Disassembly

We are given a binary file called letter_frequencies . Open it in radare2 by using :

r2 ./letter_frequencies

This will open the seek commander , to move to the desired offset , we add Shift+V+Enter to open hex view , then add p to switch to disassembler view and adding p again switches to debugger view. Adding lower case p again and again will switch views , there are string views, ascii views , color-coded views etc. To go back to the views , we will add uppercase P.

Shift + ; opens a prompt for us to input in radare2

AFL Analyze function list to see if it could find any function.

aaa automatically analyze the binary , after that u can use afl to see the function list.

s main + enter + enter to move to the main function in the disassembler code

j/k + enter j and k will move the cursor and by using enter u can jump to the code of that function

u from the code of function back to disassembler

Cross-References

To find where and how many times a certain function is being called .

axt - “Analyze Cross-References To” . When you provide an address or function name as an argument to the axt command, Radare2 searches through the disassembly to find instructions or data that reference the specified address or function.

s(seek) address - to go to location where the function is called

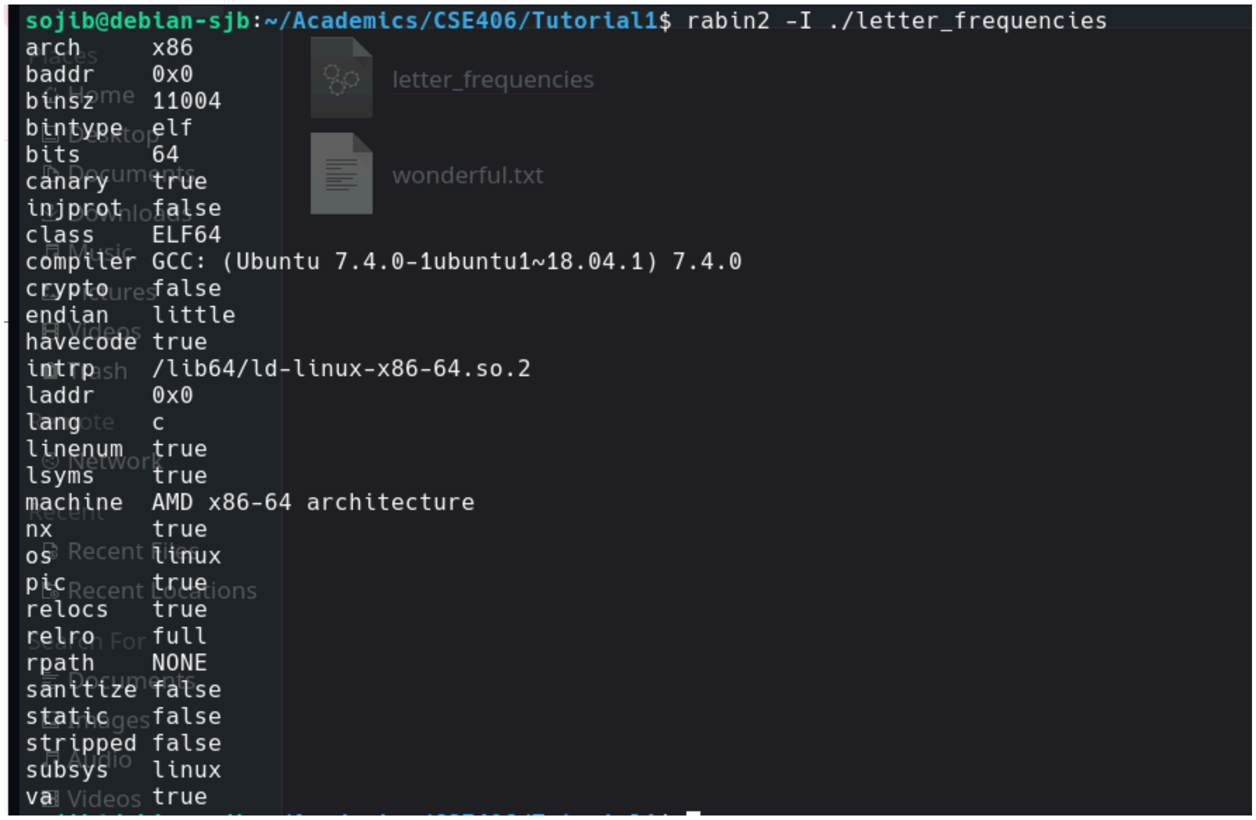
Viewing Imports, Exports, and Strings

i? - find commands that will start with the letter i (same thing can be done for other commands)
ii - "info imports," displays information about imported functions from external libraries or dynamic link libraries (DLLs).
iE - “info exports” shows the functions here that are used for exports
iS - Shows the sections (text , bss , data)

Getting General Info about a Binary

Rabin2 : Binary Program info extractor

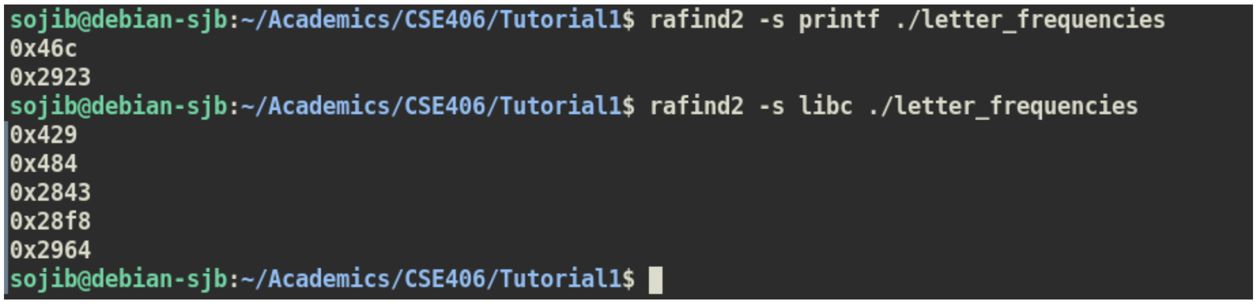
```
-I             Show binary info (see iI command in r2)
```



```
sojib@debian-sjb:~/Academics/CSE406/Tutorial1$ rabin2 -I ./letter_frequencies
arch     x86
baddr    0x0
binsz    11004
bintype  elf
bits     64
canary   true
injprot  false
class    ELF64
compiler GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  AMD x86-64 architecture
nx       true
os       linux
pic      true
relocs   true
relro    full
rpath    NONE
sanitize false
static   false
stripped false
subsys   linux
va       true
```

rafind2 — advanced command-line byte pattern search in files

```
-s str         Search for a specific string
```

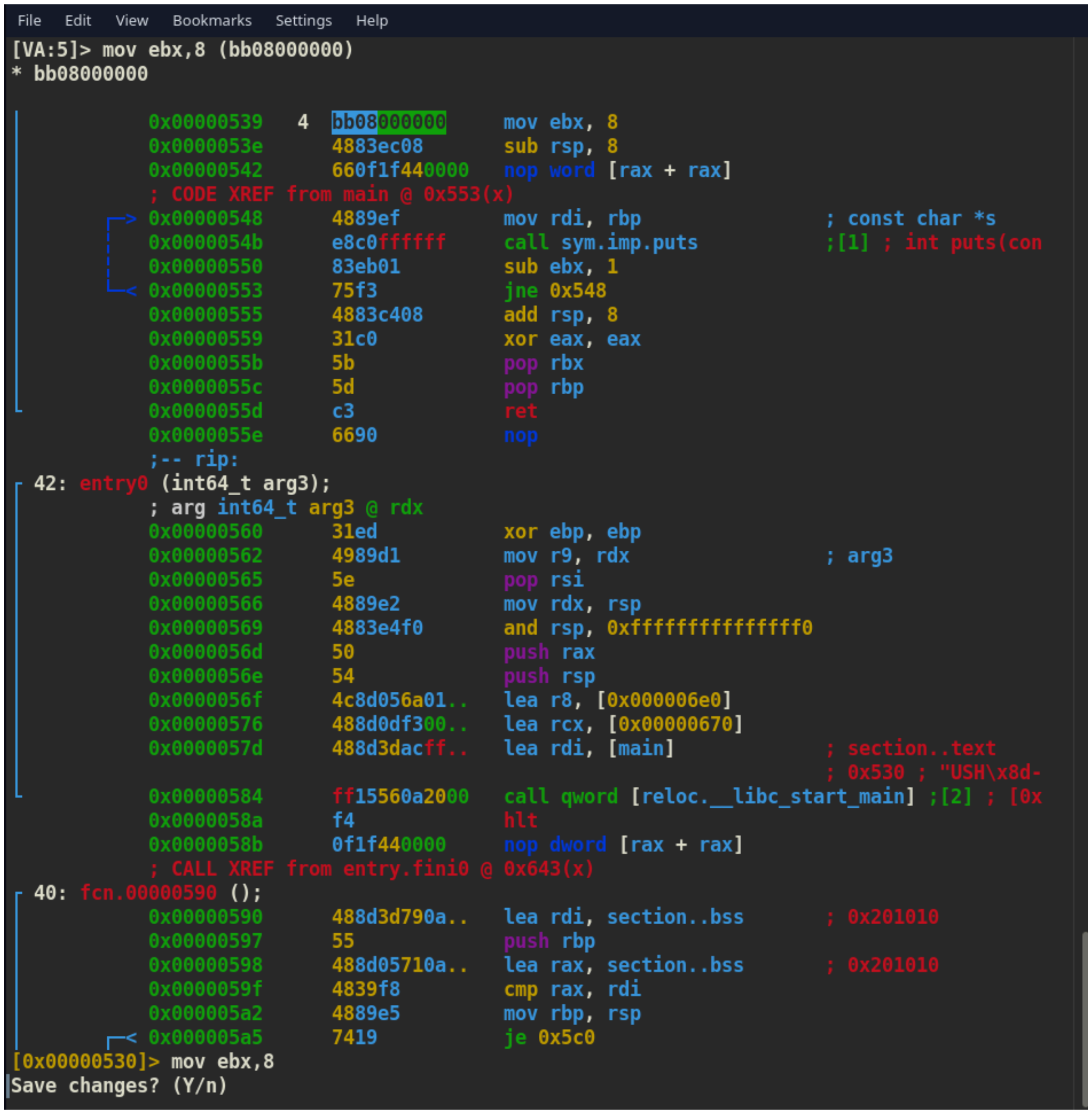


```
sojib@debian-sjb:~/Academics/CSE406/Tutorial1$ rafind2 -s printf ./letter_frequencies
0x46c
0x2923
sojib@debian-sjb:~/Academics/CSE406/Tutorial1$ rafind2 -s libc ./letter_frequencies
0x429
0x484
0x2843
0x28f8
0x2964
sojib@debian-sjb:~/Academics/CSE406/Tutorial1$ 
```

Patching Binaries

Patching binaries refers to the process of modifying a compiled executable file (binary) by directly altering its machine code instructions, data, or metadata.
In our given wonderful file , it is printing wonderful 3 times

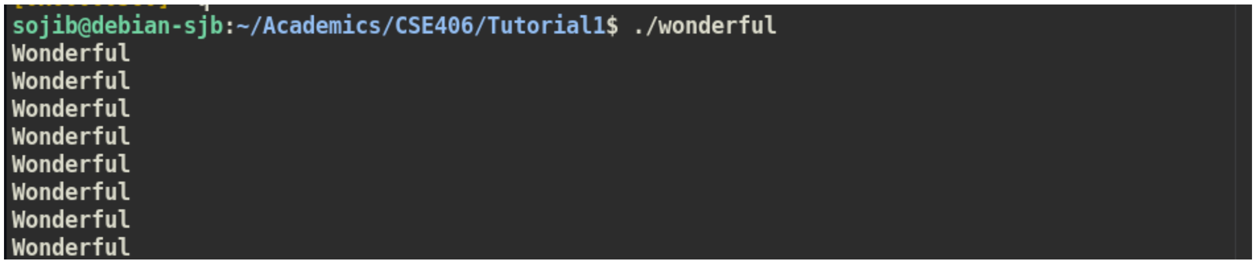
`r2 -w ./wonderful`
opens the binary in radare2 with write mode
`afl`
see the function list
`s main`
Go to the address of main
`Shift+v`
Open it in view
`p`
Open in debugger mode
mov ebx,3 (this is the loop variable)
`shift+a` and change the command , press enter until it’s the topmost line and save using y



```
[VA:5]> mov ebx,8 (bb08000000)
* bb08000000

│           0x00000539    4   bb08000000     mov ebx, 8
│           0x0000053e        4883ec08       sub rsp, 8
│           0x00000542        660f1f440000   nop word [rax + rax]
│           ; CODE XREF from main @ 0x553(x)
│       ┌─> 0x00000548        4889ef         mov rdi, rbp              ; const char *s
│       ╎   0x0000054b        e8c0ffffff     call sym.imp.puts         ;[1] ; int puts(con
│       ╎   0x00000550        83eb01         sub ebx, 1
│       └─< 0x00000553        75f3           jne 0x548
│           0x00000555        4883c408       add rsp, 8
│           0x00000559        31c0           xor eax, eax
│           0x0000055b        5b             pop rbx
│           0x0000055c        5d             pop rbp
└           0x0000055d        c3             ret
            0x0000055e        6690           nop
            ;-- rip:
┌ 42: entry0 (int64_t arg3);
│           ; arg int64_t arg3 @ rdx
│           0x00000560        31ed           xor ebp, ebp
│           0x00000562        4989d1         mov r9, rdx               ; arg3
│           0x00000565        5e             pop rsi
│           0x00000566        4889e2         mov rdx, rsp
│           0x00000569        4883e4f0       and rsp, 0xfffffffffffffff0
│           0x0000056d        50             push rax
│           0x0000056e        54             push rsp
│           0x0000056f        4c8d056a01..   lea r8, [0x000006e0]
│           0x00000576        488d0df300..   lea rcx, [0x00000670]
│           0x0000057d        488d3dacff..   lea rdi, [main]           ; section..text
│                                                                      ; 0x530 ; "USH\x8d-
└           0x00000584        ff15560a2000   call qword [reloc.__libc_start_main] ;[2] ; [0x
            0x0000058a        f4             hlt
            0x0000058b        0f1f440000     nop dword [rax + rax]
            ; CALL XREF from entry.fini0 @ 0x643(x)
┌ 40: fcn.00000590 ();
│           0x00000590        488d3d790a..   lea rdi, section..bss     ; 0x201010
│           0x00000597        55             push rbp
│           0x00000598        488d05710a..   lea rax, section..bss     ; 0x201010
│           0x0000059f        4839f8         cmp rax, rdi
│           0x000005a2        4889e5         mov rbp, rsp
│       ┌─< 0x000005a5        7419           je 0x5c0
[0x00000530]> mov ebx,8
Save changes? (Y/n)
```

And now :



```
sojib@debian-sjb:~/Academics/CSE406/Tutorial1$ ./wonderful
Wonderful
Wonderful
Wonderful
Wonderful
Wonderful
Wonderful
Wonderful
Wonderful
```