



Análisis de Vulnerabilidades

# Examen.



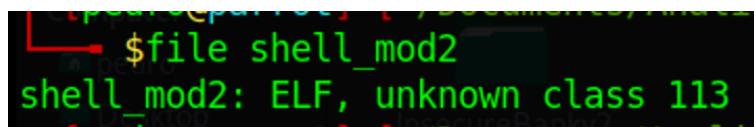
Rodríguez Gallardo Pedro Alejandro

En este documento se explicara el proceso que se siguió para la resolución del examen del curso.

En primera instancia con el comando file se inspeccionó el tipo de archivo con el que contábamos, para lo cual encontramos que era un archivo comprimido gzip, para lo cual con ayuda del siguiente comando descomprimimos el archivo.

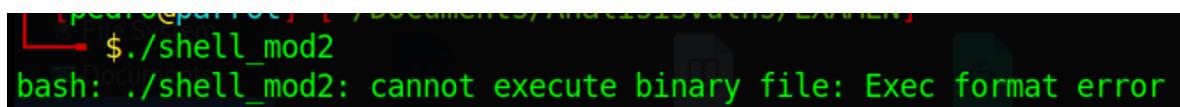
```
tar -xzf <archivo>
```

Al realizar una nueva inspección para conocer el tipo de archivo, nos encontramos que es un tipo de archivo ELF, pero nos muestra que tiene clases sin reconocer, además que si tratamos de ejecutar el binario este no marca error.



```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $file shell_mod2
    shell_mod2: ELF, unknown class 113
```

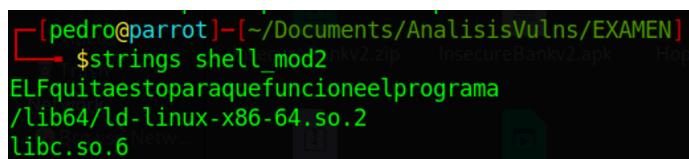
*Tipo de archivo.*



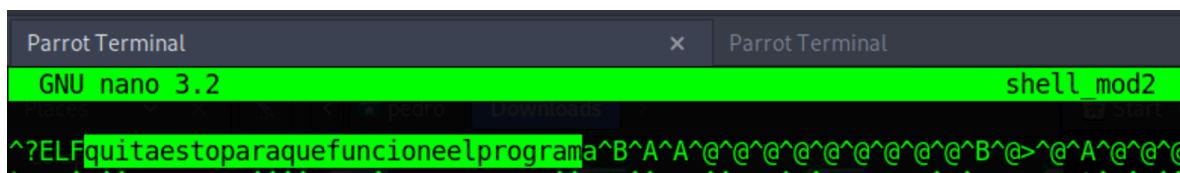
```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $./shell_mod2
    bash: ./shell_mod2: cannot execute binary file: Exec format error
```

*Error al ejecutar.*

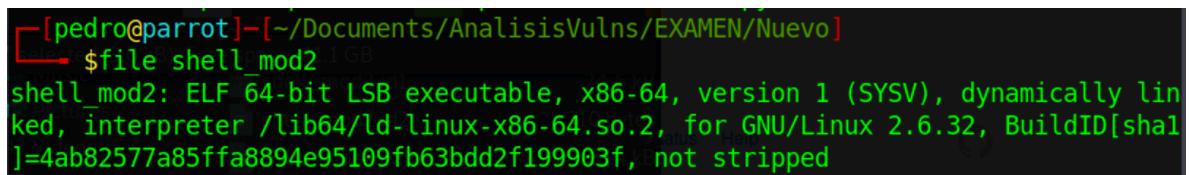
Al revisar el archivo encontramos con strings y con nano encontramos una leyenda interesante que debemos quitar para continuar.



```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $strings shell_mod2
    ELFquitaestoparaquefuncioneelprograma
    /lib64/ld-linux-x86-64.so.2
    libc.so.6
```



Una vez que quitamos la cadena obtenemos el siguiente resultado.



```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN/Nuevo]
└─ $file shell_mod2
    shell_mod2: 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]=4ab82577a85ffa8894e95109fb63bdd2f199903f, not stripped
```

*Salida del comando file.*

Ahora sin ningún problema podemos ejecutar nuestro binario.

```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $ ./shell_mod2
Baia, baia ... si que has llegado lejos
It's time to crackme Miss/Mr Reverse Enginner ;)
210 bytes
```

Binario en ejecución.

Con el comando strings encontramos algunas otras cadenas interesantes como estas;

```
erse EngH
inner ;)H
JAVAJA^A_
***35"
07654-32109-87654-321DRO-WSSAP
SHELLow was here :P
8)_b
6[34
(B;H-
Nuevolve.py 8:50
```

Me di cuenta que si invertimos la primera cadena esta comienza con la palabra PASSWORD

```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $echo "87654-32109-87654-321DRO-WSSAP" | rev
PASSW-ORD123-45678-90123-45678
```

Decidi probar ambas cadenas en el programa, una por una pero no obtuve ningun resultado, lo que si note es que el programa se queda en espera, tal cual como un socket.

```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN]
└─ $./shell_mod2
Baia, baia ... si que has llegado lejos
It's time to crackme Miss/Mr Reverse Enginner ;)
87654-32109-87654-321DRO-WSSAP
PASSW-ORD123-45678-90123-45678
Nuevolve.py 8:50
```

Asi que decidi verificar si tenia algun puerto en escucha, par mi sorpresa si se crea uno el numero 39321.

```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN/Nuevo]
└─ $netstat -lptun
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State      PID/Program name
tcp        0      0 0.0.0.0:8008             0.0.0.0:*              LISTEN    16237/[login]
tcp        0      0 0.0.0.0:39321            0.0.0.0:*              LISTEN    19452./shell_mod2
```

Puerto en escucha 39321.

Al realizar una conexión con netcat, probe las contraseña que había encontrado en ambos formatos, con ambas me da un segmentation fault.

```
[x]-[pedro@parrot]-[~/Documents/]  
└─ $nc 127.0.0.1 39321  
87654-32109-87654-321DRO-WSSAP
```

## Cliente

```
[pedro@parrot] - [~/Documents/Analisis Network]
└─ $ ./shell_mod2
Baia, baia ... si que has llegado lejos
It's time to crackme Miss/Mr Reverse Eng
87654-32109-87654-321DRO-WSSAP
PASSW-ORD123-45678-90123-45678
Segmentation fault
```

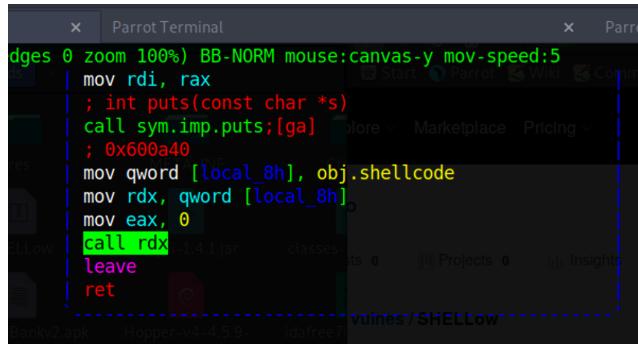
## Socket

Con ayuda del comando strace pude ver las llamadas al sistema que realiza nuestro binario podiendo ver como es que se crea el socket.

```
[x]-[pedro@parrot]-(~/Documents/AnalisisVulns/EXAMEN/Nuevo]
└ $strace ./shell_mod2
execve("./shell_mod2", ["/etc/ld.so.preload"], 0xfffffffffe1f0 /* 43 vars */) = 0
brk(NULL) = 0x601000
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=188056, ...}) = 0
mmap(NULL, 188056, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ffff7fa2000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\3\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\260A\2\0\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) = 0x7fff7fa0000
mmap(NULL, 1837056, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ffff7ddf000
mprotect(0x7ffff7e01000, 1658880, PROT_NONE) = 0
mmap(0x7ffff7e01000, 1343488, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7ffff7e01000
mmap(0x7ffff7f49000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16a000) = 0x7ffff7f49000
mmap(0x7ffff7f96000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b6000) = 0x7ffff7f96000
mmap(0x7ffff7f9c000, 14336, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ffff7f9c000
close(3) = 0
arch_prctl(ARCH_SET_FS, 0x7ffff7fa1500) = 0
mprotect(0x7ffff7f96000, 16384, PROT_READ) = 0
mprotect(0x7ffff7ffc000, 4096, PROT_READ) = 0
munmap(0x7ffff7fa2000, 188056) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
brk(NULL) = 0x601000
brk(0x622000) = 0x622000
write(1, "Baia, baia ... si que has llegad...", 40) = 40
write(1, "It's time to crackme Miss/Mr Rev...", 49) = 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) = 4

```

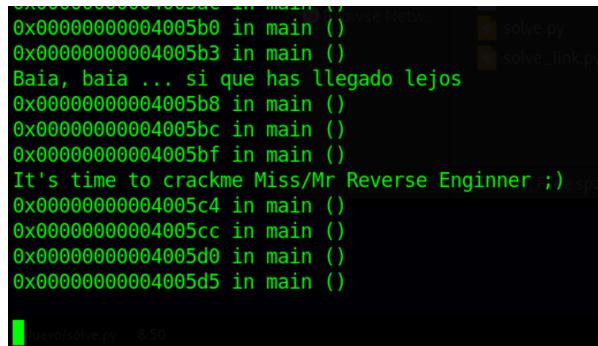
Con ayuda de radare2 vi el flujo del programa dentro de la función main *s main*, en el cual encontre que realiza una llamada al registro rdx.



The screenshot shows the Hopper Disassembler interface with the assembly code for the main function. The code includes instructions like mov rdi, rax; int puts(const char \*s); call sym.imp.puts@[ga]; mov qword [local\_8h], obj.shellcode; mov rdx, qword [local\_8h]; mov eax, 0; call rdx; leave; ret. A blue dashed box highlights the 'call rdx' instruction.

```
0x00000000004005b0 in main ()      ; BB-NORM mouse:canvas-y mov-speed:5
    mov rdi, rax
    ; int puts(const char *s)
    call sym.imp.puts@[ga]
    ; 0x600a40
    mov qword [local_8h], obj.shellcode
    mov rdx, qword [local_8h]
    mov eax, 0
    call rdx
    leave
    ret
```

La cual va despues de imprimir las cadenas de un inicio y colocar el objeto shellcode. Para lo cual procedi a revisar el binario con gdb paso a paso, unicamente creando una Break en main. Para ver el comportamiento despues de imprimir las cadenas.



The terminal window shows the program's output. It prints several lines of text, including "Baia, baia ... si que has llegado lejos" and "It's time to crackme Miss/Mr Reverse Enginner ;)", followed by a series of memory addresses. The window title is "solve.py".

```
0x00000000004005b0 in main ()      ; BB-NORM mouse:canvas-y mov-speed:5
0x00000000004005b3 in main ()
Baia, baia ... si que has llegado lejos
0x00000000004005b8 in main ()
0x00000000004005bc in main ()
0x00000000004005bf in main ()
It's time to crackme Miss/Mr Reverse Enginner ;)
0x00000000004005c4 in main ()
0x00000000004005cc in main ()
0x00000000004005d0 in main ()
0x00000000004005d5 in main ()
```

Despues de imprimir las cadenas, se crea el socket y queda en espera de la conexión, ademas se queda esperando algo mas asi que envio la cadena en la conexión, es cuando entra a shellcode y se cierra mostrando un segmentation fault.

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000600b3f in shellcode ()

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
The program is not being run.
```

Decidi ir paso a paso en esta sección, y me di cuenta que hace varias inspecciones sobre la cadena que se le envia al socket.

### 1.- La cadena debe tener una condición de 29 caracteres

En el registro RCX se realiza un conteo de caracteres el cual va incrementando por cada carácter encontrado, al tener el numero total de caracteres realiza una comparación con 0x1d que es el número 29 en decimal.

The screenshot shows the assembly and registers windows of the IDA Pro debugger. The assembly window displays the following code snippet:

```
0x400596 <main+144>    mov    %rax, -0x50(%rbp)
0x40059a <main+148>    movabs $0x293b2072656e6e69,%rax
0x600a78 <shellcode+56> pop    %rdx
0x600a9b <shellcode+91>    mov    $0x4a,%al
0x600a9d <shellcode+93>    sub    $0x40,%alt>
0x600a9f <shellcode+95>    xor    %rcx,%rcx
0x600aa2 <shellcode+98>    cmp    %al,(%rsp,%rcx,1)
0x600aa5 <shellcode+101>   je     0x600aac <shellcode+108>
>0x600aa7 <shellcode+103> inc    %rcx
0x600aaa <shellcode+106>   jmp    0x600aa2 <shellcode+98>
0x600aac <shellcode+108>   cmp    $0x1d,%rcx
0x600ab0 <shellcode+112>   jne    0x600b3f <shellcode+255>
0x600ab6 <shellcode+118>   xor    %rcx,%rcx
native600ab9 <shellcode+121> add    $0x5,%cl
0x00000000bc400570 in main24> cmpb   $0x2d,(%rsp,%rcx,1)
0x0000000000000000 in main24>
```

### 2.- La cadena debe tener cada 5 caracteres el símbolo –

En el registro rcx se inicializa en 0 y se le va sumando 6, aquí se compara que el carácter en esta posición sea – , por lo que compara en las posiciones 5, 11,17 y 23.



rax 0x0 0 rdx

rax library function 0xa user function Instr 10 Data Unexplored External symbols rdx

rcx 0x17 23 rdi

r8 0x7ffff7fa1500 140737353749760 rsi View-1

r14 0x0 0 rbp

rip 0x4005b3 0x4005b3 <main+173> r11

cs 0x33 f 51 000000000000000085> Segment permission r12

ds\_puts 0x600a85 0 600a85 <shellcode+69> seg r13

fs 0x0 96 0 init\_96 00000004003A8 86> eflags

-c8 -c8 -136> eflags

0x400596 <main+144> mov %rax,-0x50(%rbp)3A8  
0x40059a <main+148> movabs \$0x293b2072656e6e69,%rax  
0x600a78 <shellcode+56> pop %rdx :0000000004003A8 \_init\_proc publ  
0x600a88 <shellcode+72> pop %rdi :0000000004003A8 proc  
0x600a99 <shellcode+121> add %add :0000000004003A8 sub  
0x600abc <shellcode+124> cmpb \$0x2d,(%rsp,%rcx,1) test  
0x600ac0 <shellcode+128> jne 0x600b3f <shellcode+255> all  
0x600aC2 <shellcode+130> add \$0x6,%cl3A8 loc\_4003BD:  
0x600ac5 <shellcode+133> cmp \$0x11,%cl add  
0x600ac8 <shellcode+136> jbe 0x600abc <shellcode+124> endp  
0x600aca <shellcode+138> xor %rcx,%rcx:\_init ends  
0x600ad <shellcode+141> mov \$0x1c,%cl  
0x600acf <shellcode+143> xor %rax,%rax  
0x600ad2 <shellcode+146> xor %rbx,%rbx Segment type: Pure  
0x600ad5 <shellcode+149> mov (%rsp,%rcx,1),%bl permissions  
0x600ad8 <shellcode+152> add %rbx,%rax segme  
0x0000000000000000600a74 in shellcode () assme  
0x0000000000000000600a85 in shellcode () org  
(gdb) si align  
0x0000000000000000600a8c in shellcode () LOAD ends  
0x0000000000000000600abc in shellcode () pit  
0x0000000000000000600ac0 in shellcode () pit  
0x0000000000000000600ac2 in shellcode () pit  
0x0000000000000000600ac5 in shellcode () pit  
0x0000000000000000600a8c in shellcode () pit  
0x0000000000000000600abc in shellcode () pit  
0x0000000000000000600ac0 in shellcode () pit  
0x0000000000000000600ac2 in shellcode () pit  
0x0000000000000000600ac5 in shellcode () pit  
0x0000000000000000600ac8 in shellcode () pit  
(gdb)

Comparacion en la posicion 23, con 0x2d = “-“

3.- La suma de los caracteres en ASCII debe ser exactamente de 2272.

En el registro RBX se tiene el valor en ASCII en decimal del carácter, la posición del carácter la tiene RCX, mientras que la suma se va realizando en RAX, en la captura anterior se muestra la suma del último carácter. Una vez que se tiene la suma total se compara con el valor 0x8E0 que en decimal es 2272.



Para obtener la suma de una cadena válida me ayude de python para sumar el valor de cada valor.

```
>>> sum(bytearray("pedro-rodri-guez0-00000-000FA"))
2272
```

Al enviar esta cadena válida se logra obtener la shell en el sistema por medio del socket.

```
[pedro@parrot] -[~/Documents/AnalisisVulns/EXAMEN/Nuevo]
└─$ nc 127.0.0.1 39321
pedro-rodri-guez0-00000-000FA
id
uid=1000(pedro) gid=1000(pedro) groups=1000(pedro),20(dialout),24(cdrom),25(floppy),27(sudo),
etdev),112(debian-tor),122(bluetooth),139(scanner)
```