Day 17 - ReverseELFneering

Scenario

> Luckily for us, everything we need has been provided to you via an Instance that you can deploy and log into:
>
> 1. Press the "Deploy" button on the top-right of this task
> 2. Wait for the IP address of the target Instance to display
> 3. Log into your Instance using the following information:
>
> **IP Address:** `MACHINE_IP`
>
> **Username:** elfmceager
>
> **Password:** adventofcyber

> 6. Challenge
>
> Use your new-found knowledge of Radare2 to **analyse the "challenge1" file** in the Instance `MACHINE_IP` that is attached to this task to answer the questions below.

this challenge will be more to using radare2 to perform reverse engineering on a binary

let's login into the remote host using ssh

```
┌──(nobodyatall㉿0×DEADBEEF)-[~]
└─$ ssh elfmceager@10.10.31.80
The authenticity of host '10.10.31.80 (10.10.31.80)' can't be established.
ECDSA key fingerprint is SHA256:XrBuXSQs0wRKhvVRdrSfE/0F5ccAZQiXAhMhzB1dV7U.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.31.80' (ECDSA) to the list of known hosts.
elfmceager@10.10.31.80's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-128-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Dec 18 06:59:47 UTC 2020

  System load:  0.09              Processes:           103
  Usage of /:   40.6% of 11.75GB  Users logged in:     0
  Memory usage: 17%               IP address for ens5: 10.10.31.80
  Swap usage:   0%


0 packages can be updated.
0 updates are security updates.


Last login: Wed Dec 16 18:25:51 2020 from 192.168.190.1
elfmceager@tbfc-day-17:~$
```

here's the challenge binary for us to solve

```
-rwxr-xr-x 1 elfmceager elfmceager 844648 Dec 16 18:19 challenge1
-rwxr-xr-x 1 elfmceager elfmceager 844736 Dec 16 18:19 file1
drwx------ 3 elfmceager elfmceager   4096 Dec 16 18:06 .gnupg
```

challenge1 is a elf 64bit binary

```
elfmceager@tbfc-day-17:~$ file challenge1
challenge1: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0, BuildID[sha1]=884f57a67cddb0fc0104f1d556ab05118395232
4, not stripped
elfmceager@tbfc-day-17:~$
```

execute challenge1 but nothing happened

```
elfmceager@tbfc-day-17:~$ ./challenge1
elfmceager@tbfc-day-17:~$ ./challenge1 asd
elfmceager@tbfc-day-17:~$
```

start radare2 to reverse engineer the binary

```
elfmceager@tbfc-day-17:~$ r2 -d ./challenge1
Process with PID 1452 started...
= attach 1452 1452
bin.baddr 0x00400000
Using 0x400000
Warning: Cannot initialize dynamic strings
asm.bits 64
```

analyze the binary using aa

```
[0x00400a30]> aa
[ WARNING : block size exceeding max block size at 0x006ba220
[+] Try changing it with e anal.bb.maxsize
 WARNING : block size exceeding max block size at 0x006bc860
[+] Try changing it with e anal.bb.maxsize
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00400a30]>
```

find the main function by using afl & it's in the address 0x00400b4d

```
[0x00400a30]> afl | grep main
0x00400b4d    1 35           sym.main
```

using pdf to disassembler the main function

```
[0x00400a30]> pdf @main
            ;-- main:
/ (fcn) sym.main 35
|   sym.main ();
|           ; var int local_ch @ rbp-0xc
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x00400a4d (entry0)
|           0x00400b4d      55              push rbp
|           0x00400b4e      4889e5          mov rbp, rsp
|           0x00400b51      c745f4010000.   mov dword [local_ch], 1
|           0x00400b58      c745f8060000.   mov dword [local_8h], 6
|           0x00400b5f      8b45f4          mov eax, dword [local_ch]
|           0x00400b62      0faf45f8        imul eax, dword [local_8h]
|           0x00400b66      8945fc          mov dword [local_4h], eax
|           0x00400b69      b800000000      mov eax, 0
|           0x00400b6e      5d              pop rbp
\           0x00400b6f      c3              ret
[0x00400a30]>
```

set the breakpoint on the main function address using db
& continue execute until it hit the breakpoint using dc

```
[0x00400a30]> db 0x00400b4d
[0x00400a30]> dc
hit breakpoint at: 400b4d
```

checking pdf & we're now in the main function

```
[0x00400b4d]> pdf
            ;-- main:
            ;-- rax:
            ;-- rip:
/ (fcn) sym.main 35
|   sym.main ();
|           ; var int local_ch @ rbp-0x0c
|           ; var int local_8h @ rbp-0x8
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x00400a4d (entry0)
|           0x00400b4d b    55              push rbp
|           0x00400b4e      4889e5          mov rbp, rsp
|           0x00400b51      c745f4010000.   mov dword [local_ch], 1
|           0x00400b58      c745f8060000.   mov dword [local_8h], 6
|           0x00400b5f      8b45f4          mov eax, dword [local_ch]
|           0x00400b62      0faf45f8        imul eax, dword [local_8h]
|           0x00400b66      8945fc          mov dword [local_4h], eax
|           0x00400b69      b800000000      mov eax, 0
|           0x00400b6e      5d              pop rbp
\           0x00400b6f      c3              ret
```

these will be the memory address the variable value stored

```
  ; var int local_ch @ rbp-0x0c
  ; var int local_8h @ rbp-0x8
  ; var int local_4h @ rbp-0x4
```

if we can see the 1st mov assembly it will assign the value 1 into local_ch variable

```
 0x00400b51      c745f4010000.   mov dword [local_ch], 1
 ;-- rip:
```

Question: What is the value of **local_ch** when its corresponding movl instruction is called (first if multiple)?
-1

use ds to step to next instruction

now we step over the imul instruction

```
    0x00400b62      0faf45f8        imul eax, dword [local_8h]
    ;-- rip:
    0x00400b66      8945fc          mov dword [local_4h], eax
```

& check the value in rax register, it's 6 now

```
[0x00400b4d]> dr
rax = 0x00000006
rbx = 0x00400400
rcx = 0x0044b9a0
rdx = 0x7ffde61b49f8
```

Question: What is the value of **eax** when the imull instruction is called?
-6

now step until the mov eax, 0 instruction (if we're stepping on it, it haven't execute the line of assembly code yet

```
   0x00400b66      8945fc          mov dword [local
   ;-- rip:
   0x00400b69      b800000000      mov eax, 0
   0x00400b6e      5d              pop rbp
   0x00400b6f      c3              ret
```

now check the local_4h value, & the value is 6

```
[0x00400b4d]> px @rbp-0x4
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffde61b48bc  0600 0000 4018 4000 0000 0000 e910 4000  ....@.@.......@.
0x7ffde61b48cc  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffde61b48dc  0100 0000 e849 1be6 fd7f 0000 4d0b 4000  .....I......M.@.
0x7ffde61b48ec  0000 0000 0000 0000 0000 0000 0600 0000  ..............
```