

# PSP0201

## Week 5

# Writeup

Group Name: Undecided

Members

ID	Name	Role
1211101390	Aslamia Najwa Binti Ahmad Khadri	Leader
1211100431	Mohammad Omar Torofder	Member
1211103388	Vishnu Karmegam	Member
1211103092	Farryn Aisha binti Muhd Firdaus	Member

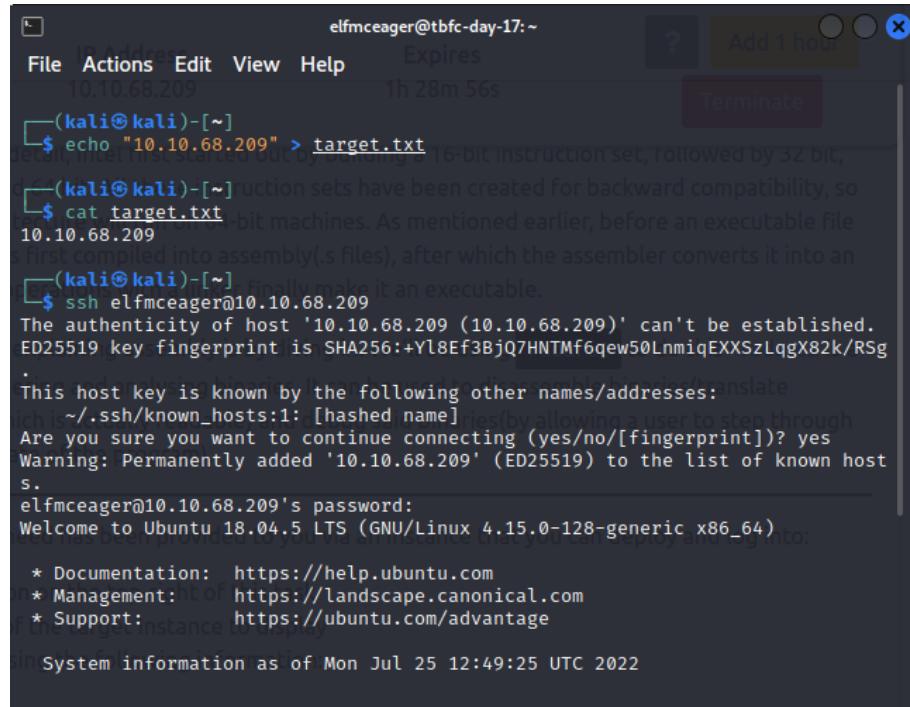
## Day 17: Reverse Engineering – ReverseELFneering

**Tools used:** Attackbox, Firefox, Virtualbox

**Solution/walkthrough:**

### Question 1

View the pdf@main and print the code that is in the sim.main



```
elfmceager@tbfc-day-17:~
```

```
File Actions Edit View Help Expires 1h 28m 56s Add 1 hour Terminate
```

```
10.10.68.209
```

```
(kali㉿kali)-[~]
```

```
$ echo "10.10.68.209" > target.txt
```

```
default instruction sets have been created for backward compatibility, so
```

```
$ cat target.txt
```

```
10.10.68.209
```

```
machines. As mentioned earlier, before an executable file
```

```
is first compiled into assembly(.s files), after which the assembler converts it into an
```

```
(kali㉿kali)-[~]
```

```
$ ssh elfmceager@10.10.68.209
```

```
The authenticity of host '10.10.68.209 (10.10.68.209)' can't be established.
```

```
ED25519 key fingerprint is SHA256:+Yl8Ef3BjQ7HNTMF6qew50LnmiqEXXSzLqgX82k/RSg
```

```
.
```

```
This host key is known by the following other names/addresses:
```

```
~/.ssh/known_hosts:1: [hashed name]
```

```
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

```
Warning: Permanently added '10.10.68.209' (ED25519) to the list of known hosts.
```

```
elfmceager@10.10.68.209's password:
```

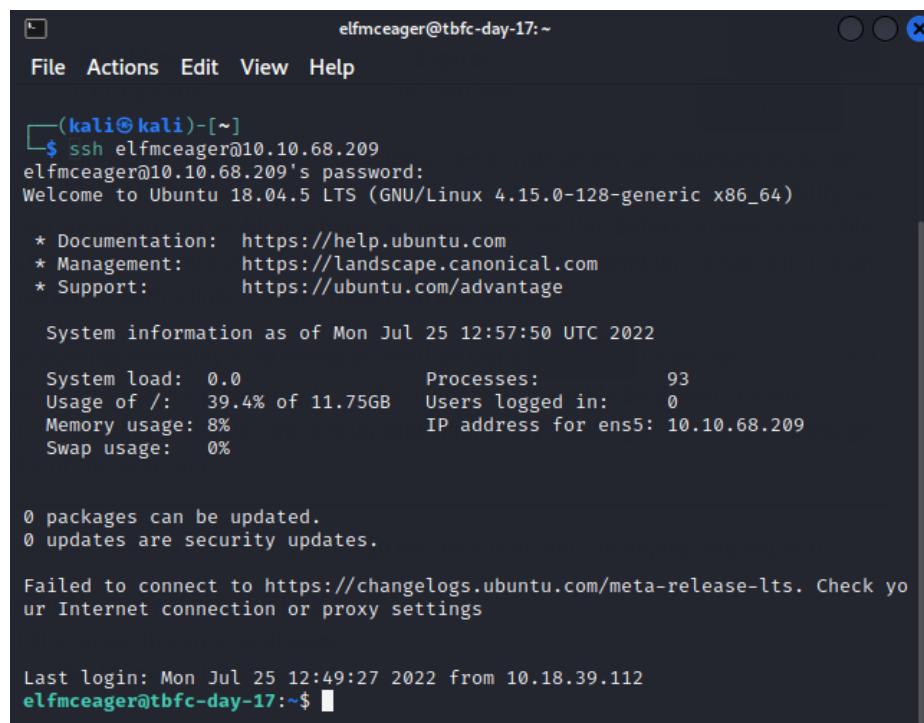
```
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.15.0-128-generic x86_64)
```

```
* Documentation: https://help.ubuntu.com
```

```
* Management: https://landscape.canonical.com
```

```
* Support: https://ubuntu.com/advantage
```

```
System information as of Mon Jul 25 12:49:25 UTC 2022
```



```
elfmceager@tbfc-day-17:~
```

```
File Actions Edit View Help
```

```
(kali㉿kali)-[~]
```

```
$ ssh elfmceager@10.10.68.209
```

```
elfmceager@10.10.68.209'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 Mon Jul 25 12:57:50 UTC 2022
```

```
System load: 0.0 Processes: 93
```

```
Usage of /: 39.4% of 11.75GB Users logged in: 0
```

```
Memory usage: 8% IP address for ens5: 10.10.68.209
```

```
Swap usage: 0%
```

```
0 packages can be updated.
```

```
0 updates are security updates.
```

```
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings
```

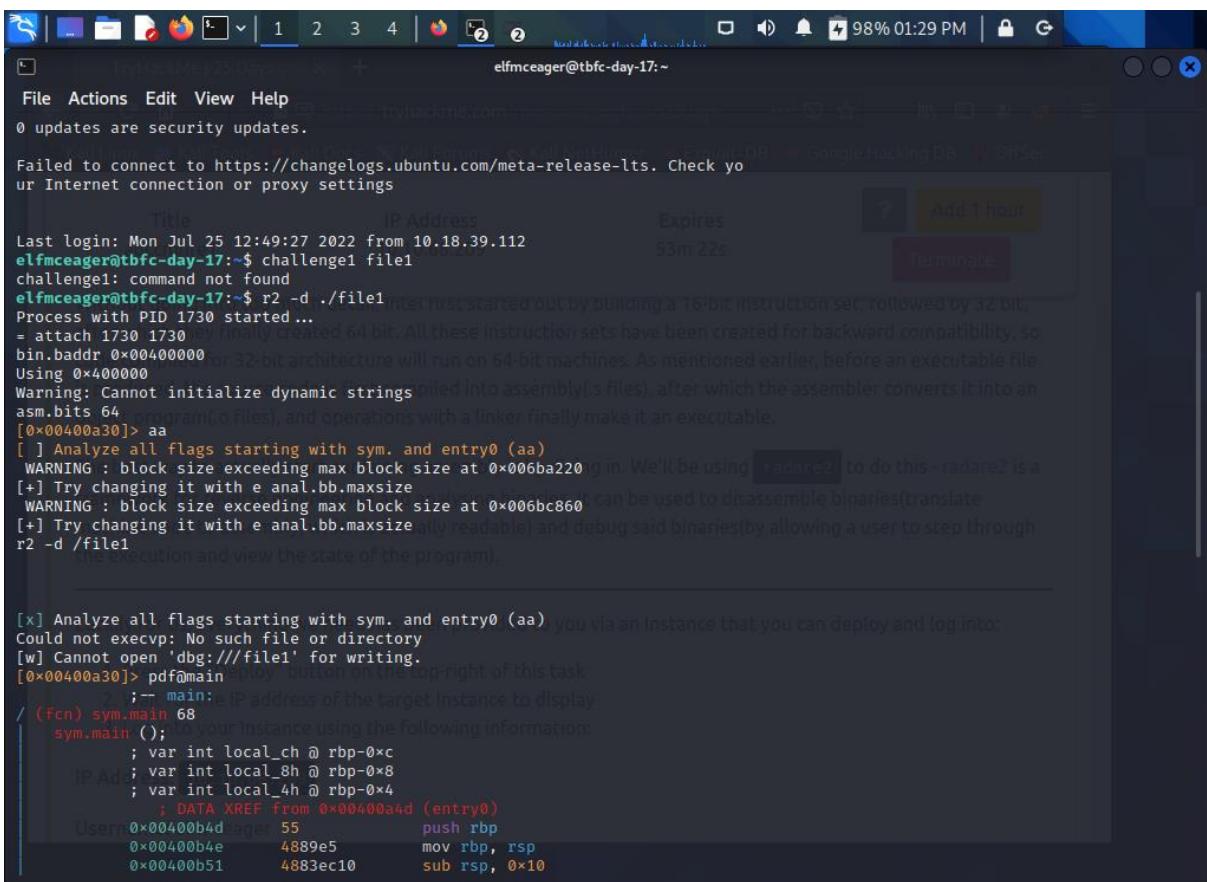
```
Last login: Mon Jul 25 12:49:27 2022 from 10.18.39.112
```

```
elfmceager@tbfc-day-17:~$
```

```
[0x00400a30]> pdf@main
    ;-- main:
/ (fcn) sym.main 68
  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      4883ec10   sub rsp, 0x10
  0x00400b55      c745f4040000 mov dword [local_ch], 4
  0x00400b5c      c745f8050000 mov dword [local_8h], 5
  0x00400b63      8b55f4      mov edx, dword [local_ch]
  0x00400b66      8b45f8      mov eax, dword [local_8h]
  0x00400b69      01d0        add eax, edx
  0x00400b6b      8945fc      mov dword [local_4h], eax
  0x00400b6e      8b4dfc      mov ecx, dword [local_4h]
  0x00400b71      8b55f8      mov edx, dword [local_8h]
  0x00400b74      8b45f4      mov eax, dword [local_ch]
  0x00400b77      89c6        mov esi, eax
  0x00400b79      488d3d881409 lea rdi, qword 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
[0x00400a30]>
```

2. Wait for the IP address of the target instance to display.  
 3. Log into your instance using the following information:

IP Address: 192.168.68.209



elfmceager@tbfc-day-17:~

File Actions Edit View Help

0 updates are security updates.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

```
Last login: Mon Jul 25 12:49:27 2022 from 10.18.39.112
elfmceager@tbfc-day-17:~$ challenge1 file1
challenge1: command not found
elfmceager@tbfc-day-17:~$ r2 -d ./file1
Process with PID 1730 started...
- attach 1730
bin.baddr 0x00400000 or 32-bit architecture will run on 64-bit machines. As mentioned earlier, before an executable file is loaded into assembly(.s files), after which the assembler converts it into an executable.
Warning: Cannot initialize dynamic strings
asm.bits 64
asm.programCoFiles, and operations with a linker finally make it an executable.
[0x00400a30]> aa
[ ] Analyze all flags starting with sym. and entry0 (aa)
WARNING : block size exceeding max block size at 0x006ba220 g in. We'll be using /reduce to do this - radare2 is a
[+] Try changing it with e anal.bb.maxsize
WARNING : block size exceeding max block size at 0x006bc860 can be used to disassemble binaries(translate
[+] Try changing it with e anal.bb.maxsize
r2 -d /file1
[ ] Analyze all flags starting with sym. and entry0 (aa)
[ ] Analyze all flags starting with sym. and entry0 (aa)
Could not execvp: No such file or directory
[w] Cannot open 'dbg:///file1' for writing.
[0x00400a30]> pdf@main
    ;-- main:
/ (fcn) sym.main 68
  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      4883ec10   sub rsp, 0x10
```

A register is a workbench where we can work on task one at a time where we can restore its value. We will have a register that will hold the value of local ch and another folder that hold the local 8h.

We have to perform mathematical functions on the two registers and it will overwrite the source register with the result.

Therefore, the data type that were matched with the size in bytes is byte = 1, Word = 2, Double Word = 4, Quad = 8, Single Precision = 4, Double Precision = 8

## Question 2

We have to run file 1 and attach, r2 -d means debug and type ./file1.

The screenshot shows a terminal window with the following content:

```
elfmceager@tbfc-day-17:~$ challenge1 file1
challenge1: command not found
elfmceager@tbfc-day-17:~$ r2 -d ./file1
Process with PID 1730 started...
= attach 1730 1730
Using 0x400000
Warning: Cannot initialize dynamic strings
asm.bits 64
[0x00400a30]> aa
[ ] Analyze all flags starting with sym. and entry0 (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
r2 -d /file1
```

Thus, `aa` is the command that we entered to start analysing

```
[x] Analyze all flags starting with sym. and entry0 (aa)
Could not execvp: No such file or directory
[w] Cannot open 'dbg:///file1' for writing.
[0x00400a30]> pdf@main
    ;-- main:
    ;-- main:          IP Address           Expires
    ;-- main:          10.10.68.209      51m 28s
    ;-- main:          ? Add 1 hour
    ;-- main:          Terminate
(fcn) sym.main 68
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      4883ec10  sub rsp, 0x10
    ; 0x00400b55      c745f4040000 mov dword [local_ch], 4
    ; 0x00400b5c      c745f8050000 mov dword [local_8h], 5
    ; 0x00400b63      b855f4    mov edx, dword [local_ch]
    ; 0x00400b66      b845f8    mov eax, dword [local_8h]
    ; 0x00400b69      01d0      add eax, edx
    ; 0x00400b6b      8945fc    mov dword [local_4h], eax
    ; 0x00400b6e      b84dfc    mov ecx, dword [local_4h]
    ; 0x00400b71      b855f8    mov edx, dword [local_8h]
    ; 0x00400b74      b845f4    mov eax, dword [local_ch]
    ; 0x00400b77      89c6      mov esi, eax
    ; 0x00400b79      488d3d881409 lea rdi, qword str.the_value_of_a
    ; 0x00400b80      b800000000 mov eax, 0
    ; 0x00400b85      e8f6ea0000 call sym._printf
    ; 0x00400b8a      b800000000 mov eax, 0
    ; 0x00400b8f      c9        leave
    ; 0x00400b90      c3        ret
[0x00400a30]> db 0x00400b55
[0x00400a30]> pdf@main
    ;-- main:
    ;-- main:          IP Address           Expires
    ;-- main:          10.10.68.209      51m 28s
    ;-- main:          ? Add 1 hour
    ;-- main:          Terminate
(fcn) sym.main 68
sym.main ();
    ; var int local_ch @ rbp-0xc
```

### Question 3

We can set the breakpoint by using the db command. Type db0x00400b55 and pdf@main.

The screenshot shows a terminal window with the GDB debugger running. The assembly code for the main function is displayed, showing instructions like mov, lea, and call. A breakpoint is set at address 0x00400b55, indicated by a lowercase 'b' prefix in the assembly listing. The PDF command is used to generate documentation for the main function. A tooltip for the PDF command indicates it's for generating documentation for 32-bit executables. The terminal also shows some internal GDB messages and a warning about the architecture being 32-bit.

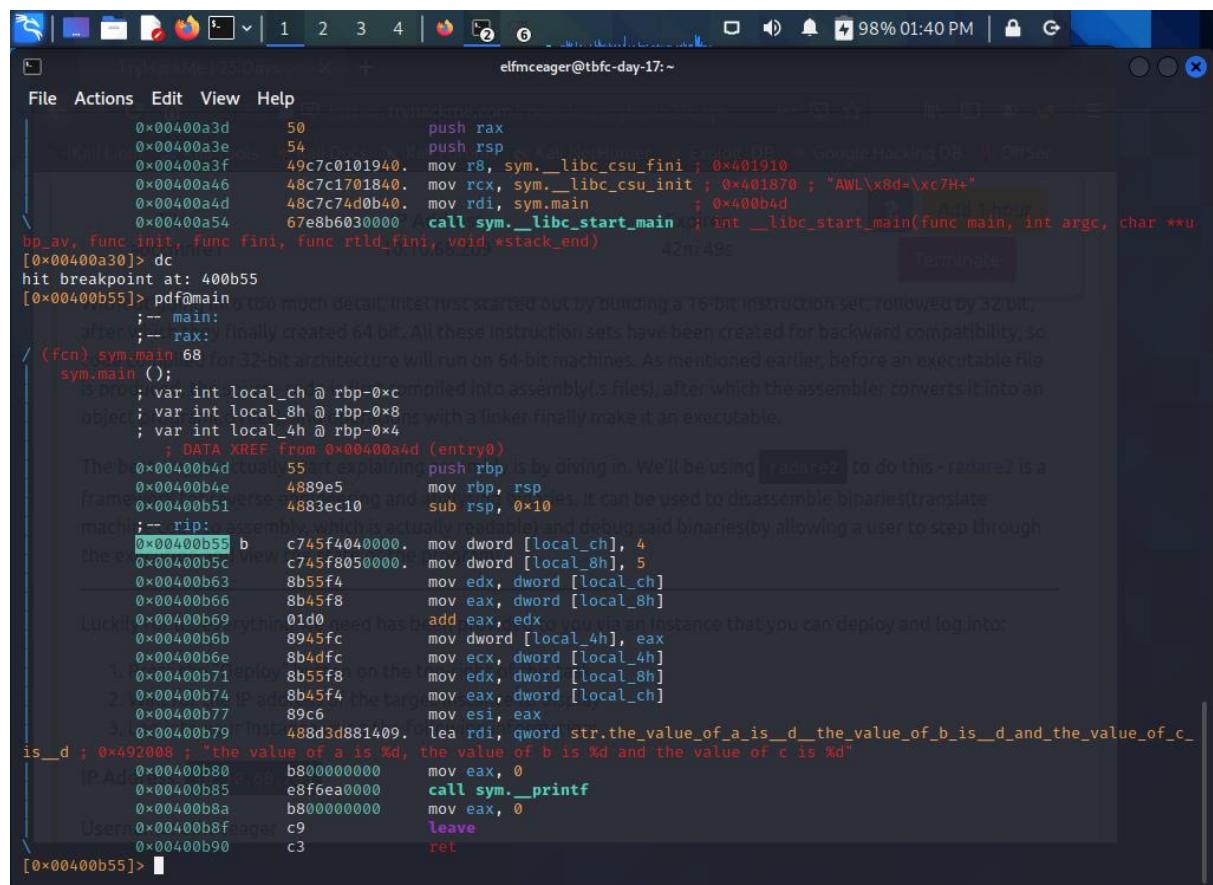
```
elfmceager@tbfc-day-17:~$ file /tmp/bf15day17/bf15day17
elfmceager@tbfc-day-17:~$ ./bf15day17
[0x00400a30]> db 0x00400b55
[0x00400a30]> pdf@main
/ (fcn) sym.main 68
sym.main ();
The best ; DATA XREF from 0x00400a4d (entry0) is by diving in. We'll be using !radare2 to do this - radare2 is a
frames ;-- main: debugger and can be used to disassemble binaries(translate
mach 0x00400b51 sym.main:0x00400b51
the e 0x00400b55 b c745f4040000. mov dword [local_ch], 4
0x00400b5c c745f8050000. mov dword [local_8h], 5
0x00400b63 8b55f4 mov edx, dword [local_ch]
0x00400b66 8b45f8 mov eax, dword [local_8h]
0x00400b69 01d0 add eax, edx
0x00400b6b 8945fc mov dword [local_4h], eax
0x00400b6e 8b40fc mov ecx, dword [local_4h]
1. 0x00400b71 b855f8 mov edx, dword [local_8h]
2. 0x00400b74 b845f4 mov eax, dword [local_ch]
3. 0x00400b77 89c6 mov esi, eax
0x00400b79 488d3d881409. lea rdi, qword 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
[0x00400a30]>
```

We will see a lowercase b next to it which is telling us that we have a breakpoint.

#### Question 4

We can type in dc which starts sunning the code and it will run up the first breakpoint.

```
[0x00400a30]> dc
child stopped with signal 28 (SIGPOLL, which is actually readable) and debug said binaries by allowing a user to step through
[+] SIGNAL 28 errno=0 addr=0x00000000 code=128 ret=0
[0x00400a30]> pdf
;-- rip:
(fcn) entry0 42
entry0 ();
0x00400a30 31ed xor ebp, ebp
0x00400a32 4989d1 mov r9, rdx
0x00400a35 5e pop rsi
0x00400a36 4889e2 mov rdx, rsp
0x00400a39 4883e4f0 and rsp, 0xfffffffffffff0
0x00400a3d 50 push rax
0x00400a3e 54 push rsp
0x00400a3f 49c7c0101940. mov r8, sym._libc_csu_fini ; 0x401910
0x00400a46 48c7c1701840. mov rcx, sym._libc_csu_init ; 0x401870 ; "AWL\x8d=\xc7H+"
0x00400a4d 48c7c74d0b40. mov rdi, sym.main ; 0x400b4d
\ User0x00400a54 67e8b6030000 call sym._libc_start_main ; int __libc_start_main(func main, int argc, char **u
bp_av, func init, func fini, func rtld_fini, void *stack_end)
[0x00400a30]>
```



The screenshot shows the radare2 debugger interface with the assembly view open. The assembly code for the main function is displayed, starting with the instruction at address 400b55. The instruction at 400b55 is highlighted in yellow, indicating it is the current instruction being executed. The assembly code includes comments explaining the purpose of each instruction, such as setting up the stack frame and preparing for function calls. The interface has a dark theme with syntax highlighting for assembly instructions.

```
File Actions Edit View Help
0x00400a3d 50 push rax
0x00400a3e 54 push rsp
0x00400a3f 49c7c0101940. mov r8, sym._libc_csu_fini ; 0x401910
0x00400a46 48c7c1701840. mov rcx, sym._libc_csu_init ; 0x401870 ; "AWL\x8d=\xc7H+"
0x00400a4d 48c7c74d0b40. mov rdi, sym.main ; 0x400b4d
\ User0x00400a54 67e8b6030000 call sym._libc_start_main ; int __libc_start_main(func main, int argc, char **u
bp_av, func init, func fini, func rtld_fini, void *stack_end)
[0x00400a30]> dc
hit breakpoint at: 400b55
[0x00400b55]> pdf@main
;-- main:
;-- rax:
;-- rip:
(fcn) sym.main 68
sym.main ();
is prog; var int local_ch @ rbp-0xc
; var int local_8h @ rbp-0x8
; var int local_4h @ rbp-0x4
; DATA XREF from 0x00400a4d (entry0)
The b0x00400b4d 55 push rbp is by diving in. We'll be using radare2 to do this - radare2 is a
0x00400b4e 4889e5 mov rbp, rsp
frame; 0x00400b51 4883ec10 sub rsp, 0x10
machine; ;-- rip: assembly which is actually readable and debug said binaries by allowing a user to step through
the e0x00400b55 b c745f4040000. mov dword [local_ch], 4
0x00400b5c c745f8050000. mov dword [local_8h], 5
0x00400b63 8b55f4 mov edx, dword [local_ch]
0x00400b66 8b45f8 mov eax, dword [local_8h]
0x00400b69 01d0 add eax, edx
0x00400b6b 8945fc mov dword [local_4h], eax
0x00400b6e 8b4dfc mov ecx, dword [local_4h]
0x00400b71 8b55f8 mov edx, dword [local_8h]
0x00400b74 8b45f4 mov eax, dword [local_ch]
0x00400b77 89c6 mov esi, eax
0x00400b79 488d3d881409. lea rdi, qword 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"
IP Address: 0x00400b80 b800000000 mov eax, 0
0x00400b85 e8f6ea0000 call sym._printf
0x00400b8a b800000000 mov eax, 0
0x00400b8f c9 leave
0x00400b90 c3 ret
[0x00400b55]>
```

We have set a memory location 400b55 and that is where it stopped. When we do a pdf@main again, we will see it is highlighted and there is a rip comment where we are currently sitting.

elfmceager@tbfc-day-17:~

```
[File] [Actions] [Edit] [View] [Help]
0x00400b80 b800000000 mov eax, 0
0x00400b85 e8f6ea0000 call sym._printf
0x00400b8a b800000000 mov eax, 0
0x00400b8f c9 leave
0x00400b90 c3 ret

[0x00400b55]> dc
child stopped with signal 28
[+] SIGNAL 28 errno=0 addr=0x00000000 code=128 ret=0
hit breakpoint at: 400b55
[0x00400b55]> pdf@main
The binary you are currently viewing has been built for 32-bit architecture. If you want to see assembly for much detail, Interfacedisassembly by building a 16-bit instruction set followed by 32 bit.
;-- main:
;-- rax: finally created 64 bit. All these instruction sets have been created for backward compatibility, so
/ (Fcn) sym.main 68
sym.main ();
  ; var int local_ch @ rbp-0xc
  ; var int local_8h @ rbp-0x8
  ; var int local_4h @ rbp-0x4
    ; DATA XREF from 0x00400a4d (entry)
The binary you are currently viewing has been built for 32-bit architecture. If you want to see assembly for much detail, Interfacedisassembly by building a 16-bit instruction set followed by 32 bit.
0x00400b4d 55 start explaining push rbp is by diving in. We'll be using radare2 to do this - radare2 is a
0x00400b4e 4889e5 mov rbp, rsp
frame pointer. It can be used to disassemble binaries(translate
0x00400b51 4883ec10 sub rsp, 0x10
machine code into assembly, which is actually readable) and debug said binaries(by allowing a user to step through
the assembly code).
0x00400b55 b c745f4040000. mov dword [local_ch], 4
0x00400b5c c745f8050000. mov dword [local_8h], 5
0x00400b63 8b55f4 mov edx, dword [local_ch]
0x00400b66 8b45f8 mov eax, dword [local_8h]
0x00400b69 01d0 add eax, edx
0x00400b6b 8945fc mov dword [local_4h], eax
0x00400b6e 8b4dfc mov ecx, dword [local_4h]
0x00400b71 8b55f8 mov edx, dword [local_8h]
0x00400b74 8b45f4 mov eax, dword [local_ch]
0x00400b77 89c6 mov esi, eax
0x00400b79 488d3d881409. lea rdi, qword 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
User: elfmceager@tbfc-day-17:~
```

## Question 5,6 & 7

When we look at the code again, we will see that we are right at the line where we do the addition of eax + edx

The screenshot shows the assembly dump of the exploit binary. The assembly code is displayed in a terminal window, showing instructions like 'px@rbp-0xc' and 'leave'. The assembly output is heavily annotated with comments explaining the assembly language. A tooltip from 'radare2' is visible, stating: 'assembly, which is actually readable) and debug said binaries(by allowing a user to step through assembly, which is actually readable) and debug said binaries(by allowing a user to step through assembly that you can deploy and log into:'. The assembly code includes various registers (eax, edx, etc.) and memory addresses.

The screenshot shows the assembly dump of the exploit binary in a Kali Linux terminal. The assembly code is annotated with comments explaining the assembly language. A tooltip from 'radare2' is visible, stating: 'assembly, which is actually readable) and debug said binaries(by allowing a user to step through assembly that you can deploy and log into:'. The assembly code includes various registers (eax, edx, etc.) and memory addresses.

The screenshot shows the radare2 debugger interface. The assembly window displays the following code:

```
[0x00400b55]> px0rbp-0x8
0x7ffd46056084 0000 0000 0000 0000 0000 0000 0000 0000 .....
[0x00400b55]> px0rbp-0x8
- offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x7ffd46055f98 1890 6b00 0000 0000 7018 4000 0000 0000 ..k.....p.@...
0x7ffd46055fa8 1911 4000 0000 0000 0000 0000 0000 0000 ..@.....
0x7ffd46055fd8 0000 0000 0100 0000 c360 0546 fd7f 0000 .....`F.ores
0x7ffd46055fc8 4d0b 4000 0000 0000 0000 0000 0000 0000 M.@.
0x7ffd46055fd8 1700 0000 0100 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46055fe8 0000 0000 0200 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46055ff8 0000 0000 0000 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46056008 0000 0000 0000 0000 0004 4000 0000 0000 .....`F.ores
0x7ffd46056018 2af2 fe7c 95c8 a7fe 1019 4000 0000 0000 *..lets have been created for backward compatibility, so
0x7ffd46056028 0000 0000 0000 0000 1890 6b00 0000 0000 .....`F.ores
0x7ffd46056038 0000 0000 0000 0000 2af2 7ef3 1f44 5d01 .....`F.ores
0x7ffd46056048 2af2 8a6d 95c8 a7fe 0000 0000 0000 0000 *..ms file, and which the assembler converts it into an
0x7ffd46056058 0000 0000 0000 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46056068 0000 0000 0000 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46056078 0000 0000 0000 0000 0000 0000 0000 0000 .....`F.ores
0x7ffd46056088 0000 0000 0000 0000 0000 0000 0000 0000 .....`F.ores
[0x00400b55]> dr
rax = 0x00400b4d
rbx = 0x00400400
rcx = 0x0044ba90
rdx = 0x7ffd460560d8
r8 = 0x01000000
r9 = 0x006bb8e0
r10 = 0x00000015
r11 = 0x00000000
r12 = 0x00401910
r13 = 0x00000000
r14 = 0x006b9018
r15 = 0x00000000
rsi = 0x7ffd460560c8 our Instance using the following information:
rdi = 0x00000001
rsp = 0x7ffd46055f90 10.68.209
rbp = 0x7ffd46055fa0
rip = 0x00400b5c
rflags = 0x00000206 mceager
orax = 0xffffffffffff
[0x00400b55]>
```

We have to look at the registers, `rax = 5`, `rdx = 4`, when we step, we will perform the addition and one of our registers will be modified.

```

File Actions Edit View Help
rbp = 0xffffd46055fa0
rip = 0x00400b63
rflags = 0x00000206
orax = 0xffffffffffffffff
[0x00400b55]> pdf@main
;-- main:
;-- rax:1
(fcn) sym.main 68
sym.main ();
Without ; var int local_ch @ rbp-0xc ; Net first started out by building a 16-bit instruction set, followed by 32 bit,
; var int local_8h @ rbp-0x8 ; after which the assembler converts it into an
; var int local_4h @ rbp-0x4 ; All these instruction sets have been created for backward compatibility, so
code comp DATA XREF from 0x00400a4d (entry0)
is proc 0x00400b4e source 4889e5 first comp
object 0x00400b51 files 4883ec10 mains
0x00400b55 b c745f4040000. mov dword [local_ch], 4
0x00400b5c c745f8050000. mov dword [local_8h], 5
The beginning of rip: actually start explaining assembly is by diving in. We'll be using radare2 to do this - radare2 is a
frame 0x00400b63 reverse engineering and used to disassemble binaries(translate
machines 0x00400b69assem 01d0 which is actually add eax, edx and debug said binaries(by allowing a user to step through
the exec 0x00400b66 view source of the
0x00400b66 8b40fc mov dword [Local_4h], eax
0x00400b71 8b55f8 mov edx, dword [local_8h]
0x00400b74 8b45f4 mov eax, dword [local_ch]
0x00400b77 89c6 mov esi, eax
Luckily 0x00400b79 488d3d881409. lea rdi, qword 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"
1. 0x00400b80 IPad 0x00400b80 target
2. 0x00400b85 e8f6ea0000 call sym._printf
3. 0x00400b8f Instace using the following information:
0x00400b90 c9 leave
0x00400b90 c3 ret

[0x00400b55]> dr 10.10.68.209
rax = 0x00400b4d
rbx = 0x00400400
rcx = 0x0044ba90 elfmceager
rdx = 0xffffd460560d8
r8 = 0x01000000
r9 = 0x006bb8e0
r10 = 0x00000015
r11 = 0x00000000
r12 = 0x00401910
r13 = 0x00000000
r14 = 0x006b9018
r15 = 0x00000000
rsi = 0xffffd460560c8
rdi = 0x00000001
rsp = 0xffffd46055f90
rbp = 0xffffd46055fa0
rip = 0x00400b63
rflags = 0x00000206
orax = 0xffffffffffffffff

```

Thus, 1 is the value of local\_ch when its corresponding movl instruction is called, 6 is the value of eax when the imull instruction is called, and 6 is the value of local\_4h before eax is set to 0.

**Thought Process/Methodology:**

Print the code found in sim.main and view the pdf@main. A register serves as a work surface where we can do tasks one at a time in order to recover their value. The local ch value will be kept in a register, while the local 8h value will be kept in another folder. The two registers must be subjected to mathematical operations, and the results will be written over the source register. As a result, byte = 1, word = 2, double word = 4, quad = 8, single precision = 4, double precision = 8, and the data type were matched with the size in bytes. To execute file 1 and attach, type./file1. Type r2 -d to debug. Thus, we entered the command aa to begin our analysis. Using the db command, we can set the breakpoint. Enter pdf@main and db0x00400b55. The fact that there is a lowercase b next to it indicates that we have reached a breakpoint. We may start the code by typing dc, and it will run up the first breakpoint. It halted at a memory location that we placed at 400b55. It will be underlined and have a rip comment where we are sitting if we run a pdf@main once more. Rereading the code will reveal that we are currently on the line where the addition of eax and edx is performed. When we step, we will conduct the addition, and one of our registers will be changed. The registers are: rax = 5, rdx = 4, etc. Thus, 1 represents the value of local ch when the movl instruction corresponding to it is called, 6 represents the value of eax when the imull instruction is executed, and 6 represents the value of local 4h prior to eax being set to 0.