

Examen Análisis de vulnerabilidades

Cadena válida (password) generada: **rafae-l.val-lejo.-*+456-7r95m**

- Lo primero que realicé fue ver que tipo de archivo era con file
- Esto devolvió que era un archivo gzip

```
[root@parrot]-[/home/user/Downloads/Examen]
#file SHELLow
SHELLow: gzip compressed data, last modified: Fri Mar 31 19:11:39 2017, from Unix, original size 10240
```

- Lo descomprimí con tar:

```
[root@parrot]-[/home/user/Downloads/Examen]
#tar xzf SHELLow
```

- Al tratar de ejecutarlo da un error:

```
[root@parrot]-[/home/user/Downloads/Examen]
#file shell_mod2
shell_mod2: ELF, unknown class 113
```

```
[root@parrot]-[/home/user/Downloads/Examen]
#./shell_mod2
bash: ./shell_mod2: cannot execute binary file: Exec format error
[x]-[root@parrot]-[/home/user/Downloads/Examen]
```

- Analicé el contenido del binario con strings:

```

[root@parrot]-[/home/user/Downloads/Examen]
#strings shell_mod2
ELFquitaestoparaquefuncioneelprograma
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
ffffff.
Baia, baH
ia...SH
i que haH
s llegadH
o lejos
It's timH
e to craH
ckme Mish
s/Mr RevH
erse EngH
inner ;)H

```

- Al revisar el contenido, nos indica que debe quitarse la línea de la cabecera para poder ejecutar el programa. Para ello:

```

[root@parrot]-[/home/user/Downloads/Examen]
#sed 's/quitaestoparaquefuncioneelprograma//' shell_mod2 > shell_funciona
[root@parrot]-[/home/user/Downloads/Examen]
#file shell_funciona
shell_funciona: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=4ab82577a85ffa8894e95109fb63bdd2f199
903f, not stripped

```

- Ahora que se obtuvo el tipo de archivo con file, nos indica que es un ejecutable para 64 bits con las bibliotecas compiladas de manera dinámica.

Se le da permiso de ejecución al binario obtenido:

```

[root@parrot]-[/home/user/Downloads/Examen]
#chmod +x shell_funciona

```

- Se obtiene el archivo descartando los símbolos:

```

[root@parrot]-[/home/user/Downloads/Examen]
#strip shell_funciona -o shell_funciona_stripped

```

- Lo abrí con strings para ver las cadenas imprimibles del binario, aquí se obtiene una especie de serial (password) que posiblemente se utilice más adelante.

```
[user@parrot]--[~/Downloads/Examen]
$strings shell_funciona_stripped
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
ffffff.
Baia,baH
ia ... sH
i que haH
s llegadH
o lejos
It's timH
e to craH
ckme MisH
s/Mr RevH
erse EngH
inner ;)H
[]A\A]A^A_
;*3$"
87654-32109-87654-321DRO-WSSAP
SHELLow was here :P
8}_b
6[J4
{B;H~
GCC: (Debian 4.9.2-10) 4.9.2
GCC: (Debian 4.8.4-1) 4.8.4
.shstrtab
.interp
```

- Abri el binario con strace (análisis dinámico) para ver las llamadas al sistema que realiza

```
[root@parrot]-[/home/user/Downloads/Examen]
#strace ./shell_funciona
execve("./shell_funciona", [ "./shell_funciona", 0x7ffda268c150 /* 21 vars */ ] = 0
brk(NULL) = 0x1a49000
access("/etc/ld.so.preload", R_OK) = 0
openat(AT_FDCWD, "/etc/ld.so.preload", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
close(3) = 0
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=215036, ...}) = 0
mmap(NULL, 215036, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3d0b76c000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0\0\0\1\0\0\0\260A\2\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1824496, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3d0b76a000
mmap(NULL, 1837056, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3d0b5a9000
mprotect(0x7f3d0b5cb000, 1658880, PROT_NONE) = 0
mmap(0x7f3d0b5cb000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f3d0b5cb000
mmap(0x7f3d0b713000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16a000) = 0x7f3d0b713000
mmap(0x7f3d0b760000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b6000) = 0x7f3d0b760000
mmap(0x7f3d0b766000, 14336, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3d0b766000
close(3) = 0
arch_prctl(ARCH_SET_FS, 0x7f3d0b76b500) = 0
mprotect(0x7f3d0b760000, 16384, PROT_READ) = 0
mprotect(0x7f3d0b7c8000, 4096, PROT_READ) = 0
munmap(0x7f3d0b76c000, 215036) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL) = 0x1a49000
brk(0x1a6a000) = 0x1a6a000
write(1, "Baia, baia ... si que has llegado"..., 40Baia, baia ... si que has llegado lejos
) = 40
write(1, "It's time to crackme Miss/Mr Rev"..., 49It's time to crackme Miss/Mr Reverse Enginner ;)
) = 49
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(39321), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 0) = 0
accept(3, NULL, 0x10) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
accept(3, NULL, 0x10) = ? ERESTARTSYS (To be restarted if SA_RESTART is set)
--- SIGWINCH {si_signo=SIGWINCH, si_code=SI_KERNEL} ---
accept(3, NULL, 0x10)
```

- Se observa que se abre el puerto 39321 y para establecer la conexión, lo hice mediante nc

```
[user@parrot]-[~]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
```

- Ahora que tenía un panorama mayor del programa, abrí el binario con gdb para ver y comprender su funcionamiento:

```
[root@parrot]-[/home/user/Downloads/Examen]
#gdb -q shell_funciona
Reading symbols from shell_funciona...(no debugging symbols found)...done.
(gdb)
```

- Primero hice un disas a main para ver las llamadas a funciones que se realizan en el programa además de ver en que punto se establece la comunicación del socket (obtenido con strace).

```

(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400506 <+0>:      push    rbp
   0x0000000000400507 <+1>:      mov     rbp, rsp
   0x000000000040050a <+4>:      sub     rsp, 0x70
   0x000000000040050e <+8>:      movabs  rax, 0x6162202c61696142
   0x0000000000400518 <+18>:     mov     QWORD PTR [rbp-0x30], rax
   0x000000000040051c <+22>:     movabs  rax, 0x73202e2e2e206169
   0x0000000000400526 <+32>:     mov     QWORD PTR [rbp-0x28], rax
   0x000000000040052a <+36>:     movabs  rax, 0x6168206575712069
   0x0000000000400534 <+46>:     mov     QWORD PTR [rbp-0x20], rax
   0x0000000000400538 <+50>:     movabs  rax, 0x646167656c6c2073
   0x0000000000400542 <+60>:     mov     QWORD PTR [rbp-0x18], rax
   0x0000000000400546 <+64>:     movabs  rax, 0x736f6a656c206f
   0x0000000000400550 <+74>:     mov     QWORD PTR [rbp-0x10], rax
   0x0000000000400554 <+78>:     movabs  rax, 0x6d69742073277449
   0x000000000040055e <+88>:     mov     QWORD PTR [rbp-0x70], rax
   0x0000000000400562 <+92>:     movabs  rax, 0x617263206f742065
   0x000000000040056c <+102>:    mov     QWORD PTR [rbp-0x68], rax
   0x0000000000400570 <+106>:   movabs  rax, 0x73694d20656d6b63
   0x000000000040057a <+116>:   mov     QWORD PTR [rbp-0x60], rax
   0x000000000040057e <+120>:   movabs  rax, 0x76655220724d2f73
   0x0000000000400588 <+130>:   mov     QWORD PTR [rbp-0x58], rax
   0x000000000040058c <+134>:   movabs  rax, 0x676e452065737265
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000400596 <+144>:   mov     QWORD PTR [rbp-0x50], rax
   0x000000000040059a <+148>:   movabs  rax, 0x293b2072656e6e69
   0x00000000004005a4 <+158>:   mov     QWORD PTR [rbp-0x48], rax
   0x00000000004005a8 <+162>:   mov     BYTE PTR [rbp-0x40], 0x0
   0x00000000004005ac <+166>:   lea     rax, [rbp-0x30]
   0x00000000004005b0 <+170>:   mov     rdi, rax
   0x00000000004005b3 <+173>:   call    0x4003e0 <puts@plt>
   0x00000000004005b8 <+178>:   lea     rax, [rbp-0x70]
   0x00000000004005bc <+182>:   mov     rdi, rax
   0x00000000004005bf <+185>:   call    0x4003e0 <puts@plt>
   0x00000000004005c4 <+190>:   mov     QWORD PTR [rbp-0x8], 0x600a40
   0x00000000004005cc <+198>:   mov     rdx, QWORD PTR [rbp-0x8]
   0x00000000004005d0 <+202>:   mov     eax, 0x0
   0x00000000004005d5 <+207>:   call    rdx

```

- En main+190, luego de poner el mensaje con puts, se mueve el contenido de la dirección 0x600a40, por lo que coloqué un break en esa dirección para ver lo que ocurre en ese punto.

```

(gdb) b *0x600a40
Breakpoint 1 at 0x600a40

```

- Al correr el programa, se detiene en el break marcado y se llega a la función shellcode, entonces con step into, comienzo a analizarla y entender su funcionamiento:

```
(gdb) r
Starting program: /home/user/Downloads/Examen/shell_funciona
Baia, baia ... si que has llegado lejos
It's time to crackme Miss/Mr Reverse Enginner ;)

Breakpoint 1, 0x0000000000600a40 in shellcode ()
(gdb) █
```

- Al llegar a shellcode+69, se realiza la llamada a la función accept, por lo que desde otra terminal me conecto a localhost en el puerto 39321 (obtenido anteriormente)

The image shows a debugger window with assembly code and a terminal window. The assembly code is as follows:

```
0x600a7e <shellcode+62> pop    rsi
0x600a7f <shellcode+63> mov    al,0x32
0x600a81 <shellcode+65> syscall
0x600a83 <shellcode+67> mov    al,0x2b
0x600a85 <shellcode+69> syscall
> 0x600a87 <shellcode+71> push   rax
0x600a88 <shellcode+72> pop    rdi
0x600a89 <shellcode+73> xor     rdx,rdx
```

The terminal window shows the following commands and output:

```
[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
```

- En shellcode+86, se realiza la llamada a read, por lo que escribo el serial (password) que obtuve antes con strings, en la conexión establecido en la otra terminal:

The image shows a debugger window with assembly code and a terminal window. The assembly code is as follows:

```
0x600a88 <shellcode+72> pop    rdi
0x600a89 <shellcode+73> xor     rdx,rdx
0x600a8c <shellcode+76> mov     dl,0x85
0x600a8e <shellcode+78> sub     dl,0x65
0x600a91 <shellcode+81> push    rsp
0x600a92 <shellcode+82> push    rsp
0x600a93 <shellcode+83> pop     rsi
0x600a94 <shellcode+84> xor     eax,eax
0x600a96 <shellcode+86> syscall
> 0x600a98 <shellcode+88> xor     rax,rax
0x600a9b <shellcode+91> mov     al,0x4a
0x600a9d <shellcode+93> sub     al,0x40
```

The terminal window shows the following commands and output:

```
[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
87654-32109-87654-321DR0-WSSAP
```

- Ahora continué con step into para ver que hace con la cadena que se lee:


```

rax - 0xa 10
rcx - 0x3 3
rsi user@parrot 0x7fffffff400 140737488348160
rbp - 0x7fffffff480 0x7fffffff480
r8 - 0x7ffff7f9a500 140737353721088
r10 0x601010 6295568
r12 0x400410 4195344
r14 0x0 0
rip 0x600aa2 0x600aa2 <shellcode+98>
cs libc_start 0x33 51
ds gmon_start 0x0 0
fs 0x0 0
LIBC 2.2.5

0x600a93 <shellcode+83> pop rsi
0x600a94 <shellcode+84> xor eax,eax
0x600a96 <shellcode+86> syscall
0x600a98 <shellcode+88> xor rax,rax
0x600a9b <shellcode+91> mov al,0x4a
0x600a9d <shellcode+93> sub al,0x40
0x600a9f <shellcode+95> xor rcx,rcx
> 0x600aa2 <shellcode+98> cmp BYTE PTR [rsp+rcx*1],al
0x600aa5 <shellcode+101> je 0x600aac <shellcode+108>
0x600aa7 <shellcode+103> inc rcx
0x600aaa <shellcode+106> jmp 0x600aa2 <shellcode+98>
0x600aac <shellcode+108> cmp rcx,0x1d
0x600ab0 <shellcode+112> jne 0x600b3f <shellcode+255>

```

Realiza la comparación de si el caracter leído es un salto de línea (shellcode+98) y si son 29 caracteres (shellcode+108), continuará con la ejecución del programa.

Dado que el serial es mayor de 29 caracteres, reduje la cantidad para poder ver que hacía después el programa.

- El serial (password) quedó; 87654-32109-87654-321DRO-WSSA

```

[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
87654-32109-87654-321DRO-WSSA

```

- En shellcode+121 y shellcode+124 se observa que las cadenas a probar deben tener el mismo formato del serial dado, es decir, 5 caracteres seguidos de un - para poder continuar con la ejecución.

```

rax      0xa      10
rcx      0x5      5
rsi user@par 0x7fffffff400 140737488348160
rbp 0x7fffffff480 0x7fffffff480
r8 0x7ffff7f9a500 140737353721088
r10 0x601010 6295568
r12 0x400410 4195344
r14 0x0 0
rip 0x600abc 0x600abc <shellcode+124>
cs 0x33 51
ds 0x0 0
fs 0x0 0
LIBC_2.2.5

0x600a9f <shellcode+95> xor rcx,rcx
0x600aa2 <shellcode+98> cmp BYTE PTR [rsp+rcx*1],al
0x600aa5 <shellcode+101> je 0x600aac <shellcode+108>
0x600aa7 <shellcode+103> inc rcx
0x600aaa <shellcode+106> jmp 0x600aa2 <shellcode+98>
0x600aac <shellcode+108> cmp rcx,0x1d
0x600ab0 <shellcode+112> jne 0x600b3f <shellcode+255>
0x600ab6 <shellcode+118> xor rcx,rcx
0x600ab9 <shellcode+121> add cl,0x5
0x600abc <shellcode+124> cmp BYTE PTR [rsp+rcx*1],0x2d
0x600ac0 <shellcode+128> jne 0x600b3f <shellcode+255>
0x600ac2 <shellcode+130> add cl,0x6
0x600ac5 <shellcode+133> cmp cl,0x11

```

- Ahora se toma cada caracter y se va sumando su valor hexadecimal o decimal de cada uno para compararse con 0x8e0 (2272) y en caso de ser iguales, continúa con la ejecución del programa.
rcx = a los primeros 28 caracteres y en shellcode+160 se agrega el último caracter faltante para ser los 29.


```

Register group: general
rax      0xa      10
rcx      0x1c     28
rsi user@par 0x7fffffff400 140737488348160
rbp 0x7fffffff400 0x7fffffff400
r8 0x7ffff7f9a500 140737353721088
r10 0x601010 6295568
r12 0x400410 4195344
r14 0x0 0
rip 0x600acf 0x600acf <shellcode+143>
cs 0x33 51
ds 0x0 0
fs 0x0 0

```

```

Register group: general
rax      0x688
rcx      0x0
rsi user@par 0x7fffffff
rbp 0x7fffffff
r8 0x7ffff7f9a500
r10 0x601010
r12 0x400410
r14 0x0
rip 0x600ae6
cs 0x33
ds 0x0
fs 0x0

```

```

> 0x600acf <shellcode+143> xor rax,rax
0x600ad2 <shellcode+146> xor rbx,rbx
0x600ad5 <shellcode+149> mov bl,BYTE PTR [rsp+rcx*1]
0x600ad8 <shellcode+152> add rax,rbx
0x600adb <shellcode+155> loop 0x600ad2 <shellcode+146>
0x600add <shellcode+157> xor rbx,rbx
0x600ae0 <shellcode+160> mov bl,BYTE PTR [rsp+rcx*1]
0x600ae3 <shellcode+163> add rax,rbx
0x600ae6 <shellcode+166> cmp rax,0x8e0
0x600aec <shellcode+172> jne 0x600b3f <shellcode+255>
0x600aee <shellcode+174> lea rdx,[rsp+0xc]
0x600af3 <shellcode+179> xor rcx,rcx
0x600af6 <shellcode+182> mov cl,0x5

```

```

0x600ac8 <shellcode+144>
0x600aca <shellcode+146>
0x600acd <shellcode+148>
0x600acf <shellcode+150>
0x600ad2 <shellcode+152>
0x600ad5 <shellcode+154>
0x600ad8 <shellcode+156>
0x600adb <shellcode+158>
0x600add <shellcode+160>
0x600ae0 <shellcode+162>
0x600ae3 <shellcode+164>
> 0x600ae6 <shellcode+166>
0x600aec <shellcode+172>

```

Como la suma del password dado no es 2272, genero otra ya con mi nombre y los caracteres faltantes para cumplir la condición y seguir con la ejecución.

En shellcode+187 y shellcode+193 compara los caracteres a y A, respectivamente, por lo que, en un principio creí que en la cadena a generar deben estar presentes.

Al realizar pruebas, noté que esa función se ejecuta pero aunque no se cumpla la condición, ejecuta la instrucción siguiente (shellcode+212).

Por lo que únicamente

es necesario que la suma sea 2272.

```

0x600ae3 <shellcode+163> add rax,rbx
0x600ae6 <shellcode+166> cmp rax,0x8e0
0x600aec <shellcode+172> jne 0x600b3f <shellcode+255>
0x600aee <shellcode+174> lea rdx,[rsp+0xc]
0x600af3 <shellcode+179> xor rcx,rcx
0x600af6 <shellcode+182> mov cl,0x5
0x600af8 <shellcode+184> xor rax,rax
0x600afb <shellcode+187> test BYTE PTR [rdx+rcx*1],0x41
0x600aff <shellcode+191> jne 0x600b0a <shellcode+202>
0x600b01 <shellcode+193> test BYTE PTR [rdx+rcx*1],0x61
0x600b05 <shellcode+197> jne 0x600b0a <shellcode+202>
0x600b07 <shellcode+199> inc rax
0x600b0a <shellcode+202> loop 0x600afb <shellcode+187>
0x600b0c <shellcode+204> test rax,0x3
0x600b12 <shellcode+210> jne 0x600b3f <shellcode+255>

```

```

0x600af8 <shellcode+184>
0x600afb <shellcode+187>
0x600aff <shellcode+191>
0x600b01 <shellcode+193>
0x600b05 <shellcode+197>
0x600b07 <shellcode+199>
0x600b0a <shellcode+202>
0x600b0c <shellcode+204>
0x600b12 <shellcode+210>
0x600b14 <shellcode+212>
0x600b17 <shellcode+215>
0x600b1a <shellcode+218>
0x600b1d <shellcode+221>
0x600b20 <shellcode+224>

```

- La cadena generada, que cumple por ahora la condición solicitada (fue generada con ayuda de un script sencillo para saber el valor de la cadena y agregar los caracteres que elja manualmente) es:
rafae-l.val-lejo.-*+456-7r95m

```

[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
rafae-l.val-lejo.-*+456-7r95m

```

- Una vez que se cumple la suma igual a 2272, en shellcode+ se toma el primer caracter, del tercer bloque (lejo.) y se compara con a y en caso de ser igual, avanza y compara el siguiente caracter con A, en caso de ser igual, continua con la ejecución del programa (ciclo que se salta y solamente es necesaria la suma de 2272) y termina de recorrer la cadena que ya no debe cumplir otra condición extra.

```

Register group: general
rax      0x0      0
rcx      0x1      1
rsi user@pa 0x7fffffff400 140737488348160
rbp      0x7fffffff480 0x7fffffff480
r8       0x7ffff7f9a500 140737353721088
r10 b64/ld- 0x601010 6295568
r12 c.so.6 0x400410 4195344
r14      0x0      0
rip      0x600afb 0x600afb <shellcode+187>
cs libc.sta 0x33 main 51
ds gmon.sta 0x0      0
fs       0x0      0
libc 2.2.5
File Edit View Search Termin
> 0x600afb <shellcode+187> test BYTE PTR [rdx+rcx*1],0x41
0x600aff <shellcode+191> jne 0x600b0a <shellcode+202>
0x600b01 <shellcode+193> test BYTE PTR [rdx+rcx*1],0x61
0x600b05 <shellcode+197> jne 0x600b0a <shellcode+202>
0x600b07 <shellcode+199> inc rax
0x600b0a <shellcode+202> loop 0x600afb <shellcode+187>
0x600b0c <shellcode+204> test rax,0x3
0x600b12 <shellcode+210> jne 0x600b3f <shellcode+255>
0x600b14 <shellcode+212> xor rax,rax
0x600b17 <shellcode+215> xor rbx,rbx
0x600b1a <shellcode+218> xor rcx,rcx
0x600b1d <shellcode+221> xor rdx,rdx
0x600b20 <shellcode+224> push 0x3

```

```

0x600afb <shellcode+187>
0x600afb <shellcode+191>
0x600aff <shellcode+193>
0x600b01 <shellcode+197>
0x600b05 <shellcode+199>
0x600b07 <shellcode+202>
0x600b0a <shellcode+204>
0x600b0c <shellcode+206>
0x600b12 <shellcode+210>
0x600b14 <shellcode+212>
0x600b17 <shellcode+215>
0x600b1a <shellcode+218>
0x600b1d <shellcode+221>
0x600b20 <shellcode+224>
native process 21993 In
0x000000000000600aff in
0x000000000000600b0a in
0x000000000000600afb in
0x000000000000600aff in
0x000000000000600b0a in
0x000000000000600b0c in
0x000000000000600b12 in
0x000000000000600b14 in
(gdb) si
0x000000000000600b17 in
(gdb) layout asm
(gdb) layout regs
(gdb) x/s $rdx
0x7fffffff40c: "lejo."
(gdb)

```

- Por lo tanto, la cadena (serial-password) obtenida que cumple las condiciones es:

rafae-l.val-lejo.-*+456-7r95m

```

[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
rafae-l.val-lejo.-*+456-7r95m

```

```

Register group: general
rax      0x3b      59
rcx      0x600b29  6294313
rsi user@parrot-[~/Downloads/Examen]
rbp      0x7fffffff480 0x7fffffff480
r8       0x7ffff7f9a500 140737353721088
r10      0x601010 6295568
r12      0x400410 4195344
r14      0x0      0
rip      0x600b3d 0x600b3d <shellcode+253>
cs       0x33      51
ds       0x0      0
fs       0x0      0
LIBC_2.2.5
File Edit View Search Terminal Help
[x]-[user@parrot]-[~/Downloads
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?)
rafae-l.val-lejo.-*+4-567qv
0x600b27 <shellcode+231> syscall
0x600b29 <shellcode+233> jne 0x600b23 <shellcode+227>
0x600b2b <shellcode+235> push rsi
0x600b2c <shellcode+236> pop rdx
0x600b2d <shellcode+237> push rsi
0x600b2e <shellcode+238> movabs rbx,0x68732f2f6e69622f
0x600b38 <shellcode+248> push rbx
0x600b39 <shellcode+249> push rsp
0x600b3a <shellcode+250> pop rdi
0x600b3b <shellcode+251> mov al,0x3b
0x600b3d <shellcode+253> syscall
0x600b3f <shellcode+255> or al,BYTE PTR [rax]
0x600b41 <completed.6661> add BYTE PTR [rax],al
0x600b43 add BYTE PTR [rax],al
0x600b45 add BYTE PTR [rax],al

```

- Realiza entonces la llamada al sistema `execve` (`shellcode+253`) y se obtiene una shell mediante el socket al introducirse el serial (password) correcto que cumple las condiciones del programa:
SHELLow es un programa que da una bind shell y espera una conexión con la contraseña correcta (que cumple las condiciones del programa) para devolverle un shell al atacante.

```

0x600b3a <shellcode+250>    pop     rdi
0x600b3b <shellcode+251>    mov     al,0x3b
0x600b3d <shellcode+253>    syscall
0x600b3f <shellcode+255>    or      al,BYTE PTR [rax]
0x600b41 <completed.6661>    add     BYTE PTR [rax],al
0x600b43                add     BYTE PTR [rax],al
0x600b45                add     BYTE PTR [rax],al

```

native process 21993 In: shellcode

0x0000000000600b3b in shellcode ()

0x0000000000600b3d in shellcode ()

process 21993 is executing new program: /bin/dash

warning: Cannot access memory at address 0x600b3d

Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:

Cannot access memory at address 0x600b3d

Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:

Cannot access memory at address 0x600b3d

Error while reading shared library symbols for /lib64/ld-linux-x86-64.so.2:

Cannot access memory at address 0x600b3d

Error while reading shared library symbols for /lib/x86_64-linux-gnu/libc.so.6:

Cannot access memory at address 0x600b3d

[Detaching after fork from child process 22573]

[Detaching after fork from child process 22576]

• Ejecución sin gdb:

```

[~]-[root@parrot]-[/home/user/Downloads/Examen]
#./shell_funciona
Baia, baia : si que has llegado lejos
It's time to crackme Miss/Mr Reverse Enginner ;)
lib64/ld-linux-x86-64.so.2
libc.so.6
puts
libc start main
gmon start
GLIBC 2.2.5
H-H
ffff
baia, bah
la ... sh
que bah

```

```

[user@parrot]-[~/Downloads/Examen]
$nc localhost 39321 -v
localhost [127.0.0.1] 39321 (?) open
rafae-l.val-lejo.-*+456-7r95m
ls
SHELLow
shell_funciona
shell_funciona_stripped
shell_mod2
whoami
root

```