo vidjeti kako se naš kontejner uspješno izgradio te radi.

Slika 6.3 Pokretanje naredbe za pregled otvorenih portova na računalu

Slika 6.3 potvrđujemo uspješnu instalaciju i izgradnju *Apache Solr*-a u kontejner. On sada radi na portu 8939

6.1. Inicijalno skeniranje ranjivog sustava

Dante pokrećemo s parametrom 127.0.0.1 što predstavlja našu lokalnu adresu računala na kojem radi *Apache Solr* u kontejneru na portu 8983



Slika 6.4 Prikaz sučelja naredbenog retka (engl. *command line interface*) nakon pokretanja Dantea

Kao što možemo vidjeti u Slika 6.4 Dante nam je prikazao otvoreni port 8983, te je pronašao da *Apache Solr* trenutno sluša na tom portu. Također možemo vidjeti kako je započeo i sken direktorija web aplikacije. Razlog tomu je što *Apache Solr* ima svoje web sučelje. Nadalje, dr.buster nam je skenirao sve rute koje smo mu pružili iz datoteke words.txt te je pronašao kako je web stranica odgovorila potvrdno za samo jednu rutu iz datoteke words.txt. Na kraju skena možemo vidjeti kako nas je Dante obavijestio o generiranom izvješću.

6.2. Pregled izvješća kojeg je generirao sustav

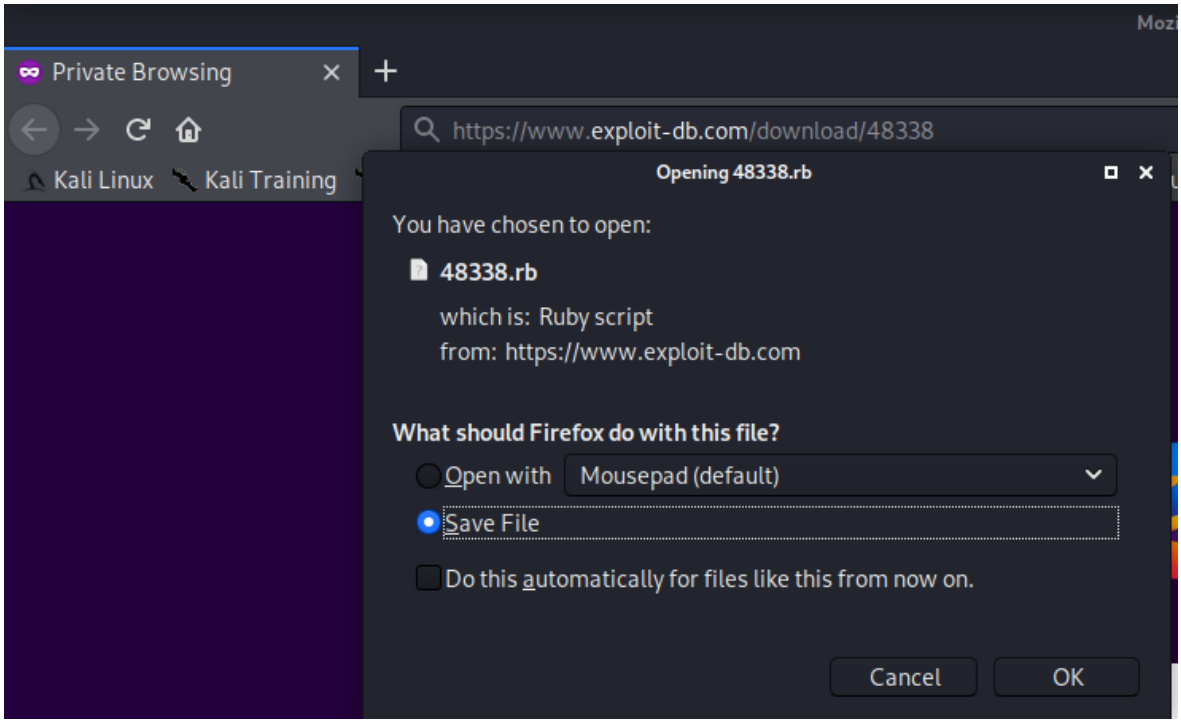
Koristeći internet preglednik otvoriti ćemo generirano izvješće

Slika 6.5 Generirano izvješće Dantea

Slika 6.5 prikazuje naše generirano izvješće. Vidimo da nam *Apache Solr* radi na portu 8939. Također vidimo kako postoje 2 potvrđena *exploit*-a, te 1 nepotvrđen. Te informacije nam je pronašao *exploit*-db-wrapper. Na dnu izvješća vidimo rute koje je pronašao dr.buster. Ukoliko bi ovdje bilo više ranjivih servisa Dante bi ih zapisao na isti način. Ovakvo izvješće je odličan početak u pisanju našeg sigurnosnog izvješća, znamo koji servisi rade, postoje li poznate ranjivosti za iste te imamo pregled pronađenih ruta ukoliko je to web aplikacija za lakše testiranje.

6.3. Korištenje kôda koji iskorištava ranjivost radi ostvarivanja kompromitacije

Nakon što nam je Dante generirao izvješće možemo iskoristiti linkove na stranice kako bi preuzeli *exploit*-e. Simulirati ćemo napadača (tj. nas kao ispitivača) kroz sustav Kali Linux[36]. On radi u virtualnoj mašini te ima pristup našoj mreži na kojoj radi kontejner s *Apache Solr*-om.



Slika 6.6 Prikaz skidanja kôda koji iskorištava ranjivost koristeći internet preglednik

Kali Linux ima već skinuti *metasploit-framework*[37], to je alat koji koriste gotovo svi sigurnosni testeri. On u sebi ima puno mogućnosti, no to je van opsega ovog rada. Nakon što smo skinuli *exploit*, potrebno ga je premjestiti u direktorij gdje se nalazi *metasploit*-ova baza *exploit*-ova.

```
root@kali:/home/kali/Downloads# ls
48338.rb
```

Slika 6.7 Prikaz naredbe *ls*

[36] https://www.kali.org/
[37] https://www.metasploit.com/

Koristeći naredbu *ls* na Slika 6.7 vidjeli smo kako naš *exploit* imena 48338.rb postoji u direktoriju.

```
root@kali:/home/kali/Downloads# cp 48338.rb /usr/share/metasploit-framework/modules/exploits/
root@kali:/home/kali/Downloads# ls -l /usr/share/metasploit-framework/modules/exploits/
total 120
-rw-r--r--  1 root root 18957 Aug 27 17:15 48338.rb
drwxr-xr-x  3 root root  4096 Aug 23 11:07 aix
drwxr-xr-x  6 root root  4096 Jul 27 13:12 android
drwxr-xr-x  5 root root  4096 Jul 27 13:12 apple_ios
drwxr-xr-x  3 root root  4096 Jul 27 13:12 bsd
drwxr-xr-x  3 root root  4096 Jul 27 13:12 bsdi
drwxr-xr-x  3 root root  4096 Jul 27 13:12 dialup
-rw-r--r--  1 root root  6491 Jul 29 18:36 example_linux_priv_esc.rb
-rw-r--r--  1 root root  2700 Jul 29 18:36 example.rb
-rw-r--r--  1 root root  6396 Jul 29 18:36 example_webapp.rb
drwxr-xr-x  3 root root  4096 Jul 27 13:12 firefox
drwxr-xr-x  9 root root  4096 Jul 27 13:12 freebsd
drwxr-xr-x  3 root root  4096 Jul 27 13:12 hpux
drwxr-xr-x  3 root root  4096 Jul 27 13:12 irix
drwxr-xr-x 23 root root  4096 Jul 27 13:12 linux
drwxr-xr-x  3 root root  4096 Jul 27 13:12 mainframe
drwxr-xr-x 27 root root  4096 Aug 23 11:07 multi
drwxr-xr-x  4 root root  4096 Jul 27 13:12 netware
drwxr-xr-x  3 root root  4096 Jul 27 13:12 openbsd
drwxr-xr-x 13 root root  4096 Jul 27 13:12 osx
drwxr-xr-x  4 root root  4096 Jul 27 13:12 qnx
drwxr-xr-x  8 root root  4096 Jul 27 13:12 solaris
drwxr-xr-x 14 root root  4096 Jul 27 13:12 unix
drwxr-xr-x 52 root root  4096 Jul 27 13:12 windows
```

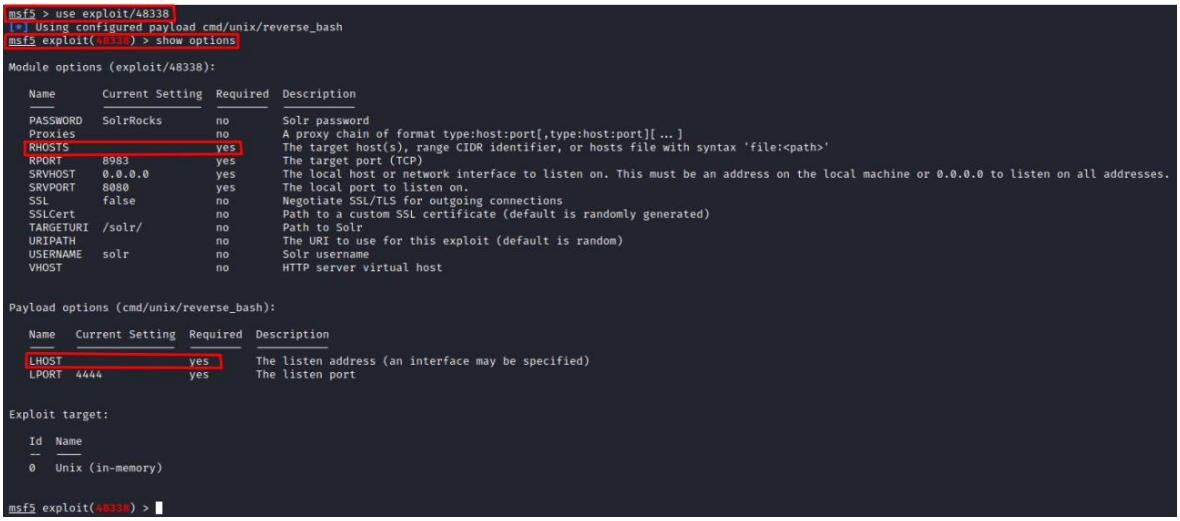
Slika 6.8 Prikaz premještanja *exploit*a u metasploit-ovu bazu

Kao što vidimo na Slika 6.8 crvenim je označen naš pripremljeni *exploit*, sada nam preostaje upaliti metasploit-framework i iskoristiti taj kôd.

```
root@kali:/home/kali/Downloads# msfconsole


      .:okOOOkdc'           'cdkOOOko:.
    .xOOOOOOOOOOOc       cOOOOOOOOOOOx.
   :OOOOOOOOOOOOOOk,   ,kOOOOOOOOOOOOOO:
  'OOOOOOOOOkkkkOOOOO: :OOOOOOOOOOOOOOOO'
  oOOOOOOOO.    .oOOOOoOOOOl.    ,OOOOOOOOo
  dOOOOOOOO.      .cOOOOOc.      ,OOOOOOOOx
  lOOOOOOOO.         ;d;         ,OOOOOOOOl
  .OOOOOOOO.   .;           ;    ,OOOOOOOO.
   cOOOOOOO.   .OOc.     'oOO.   ,OOOOOOOc
    oOOOOOO.   .OOOO.   :OOOO.   ,OOOOOOo
     lOOOOO.   .OOOO.   :OOOO.   ,OOOOOl
      ;OOOO'   .OOOO.   :OOOO.   ;OOOO;
       .dOOo   .OOOOocccxOOOO.   xOOd.
         ,kOl  .OOOOOOOOOOOOO. .dOk,
           :kk;.OOOOOOOOOOOOO.cOk:
             ;kOOOOOOOOOOOOOOk:
               ,xOOOOOOOOOOx,
                 .lOOOOOOOl.
                    ,dOd,
                      .


       =[ metasploit v5.0.101-dev                         ]
+ -- --=[ 2050 exploits - 1108 auxiliary - 344 post       ]
+ -- --=[ 562 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

Metasploit tip: Tired of setting RHOSTS for modules? Try globally setting it with setg RHOSTS x.x.x.x

msf5 > 
```

Slika 6.9 Prikaz metasploit-framework-a

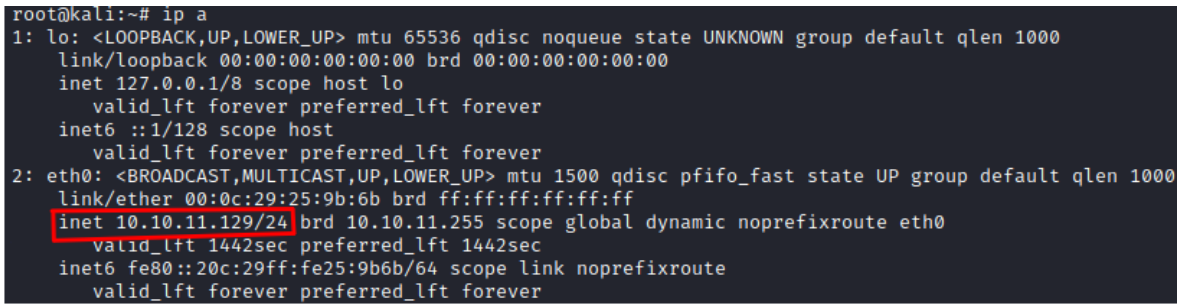
Nakon što smo upalili metasploit potrebno mu je navesti koji *exploit* želimo koristiti prikaz toga je na sljedećoj slici



Slika 6.10 Prikaz učitavanja *exploit*-a i prikaza opcija koje nudi *exploit*

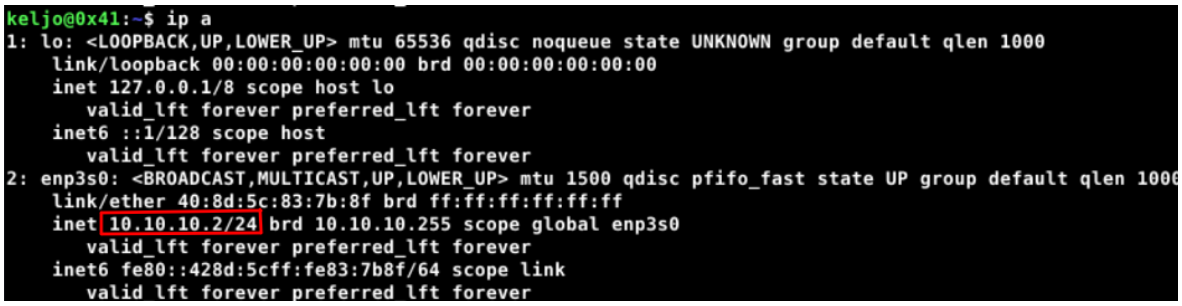
Na Slika 6.10 vidimo učitavanje *exploit*-a korištenjem ključne riječi „use“ i navođenje puta do *explot*-a (onaj direktorij u koji smo kopirali sa Slika 6.8). Nakon što smo učitali *exploit*, vrijeme je da pogledamo koje sve opcije *exploit* nudi. Nama su bitni „RHOSTS“ što predstavlja IP adresu računala kojeg napadamo, a „LHOST“ je adresa mrežnog sučelja (engl. *network interface*) koju želimo koristiti.

Koristit ćemo naredbu *ip a* kako bi došli do IP adrese Kali Linux sustava.



Slika 6.11 Prikaz naredbe *ip a* na Kali Linux sustavu

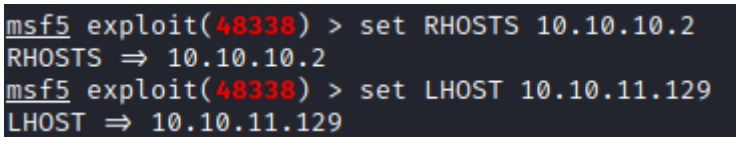
Crvenom bojom na Slika 6.11 označen je IP našeg Kali Linux napadačkog sustava koji je 10.10.11.129.



Slika 6.12 Prikaz naredbe "ip a" na sustavu gdje je *Apache Solr*

Kao i na Slika 6.11, na Slika 6.12 vidimo da je IP adresa našeg sustava gdje je *Apache Solr* 10.10.10.2.

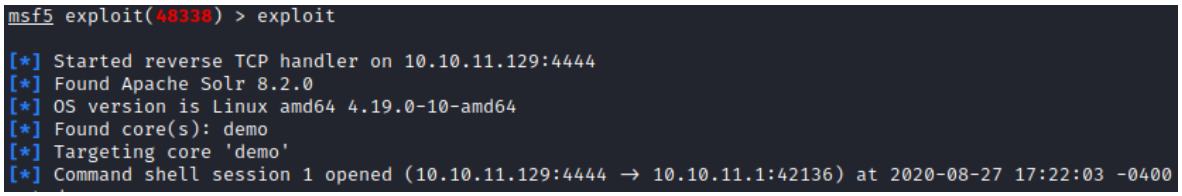
Nakon što smo prikupili ova 2 podatka spremni smo iskoristiti ranjivost na sustavu. Prije iskorištavanja ranjivosti moramo postaviti te 2 IP adrese u „LHOST“ i „RHOSTS“ varijable.



Slika 6.13 Prikaz postavljanja varijabli u metasploit-u

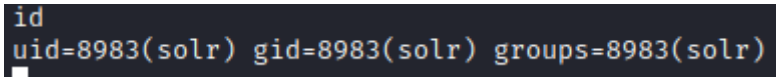
Varijable postavljamo ključnom riječi „set“ pa ime varijable i vrijednost.

Nakon postavljanja varijabli vrijeme je za kompromitaciju ranjivog sustava. U *metasploit*-u to radimo s ključnom riječi „*exploit*“.



Slika 6.14 Prikaz pokretanja *explot*-a u svrhu kompromitacije sustava

Nakon uspješnog pokretanja *exploit-a* imamo pristup ljusci (engl. shell) ranjivog sustava kao solr korisnik.



Slika 6.15 Prikaz pokretanja naredbe *id* u ljusci kompromitiranog sustava

Slika 6.15 vidimo uspješnu kompromitaciju sustava. Naredba *id* vraća informacije o trenutno prijavljenom korisniku.

Zaključak

Izrada ovog sustava bio je izazov koji je zahtijevao znanje iz više područja (mreže, sigurnost i programiranje). Tokom izrade sam spoznao kako je sigurnost još širi pojam nego što sam mislio. Postoji bezbroj parametara koji mogu narušiti sigurnosti nekog sustava. Tokom osmišljavanja i programiranja Dantea morao sam konstantno vagati između kompleksnosti sustava i korisnosti. Što sam više pojednostavio sustav to je više korisnost padala i obrnuto.

Korištenje ovog sustava napravljeno je veoma lagano za krajnjeg korisnika, dok je rezultat koristan. Stoga zaključujem da sam uspio zadovoljiti tu točku.

Ukoliko pokrenemo ovakav sustav nad nekim serverom i on ne pronađe niti jednu ranjivost, možemo reći da smo na dobrom tragu sigurnosti tog servera. Razlog tomu je taj što ne možemo za nijedan sustav reći da je sto postotno siguran. Zlonamjerni hakeri svakim danom traže nove načne i rupe kako bih ostvarili kompromitaciju sustava. Ovim radom sam pridonio zajednici svih dobronamjernih hakera pri izvršavanju njihova posla. Smatram da se ovaj rad može dodatno nadograditi, i s time postati i kompleksniji i korisniji, no to je trenutno previše kompleksno za moje znanje. Nakon određenog sakupljenog iskustva smatram da ću biti u mogućnosti nadograditi ovaj rad, tako da pruža još više mogućnosti korisnicima, no držati ću se načela da uvijek bude jednostavan za korištenje, i da nikada ne postane veliki alat za koji je potrebno preuzeti više različitih programskih paketa kako bih ga pokrenuli.

Popis kratica

HTTP	*Hypertext Transfer Protocol*	metoda prijenosa informacija na Internetu
HTTPS	*Hypertext Transfer Protocol Secure*	sigurna metoda prijenosa informacija na Internetu
HTML	*Hypertext markup language*	prezentacijski jezik za izradu internet stranicana
SSL	*Secure Sockets Layer*	sigurni transportni protocol
IP	*Internet Protocol*	mrežni protokol
URL	Uniform Resource Locator	putanja do određenog sadržaja na Internetu
JSON	*JavaScript Object Notation*	format spremanja podataka
CVE	*Common Vulnerabilities and Exposures*	sustav referenciranja javnih ranjivosti
DNS	*Domain Name System*	sustav imenovanja uređaja na nekoj mreži

Popis slika

Literatura

[1] WEIDMAN, G. *Penetration Testing.* No Strach Press *2014*

[2] ZALEWSKI, M. *Tangled Web*. No strach Press 2011.

[3] KENNEDY, D. GORMAN, J. KEARNS, D. AHARONI, M. *Metasploit.* No Strach Press 2011.

[4] ERICKSON, J. *Hacking, 2nd Edition. No Strach Press 2008.*

[5] SEITZ, J. *Black Hat Python. No Strach Press 2014.*

[6] OCCUPYTHEWEB. *Linux Basics for Hackers. No Strach Press 2018.*

[7] SWEIGART, A. *Automate the Boring Stuff with Python. No Strach Press 2019.*

[8] SWAROOP, C. H. *A Byte of Python. Swaroop C H 2013.*

[9] LUTZ, M. *Learning Python, 5th Edition. O'Reilly Media, Inc. 2013.*

[10] LUSTH, J. *The Art and Craft of Programming, Python Edition. University of Alabama 2016.*

[11] CHOU, E. *Mastering Python Networking. Packt 2017.*

[12] SEITZ, J. *Black Hat Python: Python Programming for Hackers and Pentesters. No Strach Press 2014.*

[13] SEITZ, J. *Gray Hat Python: Python Programming for Hackers and Reverse Engineers. No Strach Press 2009.*

[14] SILBERSCHATZ, A. GALVIN, P. GAGNE, G. *Operating System Concepts, 10th Edition. Wiley 2018.*

[15] RAMALHO, L. *Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media, Inc. 2015.*

Prilog

ALGEBRA VISOKO UČILIŠTE

IZRADA SUSTAVA POTPORE PENETRACIJSKOM TESTIRANJU

Pristupnik: Karlo Kegljević, JMBAG

Mentor: Robert Petrunić, P.M.Comp.Eng