

Day 17 - [Reverse Engineering] ReverseELFneering

Tool Used: Kali Linux, firefox, Nmap, radare2

Solution/walkthrough:

Q1

3. Register me this, register me that...

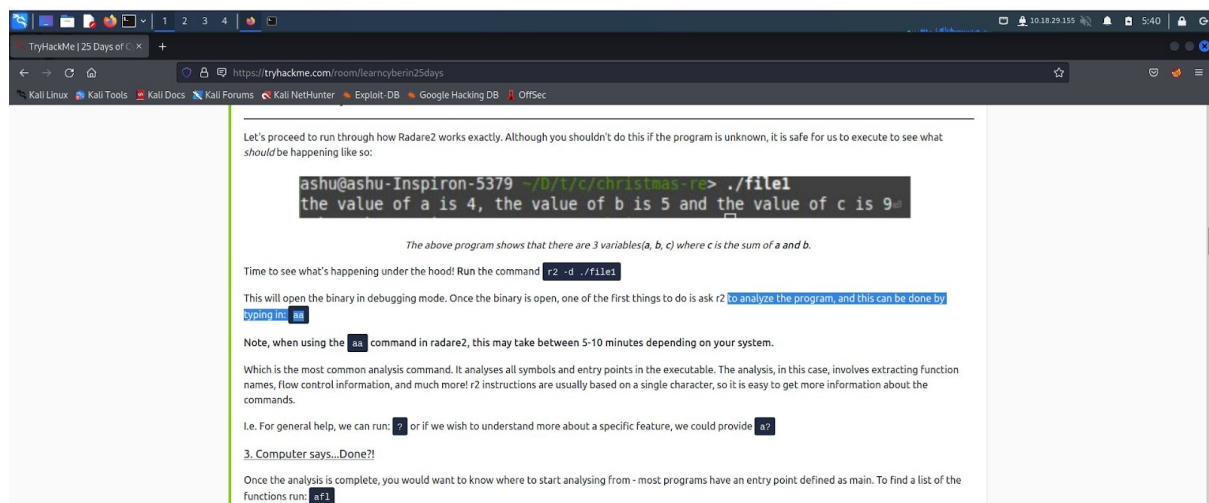
The core of assembly language involves using registers to do the following:

- Transfer data between memory and register, and vice versa
- Perform arithmetic operations on registers and data
- Transfer control to other parts of the program Since the architecture is x86-64, the registers are 64 bit and Intel has a list of 16 registers:

Initial Data Type	Suffix	Size (bytes)
Byte	b	1
Word	w	2
Double Word	l	4
Quad	q	8
Single Precision	s	4
Double Precision	l	8

study from Try Hack Me.

Q2



study from Try Hack Me.

Q3

The line starting with `sym.main` indicates we're looking at the main function. The next 3 lines are used to represent the variables stored in the function. The second column indicates that they are integers(`int`), the 3rd column specifies the name that `r2` uses to reference them and the 4th column shows the actual memory location.

The first 3 instructions are used to allocate space on that stack (ensures that there's enough room for variables to be allocated and more). We'll start looking at the program from the 4th instruction (`movl $4`). We want to analyse the program while it runs and the best way to do this is by using breakpoints.

A breakpoint specifies where the program should stop executing. This is useful as it allows us to look at the state of the program at that particular point. So let's set a breakpoint using the command `bp` in this case, it would be `bp 0x00400b55`. To ensure the breakpoint is set, we run the `pdf @main` command again and see a little `b` next to the instruction we want to stop at.

study from Try Hack Me.

Q4

Now that we've set a breakpoint, let's run the program using `dc`

```
[0x00400a30]> dc
hit breakpoint at: 400b55
[0x00400b55]> pdf
;-- main:
;-- rax:
(fcn) sym.main 68
sym.main (int argc, char **argv, char **envp);
; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from entry0 (0x00400a4d)
0x00400b4d 55 pushq %rbp
0x00400b4e 4889e5 movq %rsp, %rbp
0x00400b51 4883ec10 subq $0x10, %rsp
```

Running `dc` will execute the program until we hit the breakpoint. Once we hit the breakpoint and print out the main function, the rip which is the current instruction shows where execution has stopped. From the notes above, we know that the `mov` instruction is used to transfer values. This statement is transferring the value 4 into the `local_ch` variable. To view the contents of the `local_ch` variable, we use the following instruction `px @memory-address`. In this case, the corresponding memory address for `local_ch` will be `rbp-0xc` (from the first few lines of `@pdf main`) This instruction prints the values of memory in hex:

Study from Try Hack Me.

Q5

```
[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
```

What is the value of `local_ch` when it...

it copies value 1 to `[local_ch]`. therefore, the value of `[local_8h]` should be 1.

Q6

when `mov dword [local_8h]` is called, the value of `[local_8h]` becomes 6. when `mov eax, dword [local_ch]` is called, the value of `eax` becomes 1. when `imul` is called, it multiplies the value of `[local_8h]` to `eax`, which is 6×1 . Therefore, the value of `eax` should be 6.

Q7

it copies the value of `eax` to `[local_4h]`. Therefore, the value should be 6.