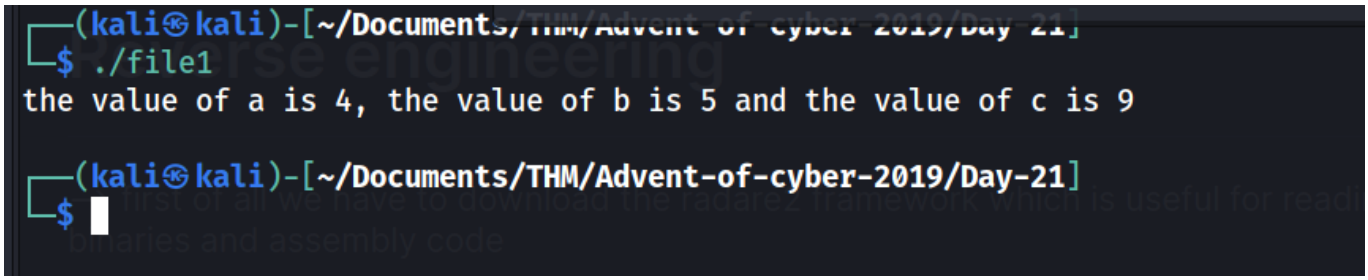
# Reverse engineering

---

→ first of all we have to download the radare2 framework which is useful for reading binaries and assembly code

→ after downloading radare2 first execute the file which we have to run

`./file1`




→ as we can see it's adding a and b and giving the value of c

---

## Radare 2

→ now Let's analyse it using radare2

→ command :

```
r2 -d ./file1
```

```
┌──(kali㉿kali)-[~/Documents/THM/Advent-of-cyber-2019/Day-21]
└─$ r2 -d ./file1
Process with PID 25137 started...
= attach 25137 25137
bin.baddr 0x00400000
Using 0x400000
asm.bits 64
[0x00400a30]> aa
```

→ now first we analyse it by writing `aa`

→ we got the main function using `afl | grep main`

→ here `afl` lists all function but we are interested only in main function

```
[0x00400a30]> afl | grep main
0x00400e10  114 1657          sym.__libc_start_main
0x0048fb30   16 247   -> 237  sym._nl_unload_domain
0x00403b10  308 5366  -> 5301 sym._nl_load_domain
0x00470520    1 49            sym._IO_switch_to_main_wget_area
0x00403870   39 672   -> 640  sym._nl_find_domain
0x00400b4d    1 68            main
0x0048fae0    7 73    -> 69   sym._nl_finddomain_subfreeres
0x0044cf00    1 8             sym._dl_get_dl_main_map
0x00415fe0    1 43            sym._IO_switch_to_main_get_area
```

→ we can see the assembly code for main function using `pdf@main` command where pdf stands for `print disassembly function`

```
[0x00400a30]> pdf@main
        ; DATA XREF from entry0 @ 0x400a4d
┌ 68: int main (int argc, char **argv, char **envp);
        ; var int64_t var_ch @ rbp-0xc
        ; var int64_t var_8h @ rbp-0x8
        ; var int64_t var_4h @ rbp-0x4
        0x00400b4d      55              push rbp
        0x00400b4e      4889e5          mov rbp, rsp
        0x00400b51      4883ec10        sub rsp, 0x10
        0x00400b55      c745f4040000.   mov dword [var_ch], 4
        0x00400b5c      c745f8050000.   mov dword [var_8h], 5
        0x00400b63      8b55f4          mov edx, dword [var_ch]
        0x00400b66      8b45f8          mov eax, dword [var_8h]
        0x00400b69      01d0            add eax, edx
        0x00400b6b      8945fc          mov dword [var_4h], eax
        0x00400b6e      8b4dfc          mov ecx, dword [var_4h]
        0x00400b71      8b55f8          mov edx, dword [var_8h]
        0x00400b74      8b45f4          mov eax, dword [var_ch]
        0x00400b77      89c6            mov esi, eax
        0x00400b79      488d3d881409.   lea rdi, str.the_value_of_a_is__d__the_value_of_b_is__d_and_the_value_of_c_is__d ; 0x492008 ; "the value of a is %d, the value of b is %d and the value of c is %d"
        0x00400b80      b800000000      mov eax, 0
        0x00400b85      e8f6ea0000      call sym.__printf
        0x00400b8a      b800000000      mov eax, 0
        0x00400b8f      c9              leave
        0x00400b90      c3              ret
```

# Breakpoint

→ breakpoint is the place when program stops executing and we can analyse it

→ here we will set the breakpoint at 4th line which is `mov dword [var_ch], 4`

→ to set the breakpoing use `db <address>` in this case `db 0x00400b55`

→ after that do `pdf@main` to check whether break point set or not

→ and we can see the breakpoint is set


→ now we can run the program using `dc` and we can see the program stopped at the breakpoint

---

# Analysing variable value

→ and we will display the main function again

```
[0x00400a30]> dc
hit breakpoint at: 0x400b55
[0x00400b55]> pdf@main
          ; DATA XREF from entry0 @ 0x400a4d
          ;-- rax:
┌ 68: int main (int argc, char **argv, char **envp);
          ; var int64_t var_ch @ rbp-0xc
          ; var int64_t var_8h @ rbp-0x8
          ; var int64_t var_4h @ rbp-0x4
          0x00400b4d      55              push rbp
          0x00400b4e      4889e5          mov rbp, rsp
          0x00400b51      4883ec10        sub rsp, 0x10
          ;-- rip:
          0x00400b55 b    c745f4040000.   mov dword [var_ch], 4
          0x00400b5c      c745f8050000.   mov dword [var_8h], 5
          0x00400b63      8b55f4          mov edx, dword [var_ch]
          0x00400b66      8b45f8          mov eax, dword [var_8h]
          0x00400b69      01d0            add eax, edx
          0x00400b6b      8945fc          mov dword [var_4h], eax
          0x00400b6e      8b4dfc          mov ecx, dword [var_4h]
          0x00400b71      8b55f8          mov edx, dword [var_8h]
          0x00400b74      8b45f4          mov eax, dword [var_ch]
          0x00400b77      89c6            mov esi, eax
          0x00400b79      488d3d881409.   lea rdi, str.the_value_of_a_is__d__the_value_of_b_is__d_and_the_value_of_c_is__d ; 0x492008 ; "the value of a is %d, the value of
b is %d and the value of c is %d"
          0x00400b80      b800000000      mov eax, 0
          0x00400b85      e8f6ea0000      call sym.__printf
          0x00400b8a      b800000000      mov eax, 0
          0x00400b8f      c9              leave
          0x00400b90      c3              ret
[0x00400b55]>
```

→ to see the value of the variable we use the `px@<address>` and in this case we want to see the value of `var_ch` and it's address is `rbp-0xc`

```
[0x00400b55]> px @rbp-0xc
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffcd43cc884  0000 0000 1890 6b00 0000 0000 7018 4000  ......k.....p.@.
0x7ffcd43cc894  0000 0000 1911 4000 0000 0000 0000 0000  ......@.........
0x7ffcd43cc8a4  0000 0000 0000 0000 0100 0000 b8c9 3cd4  ..............<.
0x7ffcd43cc8b4  fc7f 0000 4d0b 4000 0000 0000 0000 0000  ....M.@.........
0x7ffcd43cc8c4  0000 0000 0600 0000 7e00 0000 7000 0000  ........~...p...
0x7ffcd43cc8d4  0500 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffcd43cc8e4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffcd43cc8f4  0000 0000 0000 0000 0000 0000 0004 4000  ..............@.
0x7ffcd43cc904  0000 0000 d599 366c bf16 b0de 1019 4000  ......6l......@.
0x7ffcd43cc914  0000 0000 0000 0000 0000 0000 1890 6b00  ..............k.
```

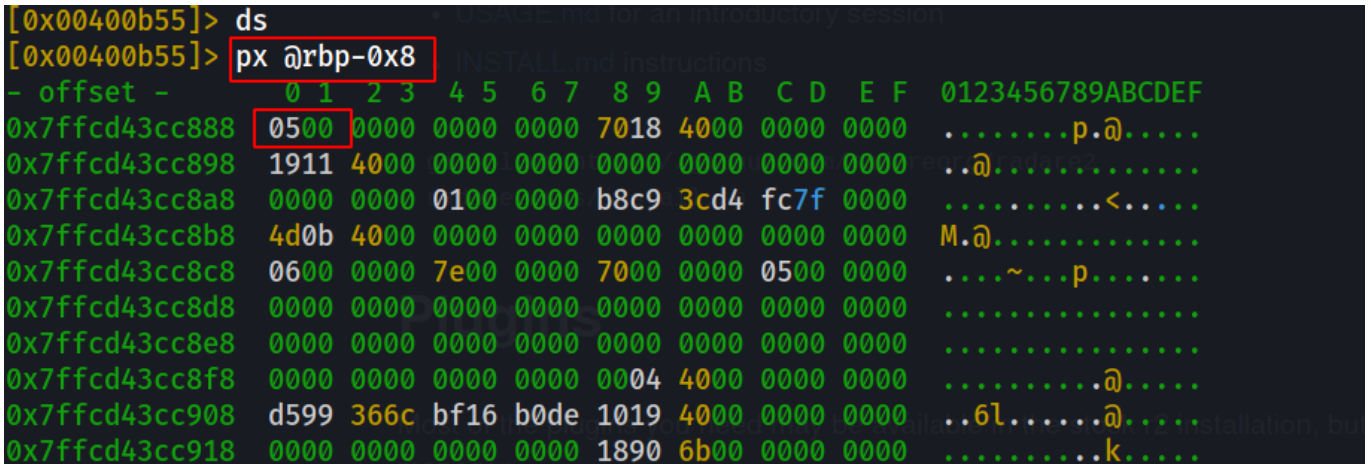
→ now we will goto next instruction using `ds` and then we will again see the value of `var_ch`

→ now we can see the value is 4 in variable `local_ch`

```
[0x00400b55]> ds
[0x00400b55]> px @rbp-0xc
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7ffcd43cc884  0400 0000 1890 6b00 0000 0000 7018 4000  ......k.....p.@.
0x7ffcd43cc894  0000 0000 1911 4000 0000 0000 0000 0000  ......@.........
0x7ffcd43cc8a4  0000 0000 0000 0000 0100 0000 b8c9 3cd4  ..............<.
0x7ffcd43cc8b4  fc7f 0000 4d0b 4000 0000 0000 0000 0000  ....M.@.........
0x7ffcd43cc8c4  0000 0000 0600 0000 7e00 0000 7000 0000  ........~...p...
0x7ffcd43cc8d4  0500 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffcd43cc8e4  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x7ffcd43cc8f4  0000 0000 0000 0000 0000 0000 0004 4000  ..............@.
```

→ Let's do ds again and check the value but this time we will give the address of `local_8h` variable

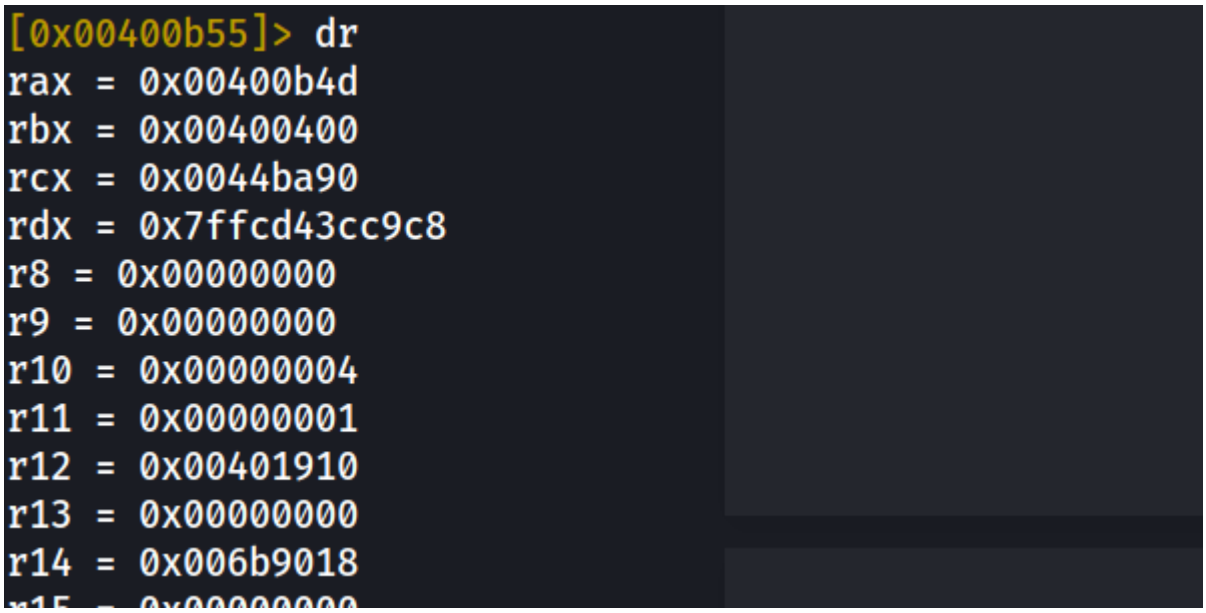
→ and we can see we got the value 5 in variable `local_8h`




---

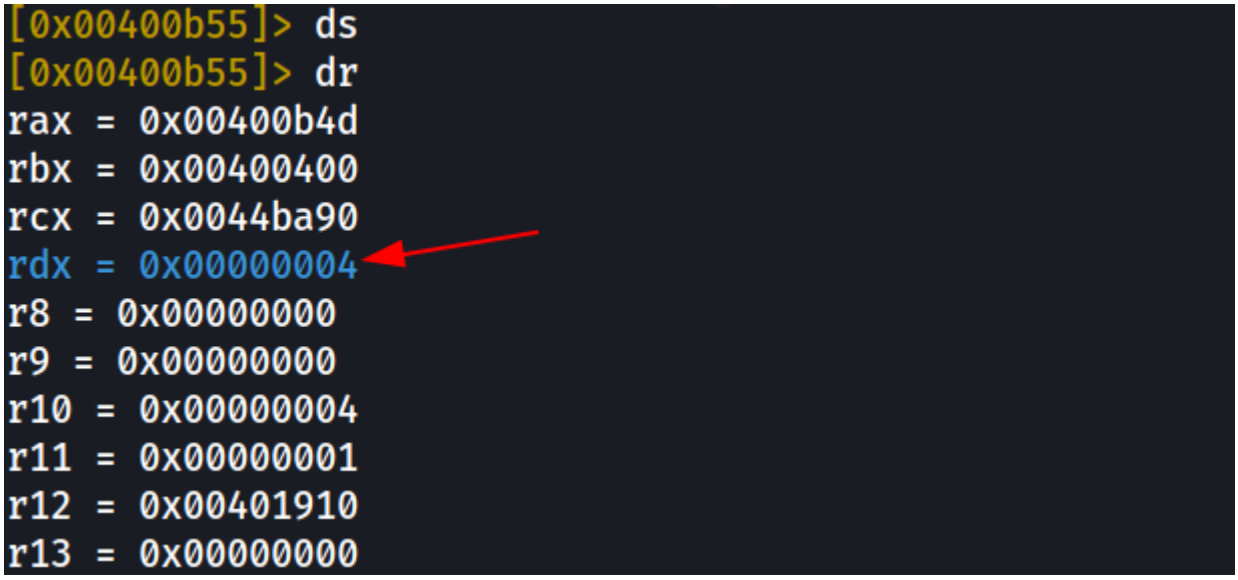
# Analysing register value

→ now we want to see the values of registers so for that we use `dr` command



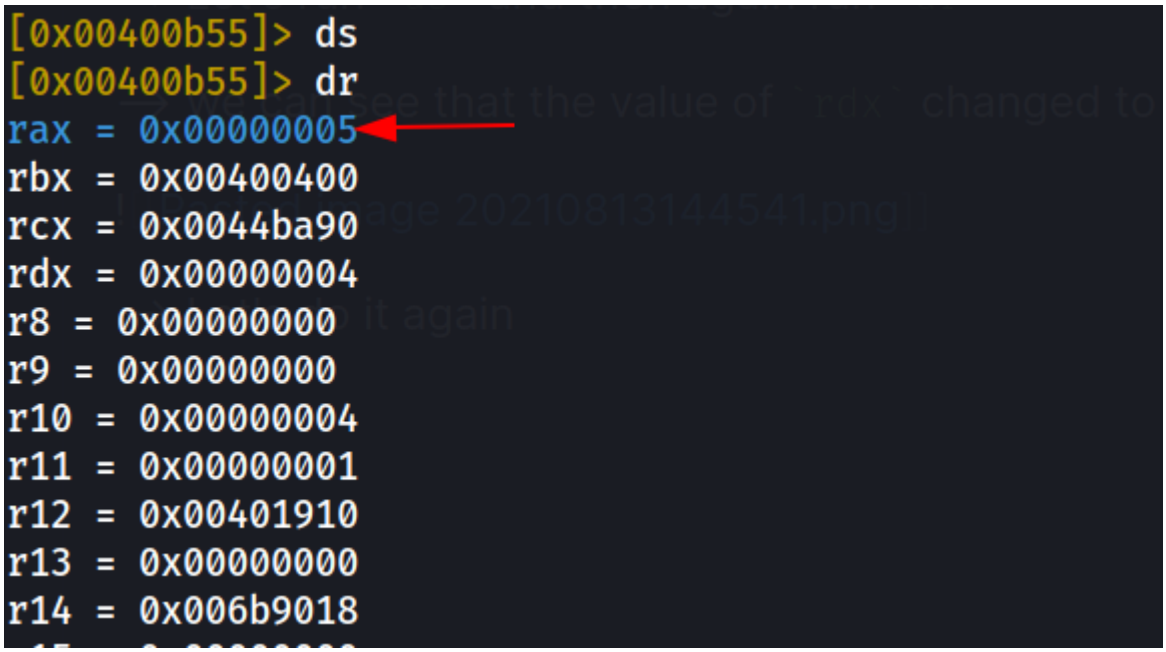

→ Let's run `ds` and then again run `dr`

→ we can see that the value of `rdx` changed to 4




→ Let's do it again

→ and now the value of `rax` changed to 5




→ so now we expect the value of `rax` should be 9 from this expression

```
c745f4040000.  mov dword [var_ch], 4
c745f8050000.  mov dword [var_8h], 5
8b55f4         mov edx, dword [var_ch]
8b45f8         mov eax, dword [var_8h]
01d0           add eax, edx
8945fc         mov dword [var_4h], eax
8b4dfc         mov ecx, dword [var_4h]
8b55f8         mov edx, dword [var_8h]
8b45f4         mov eax, dword [var_ch]
89c6           mov esi, eax
```

→ so Let's try it !

→ and the value is changed !

```
[0x00400b55]> ds
[0x00400b55]> dr
rax = 0x00000009
rbx = 0x00400400
rcx = 0x0044ba90
rdx = 0x00000004
r8 = 0x00000000
r9 = 0x00000000
r10 = 0x00000004
r11 = 0x00000001
r12 = 0x00401910
```

---