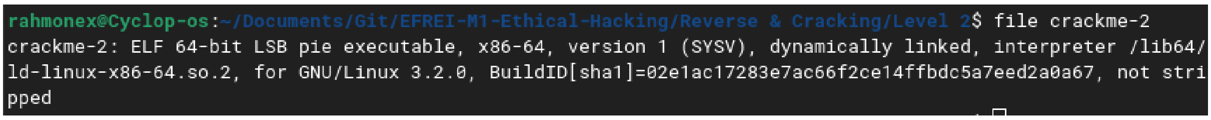
Bilel RAHMOUNI & Meriem NOUIRA

Reverse & Cracking

Crack : Crack-2

1 . Identify the file

At first, we identify the type of the file with the command file.



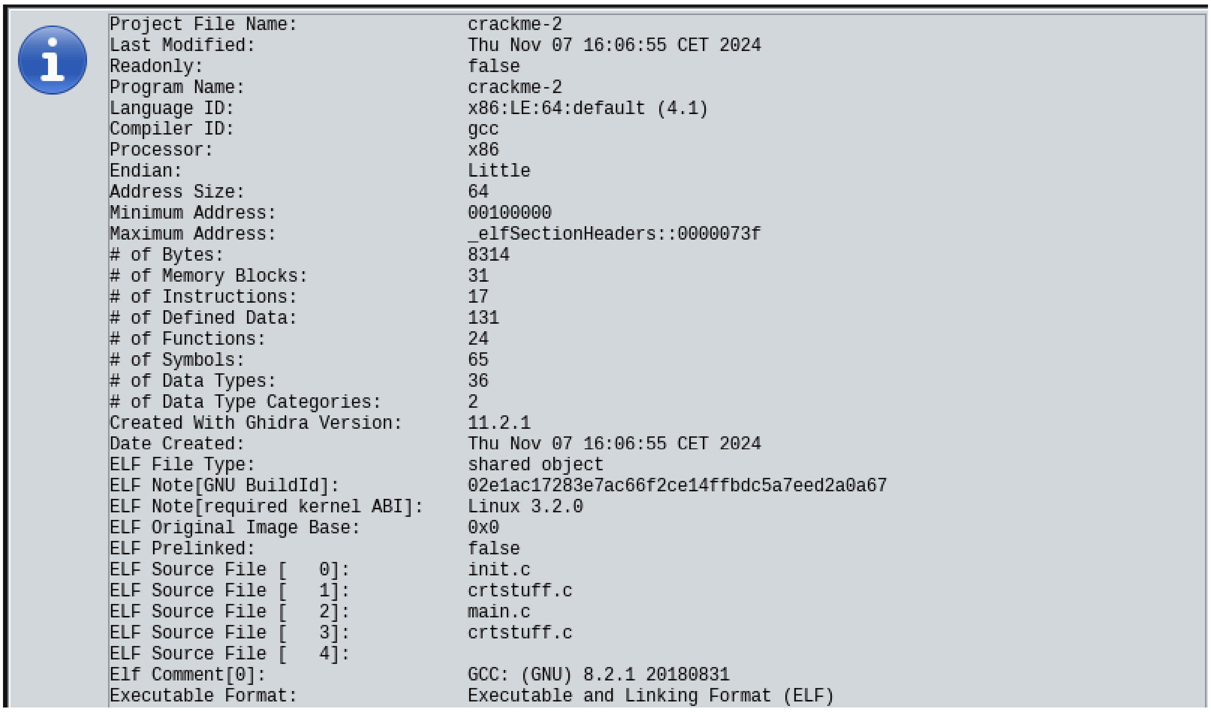
```
rahmonex@Cyclop-os:~/Documents/Git/EFREI-M1-Ethical-Hacking/Reverse & Cracking/Level 2$ file crackme-2
crackme-2: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/
ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=02e1ac17283e7ac66f2ce14ffbdc5a7eed2a0a67, not stri
pped
```

File type : ELF 64-bit LSB pie executable

2 . Analyze the Binary in Ghidra

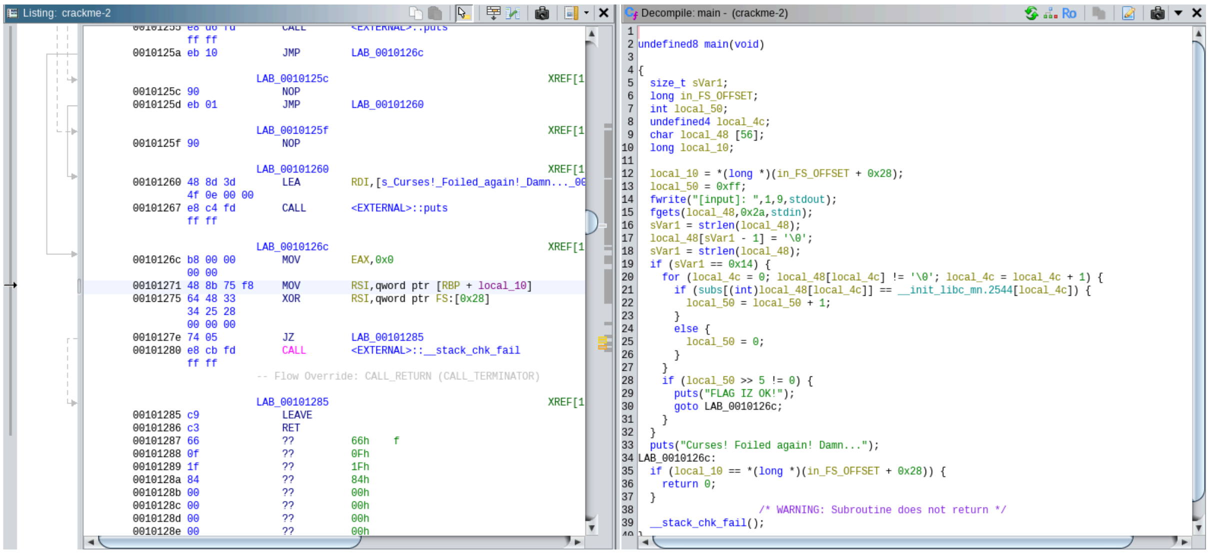
We used Ghidra to analyze the crackme-2 binary, focusing on two critical arrays: subs and __init_libc_mn.2544. The subs array contains transformed values for ASCII characters, while __init_libc_mn.2544 holds the expected output values needed for input validation.

We install Ghidra, and we import the binary file crakme-2 :



```
Project File Name:              crackme-2
Last Modified:                  Thu Nov 07 16:06:55 CET 2024
Readonly:                       false
Program Name:                   crackme-2
Language ID:                    x86:LE:64:default (4.1)
Compiler ID:                    gcc
Processor:                      x86
Endian:                         Little
Address Size:                   64
Minimum Address:                00100000
Maximum Address:                _elfSectionHeaders::0000073f
# of Bytes:                     8314
# of Memory Blocks:             31
# of Instructions:              17
# of Defined Data:              131
# of Functions:                 24
# of Symbols:                   65
# of Data Types:                36
# of Data Type Categories:      2
Created With Ghidra Version:    11.2.1
Date Created:                   Thu Nov 07 16:06:55 CET 2024
ELF File Type:                  shared object
ELF Note[GNU BuildId]:          02e1ac17283e7ac66f2ce14ffbdc5a7eed2a0a67
ELF Note[required kernel ABI]:  Linux 3.2.0
ELF Original Image Base:        0x0
ELF Prelinked:                  false
ELF Source File [   0]:         init.c
ELF Source File [   1]:         crtstuff.c
ELF Source File [   2]:         main.c
ELF Source File [   3]:         crtstuff.c
ELF Source File [   4]:
Elf Comment[0]:                 GCC: (GNU) 8.2.1 20180831
Executable Format:              Executable and Linking Format (ELF)
```

And we analyze the file :



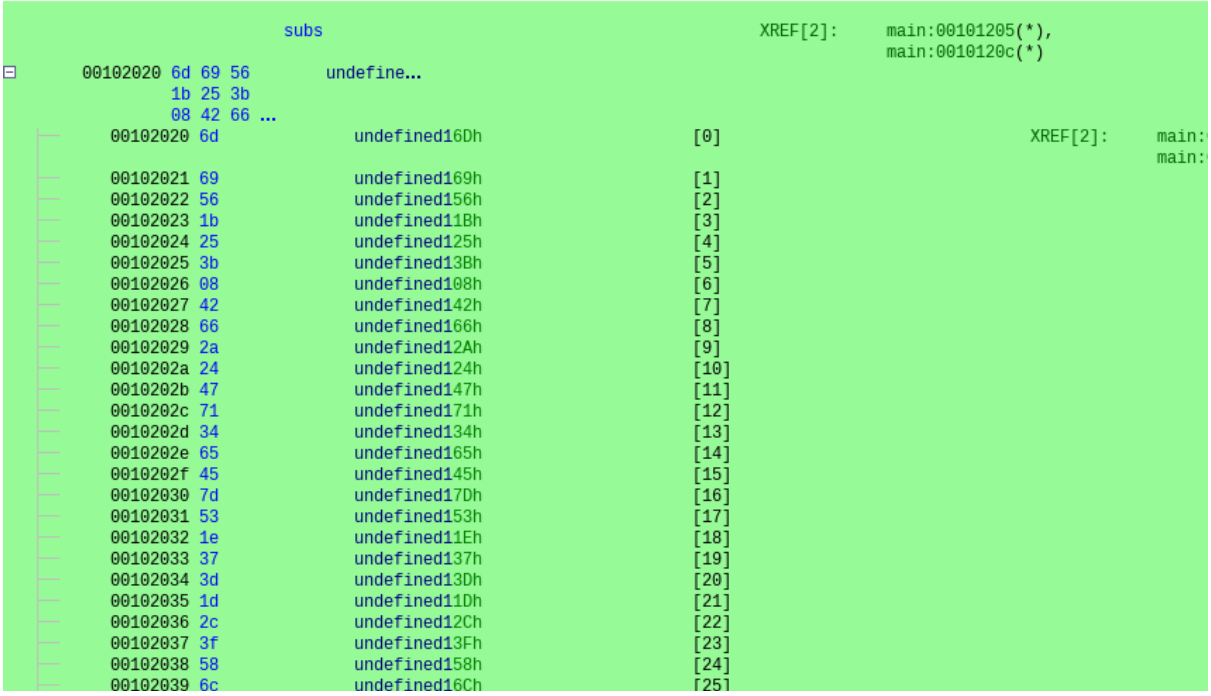
```
00101255 e8 d6 fd        CALL       <EXTERNAL>::puts
         ff ff
0010125a eb 10           JMP        LAB_0010126c

                     LAB_0010125c                                    XREF[1
0010125c 90              NOP
0010125d eb 01           JMP        LAB_00101260

                     LAB_0010125f                                    XREF[1
0010125f 90              NOP

                     LAB_00101260                                    XREF[1
00101260 48 8d 3d        LEA        RDI,[s_Curses!_Foiled_again!_Damn..._00
         4f 0e 00 00
00101267 e8 c4 fd        CALL       <EXTERNAL>::puts
         ff ff

                     LAB_0010126c                                    XREF[1
0010126c b8 00 00        MOV        EAX,0x0
         00 00
00101271 48 8b 75 f8     MOV        RSI,qword ptr [RBP + local_10]
00101275 64 48 33        XOR        RSI,qword ptr FS:[0x28]
         34 25 28
         00 00 00
0010127e 74 05           JZ         LAB_00101285
00101280 e8 cb fd        CALL       <EXTERNAL>::__stack_chk_fail
         ff ff
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     LAB_00101285                                    XREF[1
00101285 c9              LEAVE
00101286 c3              RET
00101287 66              ??         66h    f
00101288 0f              ??         0Fh
00101289 1f              ??         1Fh
0010128a 84              ??         84h
0010128b 00              ??         00h
0010128c 00              ??         00h
0010128d 00              ??         00h
0010128e 00              ??         00h
```

```

undefined8 main(void)

{
  size_t sVar1;
  long in_FS_OFFSET;
  int local_50;
  undefined4 local_4c;
  char local_48 [56];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_50 = 0xff;
  fwrite("[input]: ",1,9,stdout);
  fgets(local_48,0x2a,stdin);
  sVar1 = strlen(local_48);
  local_48[sVar1 - 1] = '\0';
  sVar1 = strlen(local_48);
  if (sVar1 == 0x14) {
    for (local_4c = 0; local_48[local_4c] != '\0'; local_4c = local_4c + 1) {
      if (subs[(int)local_48[local_4c]] == __init_libc_mn.2544[local_4c]) {
        local_50 = local_50 + 1;
      }
      else {
        local_50 = 0;
      }
    }
    if (local_50 >> 5 != 0) {
      puts("FLAG IZ OK!");
      goto LAB_0010126c;
    }
  }
  puts("Curses! Foiled again! Damn...");
LAB_0010126c:
  if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
    return 0;
  }
                    /* WARNING: Subroutine does not return */
  __stack_chk_fail();
}
```

We get this code :

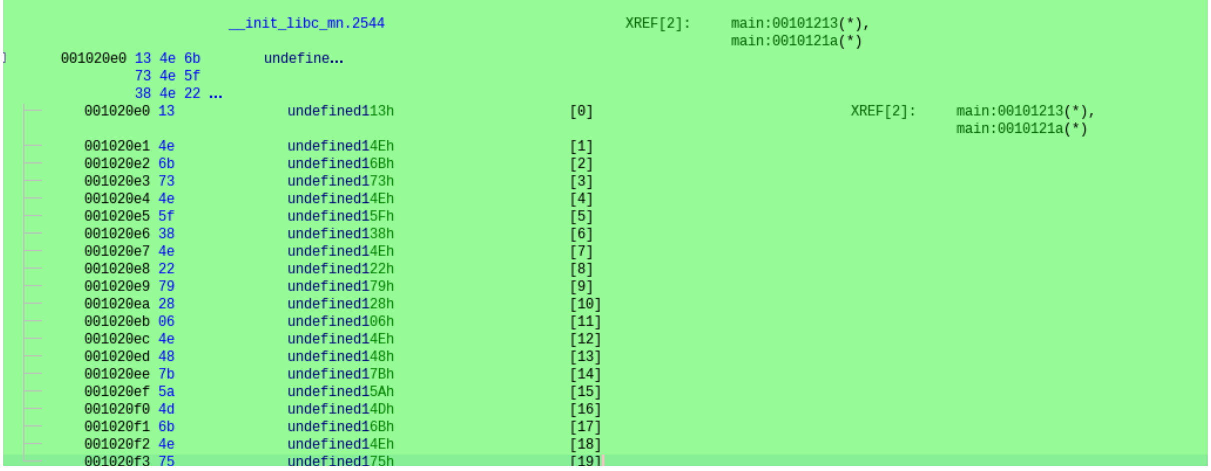
```
undefined8 main(void)

{
  size_t sVar1;
  long in_FS_OFFSET;
  int local_50;
  undefined4 local_4c;
  char local_48 [56];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_50 = 0xff;
  fwrite("[input]: ",1,9,stdout);
  fgets(local_48,0x2a,stdin);
  sVar1 = strlen(local_48);
  local_48[sVar1 – 1] = '\0';
  sVar1 = strlen(local_48);
  if (sVar1 == 0x14) {
    for (local_4c = 0; local_48[local_4c] != '\0'; local_4c = local_4c + 1) {
      if (subs[(int)local_48[local_4c]] == __init_libc_mn.2544[local_4c]) {
        local_50 = local_50 + 1;
      }
      else {
        local_50 = 0;
      }
    }
    if (local_50 >> 5 != 0) {
      puts("FLAG IZ OK!");
      goto LAB_0010126c;
    }
  }
  puts("Curses! Foiled again! Damn...");
LAB_0010126c:
  if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
    return 0;
  }
                    /* WARNING: Subroutine does not return */
  __stack_chk_fail();
}
```

We get the subs values :



```
                             subs                                            XREF[2]:     main:00101205(*),
                                                                                          main:0010120c(*)
        00102020 6d 69 56        undefine...
                 1b 25 3b
                 08 42 66 ...
           00102020 6d           undefined16Dh                  [0]                            XREF[2]:     main:
                                                                                                            main:
           00102021 69           undefined169h                  [1]
           00102022 56           undefined156h                  [2]
           00102023 1b           undefined11Bh                  [3]
           00102024 25           undefined125h                  [4]
           00102025 3b           undefined13Bh                  [5]
           00102026 08           undefined108h                  [6]
           00102027 42           undefined142h                  [7]
           00102028 66           undefined166h                  [8]
           00102029 2a           undefined12Ah                  [9]
           0010202a 24           undefined124h                  [10]
           0010202b 47           undefined147h                  [11]
           0010202c 71           undefined171h                  [12]
           0010202d 34           undefined134h                  [13]
           0010202e 65           undefined165h                  [14]
           0010202f 45           undefined145h                  [15]
           00102030 7d           undefined17Dh                  [16]
           00102031 53           undefined153h                  [17]
           00102032 1e           undefined11Eh                  [18]
           00102033 37           undefined137h                  [19]
           00102034 3d           undefined13Dh                  [20]
           00102035 1d           undefined11Dh                  [21]
           00102036 2c           undefined12Ch                  [22]
           00102037 3f           undefined13Fh                  [23]
           00102038 58           undefined158h                  [24]
           00102039 6c           undefined16Ch                  [25]
```

We search for __init_libc_mn.2544 values :



```
                             __init_libc_mn.2544                             XREF[2]:     main:00101213(*),
                                                                                          main:0010121a(*)
        001020e0 13 4e 6b        undefine...
                 73 4e 5f
                 38 4e 22 ...
           001020e0 13           undefined113h                  [0]                            XREF[2]:     main:00101213(*),
                                                                                                            main:0010121a(*)
           001020e1 4e           undefined14Eh                  [1]
           001020e2 6b           undefined16Bh                  [2]
           001020e3 73           undefined173h                  [3]
           001020e4 4e           undefined14Eh                  [4]
           001020e5 5f           undefined15Fh                  [5]
           001020e6 38           undefined138h                  [6]
           001020e7 4e           undefined14Eh                  [7]
           001020e8 22           undefined122h                  [8]
           001020e9 79           undefined179h                  [9]
           001020ea 28           undefined128h                  [10]
           001020eb 06           undefined106h                  [11]
           001020ec 4e           undefined14Eh                  [12]
           001020ed 48           undefined148h                  [13]
           001020ee 7b           undefined17Bh                  [14]
           001020ef 5a           undefined15Ah                  [15]
           001020f0 4d           undefined14Dh                  [16]
           001020f1 6b           undefined16Bh                  [17]
           001020f2 4e           undefined14Eh                  [18]
           001020f3 75           undefined175h                  [19]
```

After that we map with the ASCII values :

Index 0: N (0x4e)

Index 1: k (0x6b)

Index 2: V (0x56)

Index 3: (Control Char)

Index 4: % (0x25)

Index 5: ; (0x3b)

Index 6: (Control Char)

Index 7: B (0x42)

Index 8: f (0x66)

Index 9: * (0x2a)

Index 10: $ (0x24)

Index 11: G (0x47)

Index 12: q (0x71)

Index 13: 4 (0x34)

Index 14: e (0x65)

Index 15: E (0x45)

Index 16: } (0x7d)

Index 17: S (0x53)

Index 18: (Control Char)

Index 19: 7 (0x37)

3 . Python Search

We created a Python script named `find_flag.py` to reconstruct the potential flag. The script initializes an empty string for the flag and iterates through each value in the `reference` array.

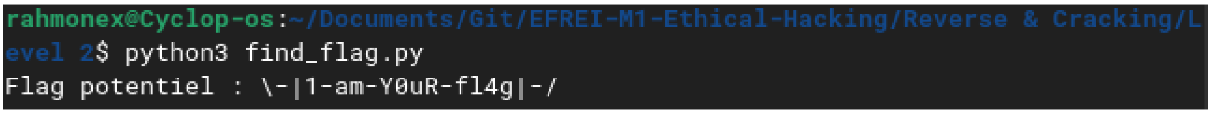
```
# Tableaux extraits de Ghidra
subs = [
    0x6d, 0x69, 0x56, 0x1b, 0x25, 0x3b, 0x08, 0x42, 0x66, 0x2a,
    0x24, 0x47, 0x71, 0x34, 0x65, 0x45, 0x7d, 0x53, 0x1e, 0x37,
    0x3d, 0x1d, 0x2c, 0x3f, 0x58, 0x6c, 0x19, 0x2f, 0x3c, 0x0c,
    0x6a, 0x7f, 0x0d, 0x12, 0x43, 0x70, 0x41, 0x72, 0x51, 0x4f,
    0x21, 0x30, 0x4b, 0x40, 0x16, 0x4e, 0x60, 0x75, 0x79, 0x73,
    0x0a, 0x44, 0x5a, 0x17, 0x0e, 0x67, 0x4a, 0x49, 0x23, 0x09,
    0x2e, 0x33, 0x55, 0x4c, 0x32, 0x63, 0x05, 0x50, 0x03, 0x77,
    0x6f, 0x5c, 0x5d, 0x10, 0x36, 0x78, 0x2d, 0x7e, 0x35, 0x1a,
    0x01, 0x11, 0x06, 0x1c, 0x1f, 0x2b, 0x54, 0x61, 0x14, 0x22,
    0x39, 0x04, 0x13, 0x74, 0x18, 0x7a, 0x20, 0x5f, 0x76, 0x7c,
    0x02, 0x46, 0x48, 0x4d, 0x57, 0x31, 0x59, 0x29, 0x7b, 0x38,
    0x07, 0x68, 0x0f, 0x52, 0x27, 0x26, 0x62, 0x28, 0x0b, 0x5b,
    0x15, 0x00, 0x5e, 0x64, 0x6b, 0x6e, 0x3a, 0x3e
]

reference = [
    0x13, 0x4e, 0x6b, 0x73, 0x4e, 0x5f, 0x38, 0x4e, 0x22, 0x79,
    0x28, 0x06, 0x4e, 0x48, 0x7b, 0x5a, 0x4d, 0x6b, 0x4e, 0x75
]

# Reconstruire le flag
flag = ""
for i in range(len(reference)):
    for c in range(256):  # Parcourt tous les caractères possibles
        if subs[c] == reference[i]:
            flag += chr(c)
            break

print("Flag potentiel :", flag)
```

For each expected value in the reference array, the script checks all possible ASCII characters (0 to 255) to find a character from the subs array that matches the current reference value. When a match is found, the corresponding ASCII character is appended to the flag string.

After processing all values, the script prints the reconstructed potential flag. This method effectively reverses the transformation logic used in the original binary, allowing us to identify the input characters required for validation.

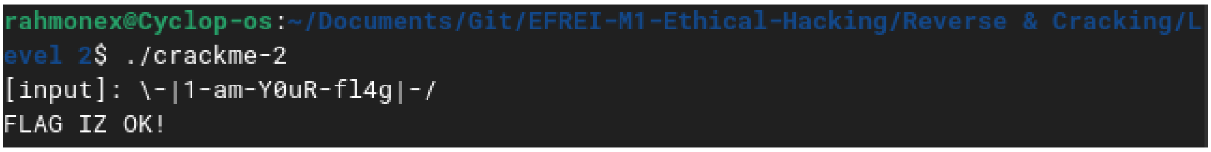
4 . Output & Testing

Now we execute the python script and we test if the flag is correct :



```
rahmonex@Cyclop-os:~/Documents/Git/EFREI-M1-Ethical-Hacking/Reverse & Cracking/Level 2$ python3 find_flag.py
Flag potentiel : \-|1-am-Y0uR-fl4g|-/
```

And we proceed with testing :



```
rahmonex@Cyclop-os:~/Documents/Git/EFREI-M1-Ethical-Hacking/Reverse & Cracking/Level 2$ ./crackme-2
[input]: \-|1-am-Y0uR-fl4g|-/
FLAG IZ OK!
```

So the Flag is : **\-|1-am-Y0uR-fl4g|-/**