# Writeup: REasy

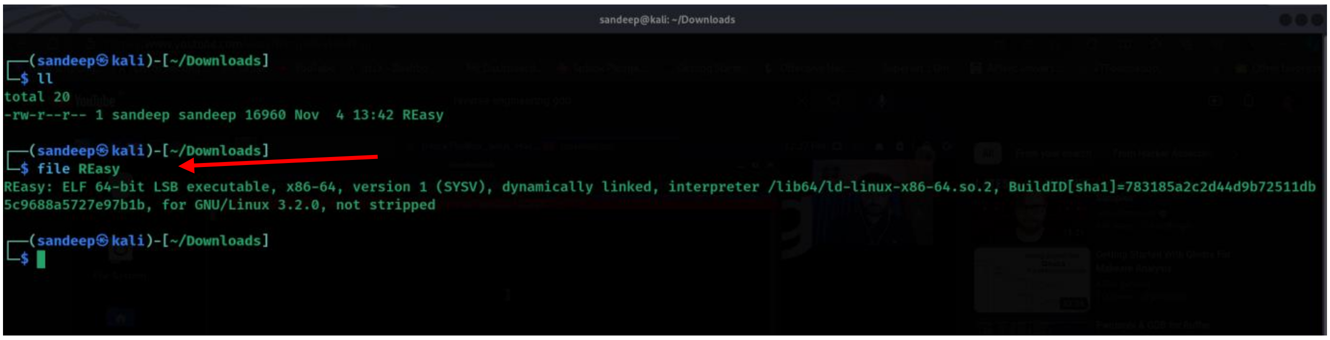
Description: Basics of Reversing

Points: 50

File: REasy

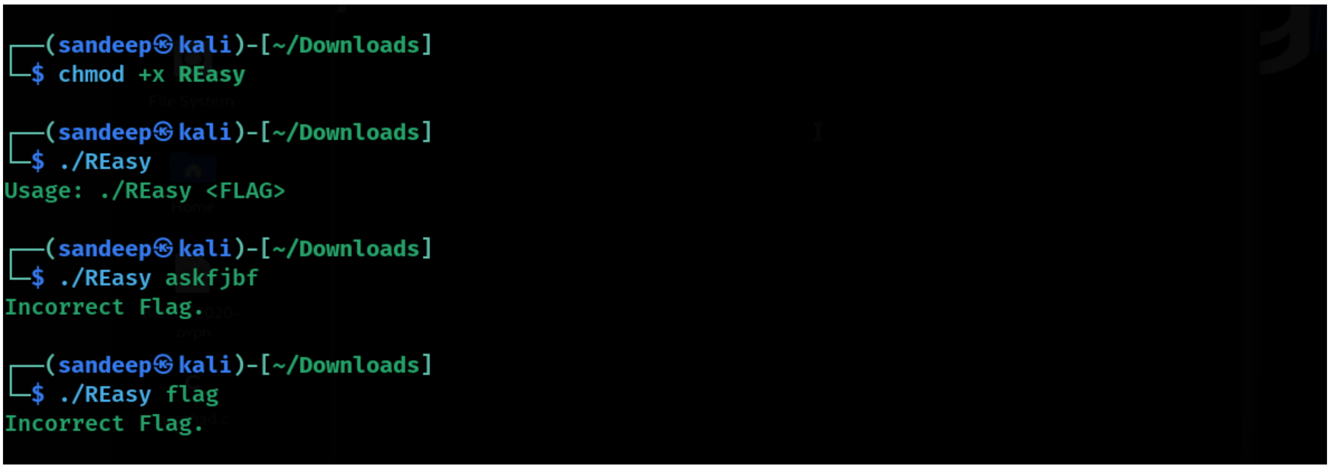
Team: **Kaliyug_x64**

- Initial checks to know what kind of binary it is.



```
┌──(sandeep㉿kali)-[~/Downloads]
└─$ ll
total 20
-rw-r--r-- 1 sandeep sandeep 16960 Nov  4 13:42 REasy

┌──(sandeep㉿kali)-[~/Downloads]
└─$ file REasy
REasy: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=783185a2c2d44d9b72511db
5c9688a5727e97b1b, for GNU/Linux 3.2.0, not stripped

┌──(sandeep㉿kali)-[~/Downloads]
└─$ 
```

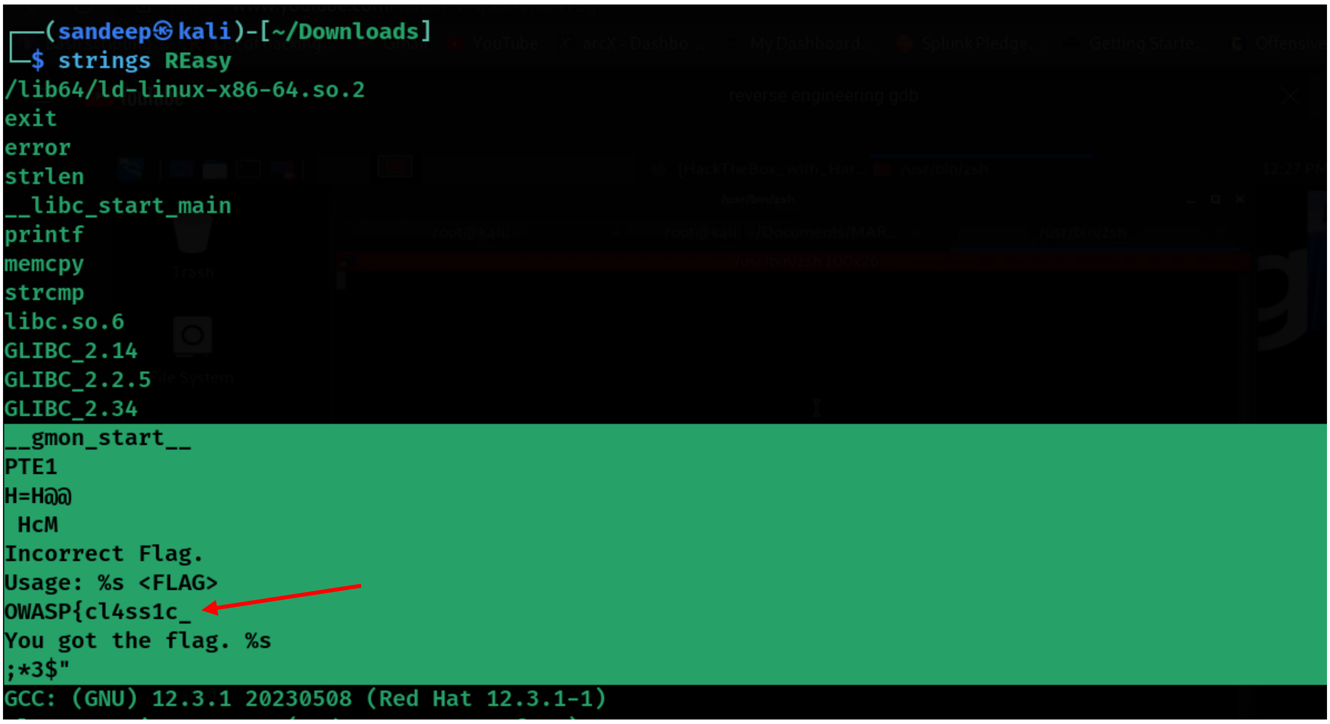
- Running the binary to see what it does



```
┌──(sandeep㉿kali)-[~/Downloads]
└─$ chmod +x REasy

┌──(sandeep㉿kali)-[~/Downloads]
└─$ ./REasy
Usage: ./REasy <FLAG>

┌──(sandeep㉿kali)-[~/Downloads]
└─$ ./REasy askfjbf
Incorrect Flag.

┌──(sandeep㉿kali)-[~/Downloads]
└─$ ./REasy flag
Incorrect Flag.
```

It is taking flag as an argument and checking whether it is correct or not.

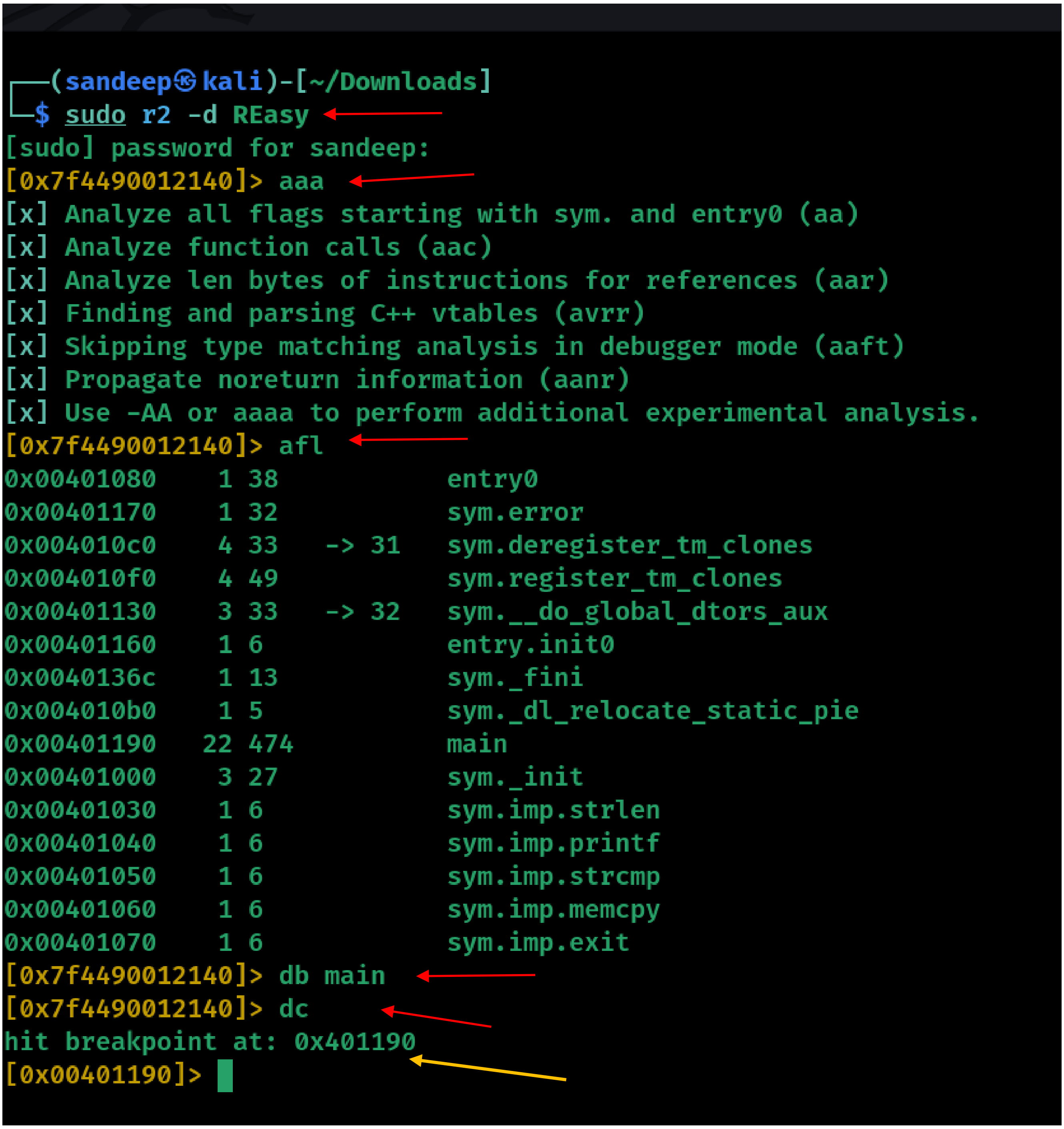
- Doing strings command on binary to see whether we can get flag or not.



```
┌──(sandeep㉿kali)-[~/Downloads]
└─$ strings REasy
/lib64/ld-linux-x86-64.so.2
exit
error
strlen
__libc_start_main
printf
memcpy
strcmp
libc.so.6
GLIBC_2.14
GLIBC_2.2.5
GLIBC_2.34
__gmon_start__
PTE1
H=H@@
 HcM
Incorrect Flag.
Usage: %s <FLAG>
OWASP{cl4ss1c_
You got the flag. %s
;*3$"
GCC: (GNU) 12.3.1 20230508 (Red Hat 12.3.1-1)
```

We see that there are initial part of flag, but it is incomplete. For now we will note that and move on to find remaining.

- Open the binary in debugger to see what is there in assembly code. I use **radare2**, one can use any debugger

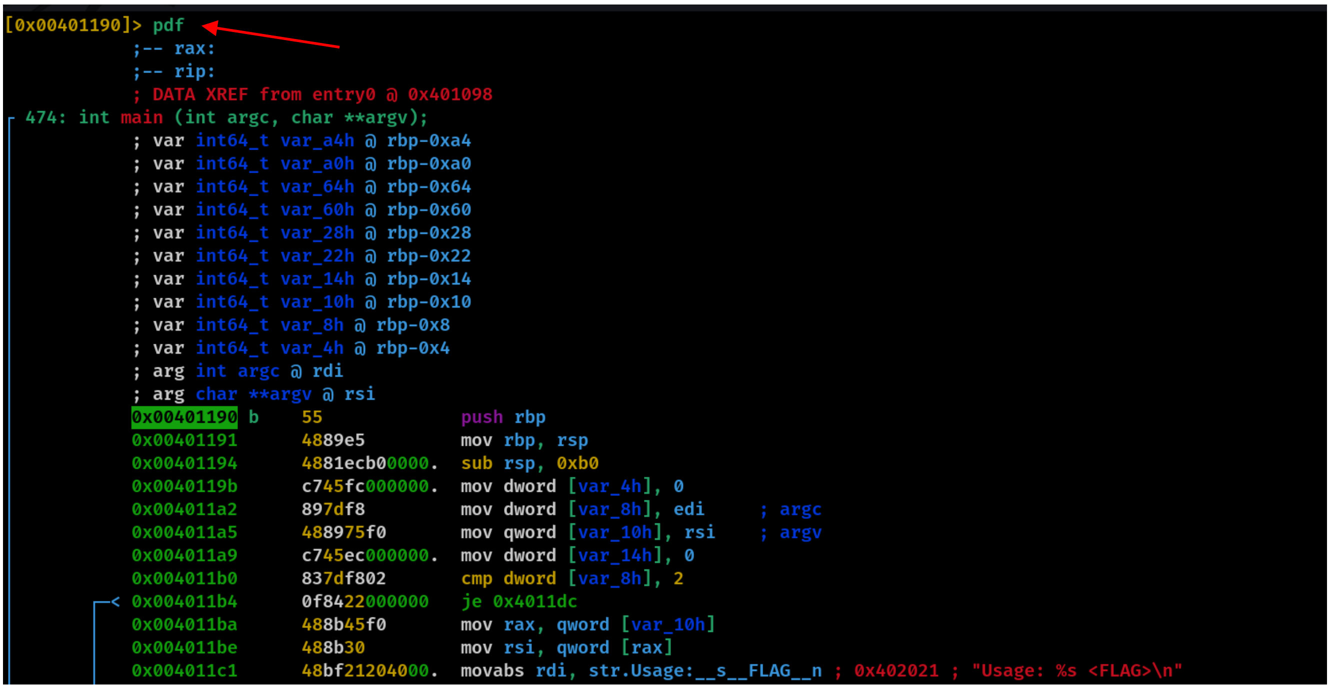


```
┌──(sandeep㉿kali)-[~/Downloads]
└─$ sudo r2 -d REasy
[sudo] password for sandeep:
[0x7f4490012140]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x7f4490012140]> afl
0x00401080    1 38           entry0
0x00401170    1 32           sym.error
0x004010c0    4 33   -> 31   sym.deregister_tm_clones
0x004010f0    4 49           sym.register_tm_clones
0x00401130    3 33   -> 32   sym.__do_global_dtors_aux
0x00401160    1 6            entry.init0
0x0040136c    1 13           sym._fini
0x004010b0    1 5            sym._dl_relocate_static_pie
0x00401190   22 474          main
0x00401000    3 27           sym._init
0x00401030    1 6            sym.imp.strlen
0x00401040    1 6            sym.imp.printf
0x00401050    1 6            sym.imp.strcmp
0x00401060    1 6            sym.imp.memcpy
0x00401070    1 6            sym.imp.exit
[0x7f4490012140]> db main
[0x7f4490012140]> dc
hit breakpoint at: 0x401190
[0x00401190]>
```

| | |
|---|---|
| **> sudo r2 –d REasy** | // command to open the binary in debugger |
| **> aaa** | // command to analysis binary |
| **> afl** | // to list all analysed function |
| **> db main** | // setting break on main function. So it should stop as it reaches to main |
| **> dc** | //to continue the binary run |

We can see it hits to breakpoint at **0x401190**. (Yellow arrow)

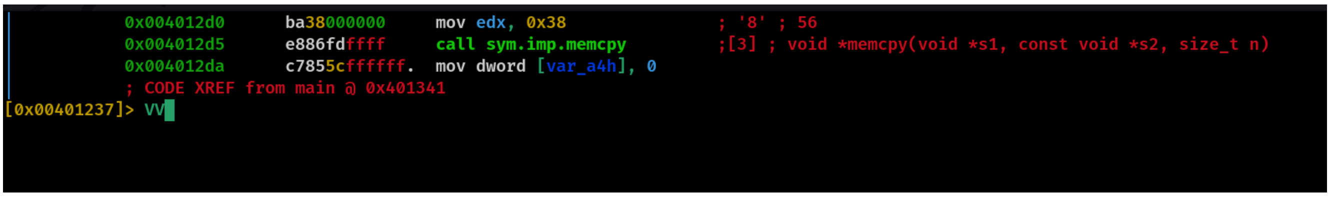
**> pdf** //To see the assembly code code



```
[0x00401190]> pdf
            ;-- rax:
            ;-- rip:
            ; DATA XREF from entry0 @ 0x401098
┌ 474: int main (int argc, char **argv);
│           ; var int64_t var_a4h @ rbp-0xa4
│           ; var int64_t var_a0h @ rbp-0xa0
│           ; var int64_t var_64h @ rbp-0x64
│           ; var int64_t var_60h @ rbp-0x60
│           ; var int64_t var_28h @ rbp-0x28
│           ; var int64_t var_22h @ rbp-0x22
│           ; var int64_t var_14h @ rbp-0x14
│           ; var int64_t var_10h @ rbp-0x10
│           ; var int64_t var_8h @ rbp-0x8
│           ; var int64_t var_4h @ rbp-0x4
│           ; arg int argc @ rdi
│           ; arg char **argv @ rsi
│           0x00401190 b     55             push rbp
│           0x00401191       4889e5         mov rbp, rsp
│           0x00401194       4881ecb00000.  sub rsp, 0xb0
│           0x0040119b       c745fc000000.  mov dword [var_4h], 0
│           0x004011a2       897df8         mov dword [var_8h], edi      ; argc
│           0x004011a5       488975f0       mov qword [var_10h], rsi     ; argv
│           0x004011a9       c745ec000000.  mov dword [var_14h], 0
│           0x004011b0       837df802       cmp dword [var_8h], 2
│       ┌─< 0x004011b4       0f8422000000   je 0x4011dc
│       │   0x004011ba       488b45f0       mov rax, qword [var_10h]
│       │   0x004011be       488b30         mov rsi, qword [rax]
│       │   0x004011c1       48bf21204000.  movabs rdi, str.Usage:__s__FLAG__n ; 0x402021 ; "Usage: %s <FLAG>\n"
```

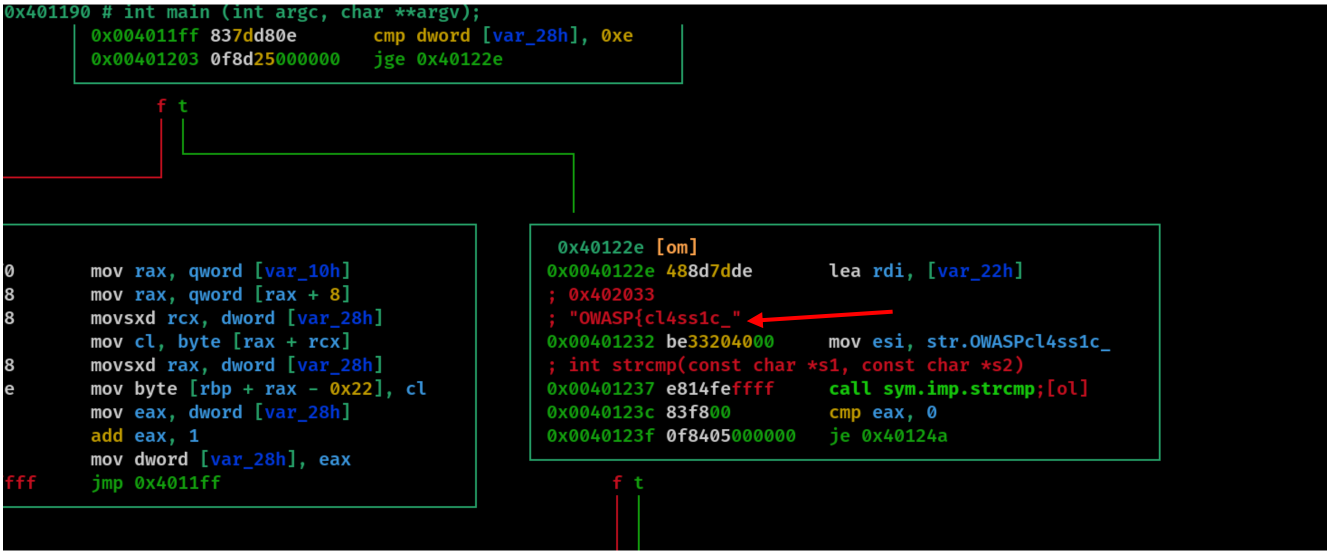
➢ Changing the view of debugger to see flow and assembly code

**> VV** // to change the view



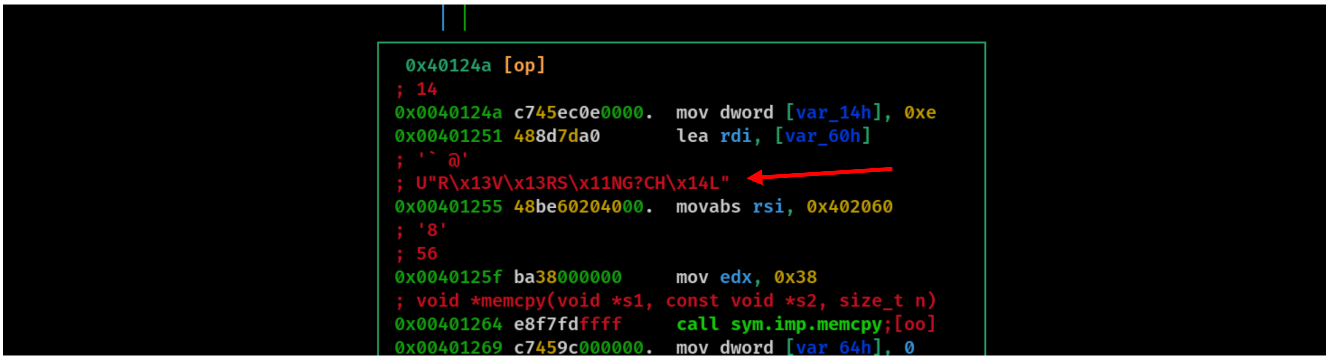
```
│          0x004012d0      ba38000000     mov edx, 0x38             ; '8' ; 56
│          0x004012d5      e886fdffff     call sym.imp.memcpy       ;[3] ; void *memcpy(void *s1, const void *s2, size_t n)
│          0x004012da      c7855cffffff.  mov dword [var_a4h], 0
           ; CODE XREF from main @ 0x401341
[0x00401237]> VV
```

➢ You will see the graph mode, Scroll down we will see "OWASP{cl4ss1c_"



```
0x401190 # int main (int argc, char **argv);
      │ 0x004011ff 837dd80e        cmp dword [var_28h], 0xe       │
      │ 0x00401203 0f8d25000000    jge 0x40122e                   │
      └───────────────────────────────────────────────────────────┘
                f t
                │ │
────────────────┘ └───────────────────────────────┐
                                                   │

────────────────────────────────────────────┐   ┌─────────────────────────────────────────────────────┐
                                            │   │ 0x40122e [om]                                       │
0       mov rax, qword [var_10h]            │   │0x0040122e 488d7dde       lea rdi, [var_22h]         │
8       mov rax, qword [rax + 8]            │   │; 0x402033                                           │
8       movsxd rcx, dword [var_28h]         │   │; "OWASP{cl4ss1c_"                                   │
        mov cl, byte [rax + rcx]            │   │0x00401232 be33204000     mov esi, str.OWASPcl4ss1c_ │
8       movsxd rax, dword [var_28h]         │   │; int strcmp(const char *s1, const char *s2)         │
e       mov byte [rbp + rax - 0x22], cl     │   │0x00401237 e814feffff     call sym.imp.strcmp;[ol]   │
        mov eax, dword [var_28h]            │   │0x0040123c 83f800         cmp eax, 0                 │
        add eax, 1                          │   │0x0040123f 0f8405000000   je 0x40124a                │
        mov dword [var_28h], eax            │   └─────────────────────────────────────────────────────┘
fff     jmp 0x4011ff                        │                f t
────────────────────────────────────────────┘                │ │
                                                             │ │
```

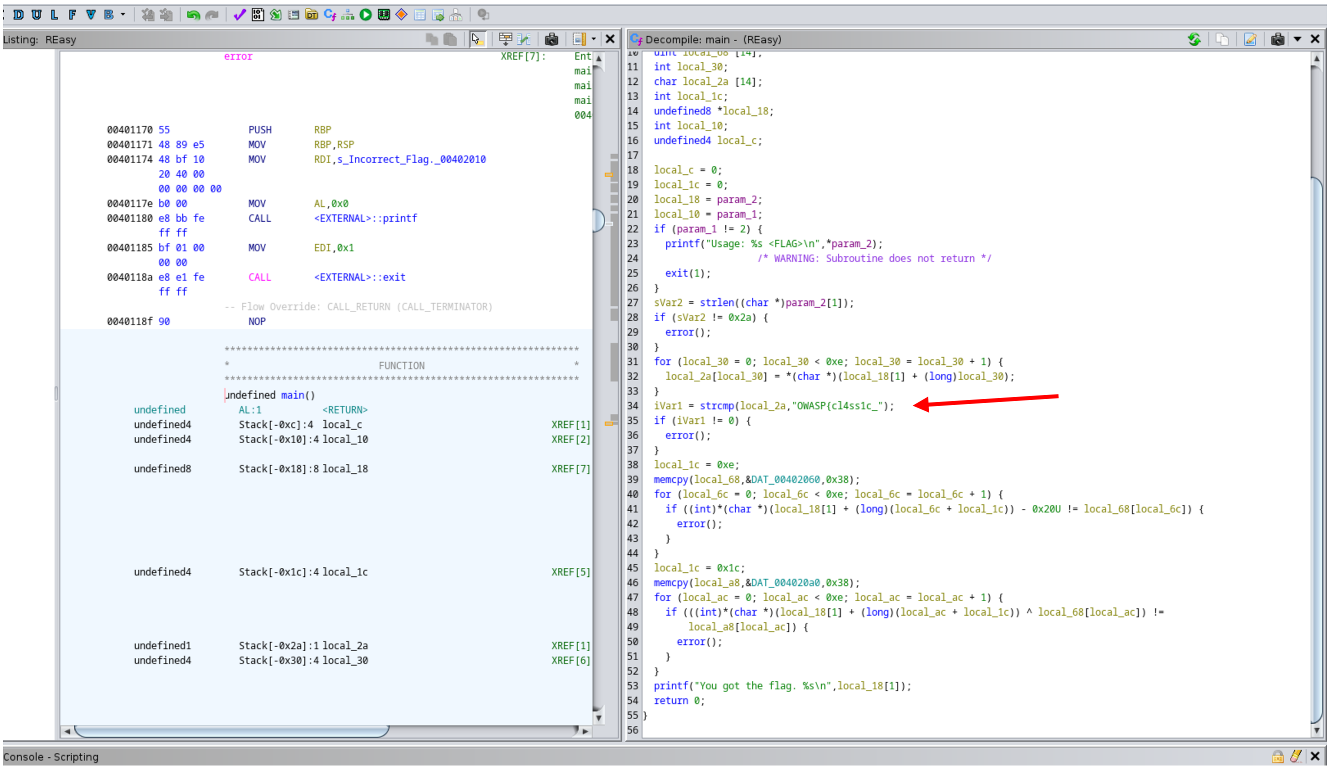
➢ Scroll little bit more and, one more string is their "**U"R\x13V\x13RS\x11NG?CN\x14L**"



```
                                      │ │
                        ┌──────────────────────────────────────────────────┐
                        │ 0x40124a [op]                                    │
                        │; 14                                              │
                        │0x0040124a c745ec0e0000.  mov dword [var_14h], 0xe│
                        │0x00401251 488d7da0       lea rdi, [var_60h]      │
                        │; '` @'                                           │
                        │; U"R\x13V\x13RS\x11NG?CH\x14L"                    │
                        │0x00401255 48be60204000.  movabs rsi, 0x402060    │
                        │; '8'                                             │
                        │; 56                                              │
                        │0x0040125f ba38000000     mov edx, 0x38           │
                        │; void *memcpy(void *s1, const void *s2, size_t n)│
                        │0x00401264 e8f7fdffff     call sym.imp.memcpy;[oo]│
                        │0x00401269 c7459c000000.  mov dword [var_64h], 0  │
```

- If "\x1" is in between every character. If we remove that, we get "**R3V3RS1NG?CH4L**". "**Reversing**" word and initials of "**challenge**" word. Mixed with numbers.
  Now our flag is "**OWASP{cl4ss1c_r3v3rs1ng?ch4l**".

- As I didn't find anything I open the binary in **Ghidra**.

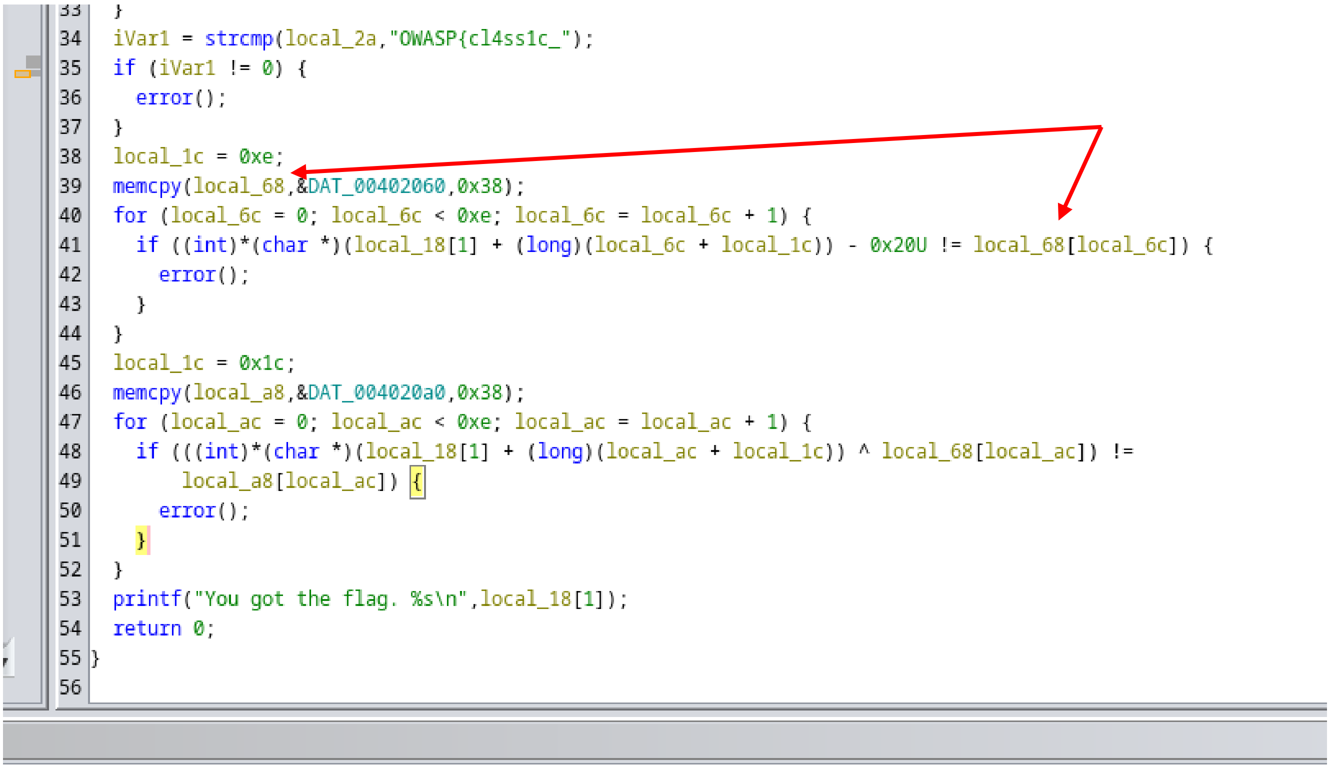


```
                             error                                        XREF[7]:     Ent
                                                                                       mai
                                                                                       mai
                                                                                       mai
                                                                                       004
        00401170 55              PUSH       RBP
        00401171 48 89 e5        MOV        RBP,RSP
        00401174 48 bf 10        MOV        RDI,s_Incorrect_Flag._00402010
                 20 40 00
                 00 00 00 00
        0040117e b0 00           MOV        AL,0x0
        00401180 e8 bb fe        CALL       <EXTERNAL>::printf
                 ff ff
        00401185 bf 01 00        MOV        EDI,0x1
                 00 00
        0040118a e8 e1 fe        CALL       <EXTERNAL>::exit
                 ff ff
                             -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
        0040118f 90              NOP

                             **************************************************************
                             *                          FUNCTION                          *
                             **************************************************************
                             undefined main()
             undefined         AL:1           <RETURN>
             undefined4        Stack[-0xc]:4  local_c                              XREF[1]
             undefined4        Stack[-0x10]:4 local_10                             XREF[2]

             undefined8        Stack[-0x18]:8 local_18                             XREF[7]

             undefined4        Stack[-0x1c]:4 local_1c                             XREF[5]

             undefined1        Stack[-0x2a]:1 local_2a                             XREF[1]
             undefined4        Stack[-0x30]:4 local_30                             XREF[6]
```

```
10  uint local_68 [14];
11  int local_30;
12  char local_2a [14];
13  int local_1c;
14  undefined8 *local_18;
15  int local_10;
16  undefined4 local_c;
17
18  local_c = 0;
19  local_1c = 0;
20  local_18 = param_2;
21  local_10 = param_1;
22  if (param_1 != 2) {
23    printf("Usage: %s <FLAG>\n",*param_2);
24                      /* WARNING: Subroutine does not return */
25    exit(1);
26  }
27  sVar2 = strlen((char *)param_2[1]);
28  if (sVar2 != 0x2a) {
29    error();
30  }
31  for (local_30 = 0; local_30 < 0xe; local_30 = local_30 + 1) {
32    local_2a[local_30] = *(char *)(local_18[1] + (long)local_30);
33  }
34  iVar1 = strcmp(local_2a,"OWASP{cl4ss1c_");
35  if (iVar1 != 0) {
36    error();
37  }
38  local_1c = 0xe;
39  memcpy(local_68,&DAT_00402060,0x38);
40  for (local_6c = 0; local_6c < 0xe; local_6c = local_6c + 1) {
41    if ((int)*(char *)(local_18[1] + (long)(local_6c + local_1c)) - 0x20U != local_68[local_6c]) {
42      error();
43    }
44  }
45  local_1c = 0x1c;
46  memcpy(local_a8,&DAT_004020a0,0x38);
47  for (local_ac = 0; local_ac < 0xe; local_ac = local_ac + 1) {
48    if (((int)*(char *)(local_18[1] + (long)(local_ac + local_1c)) ^ local_68[local_ac]) !=
49        local_a8[local_ac]) {
50      error();
51    }
52  }
53  printf("You got the flag. %s\n",local_18[1]);
54  return 0;
55 }
56
```

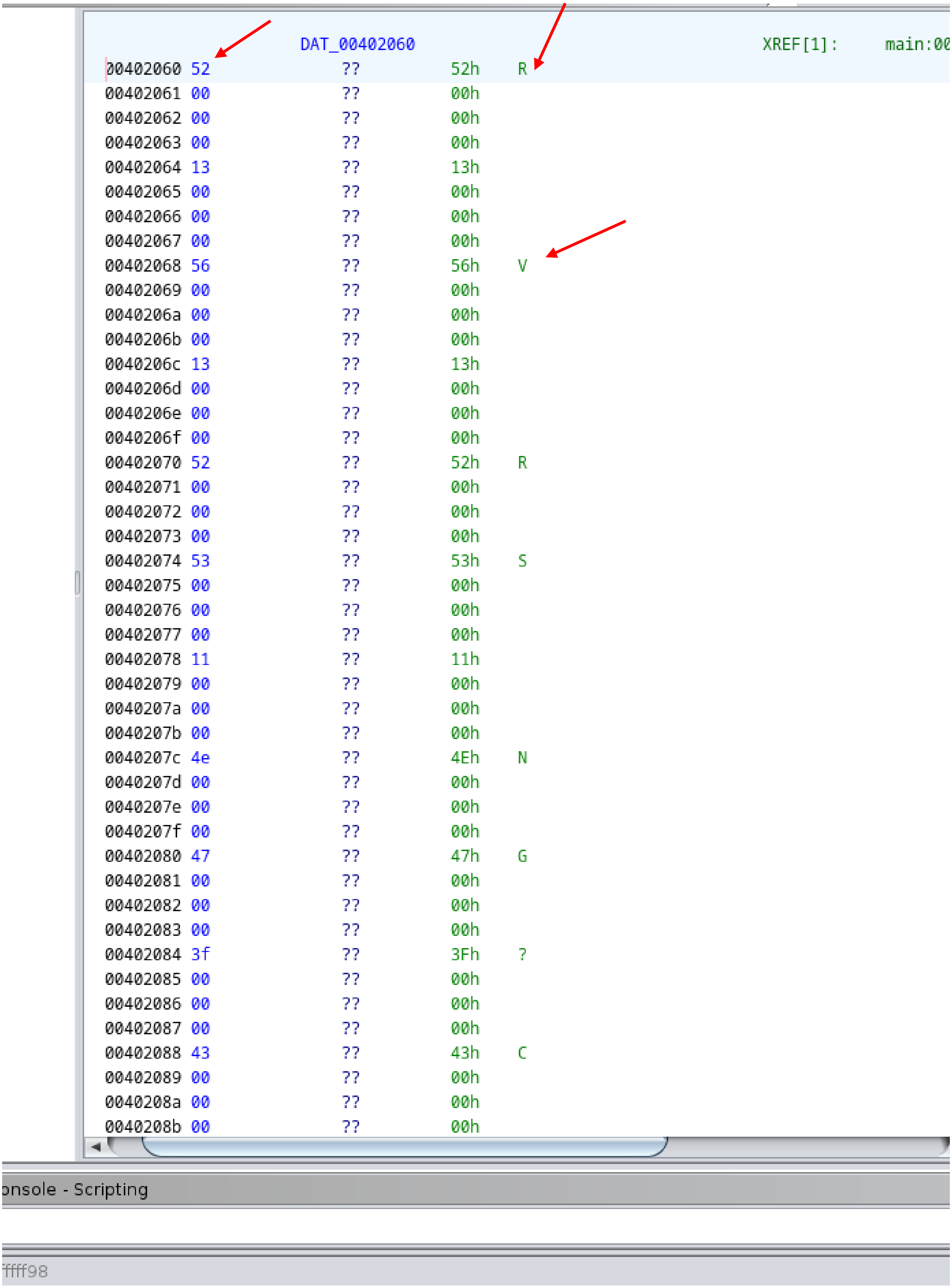
After analysing we see that code is running some checks on argument. On line 34 (red arrow) it is checking for string "**OWASP{cl4ss1c_**" in argument.

- On line 39. We see it is coping "**DAT_00402060**" in variable and using it in condition on line 41 for comparing.



```
33  }
34  iVar1 = strcmp(local_2a,"OWASP{cl4ss1c_");
35  if (iVar1 != 0) {
36    error();
37  }
38  local_1c = 0xe;
39  memcpy(local_68,&DAT_00402060,0x38);
40  for (local_6c = 0; local_6c < 0xe; local_6c = local_6c + 1) {
41    if ((int)*(char *)(local_18[1] + (long)(local_6c + local_1c)) - 0x20U != local_68[local_6c]) {
42      error();
43    }
44  }
45  local_1c = 0x1c;
46  memcpy(local_a8,&DAT_004020a0,0x38);
47  for (local_ac = 0; local_ac < 0xe; local_ac = local_ac + 1) {
48    if (((int)*(char *)(local_18[1] + (long)(local_ac + local_1c)) ^ local_68[local_ac]) !=
49        local_a8[local_ac]) {
50      error();
51    }
52  }
53  printf("You got the flag. %s\n",local_18[1]);
54  return 0;
55 }
56
```

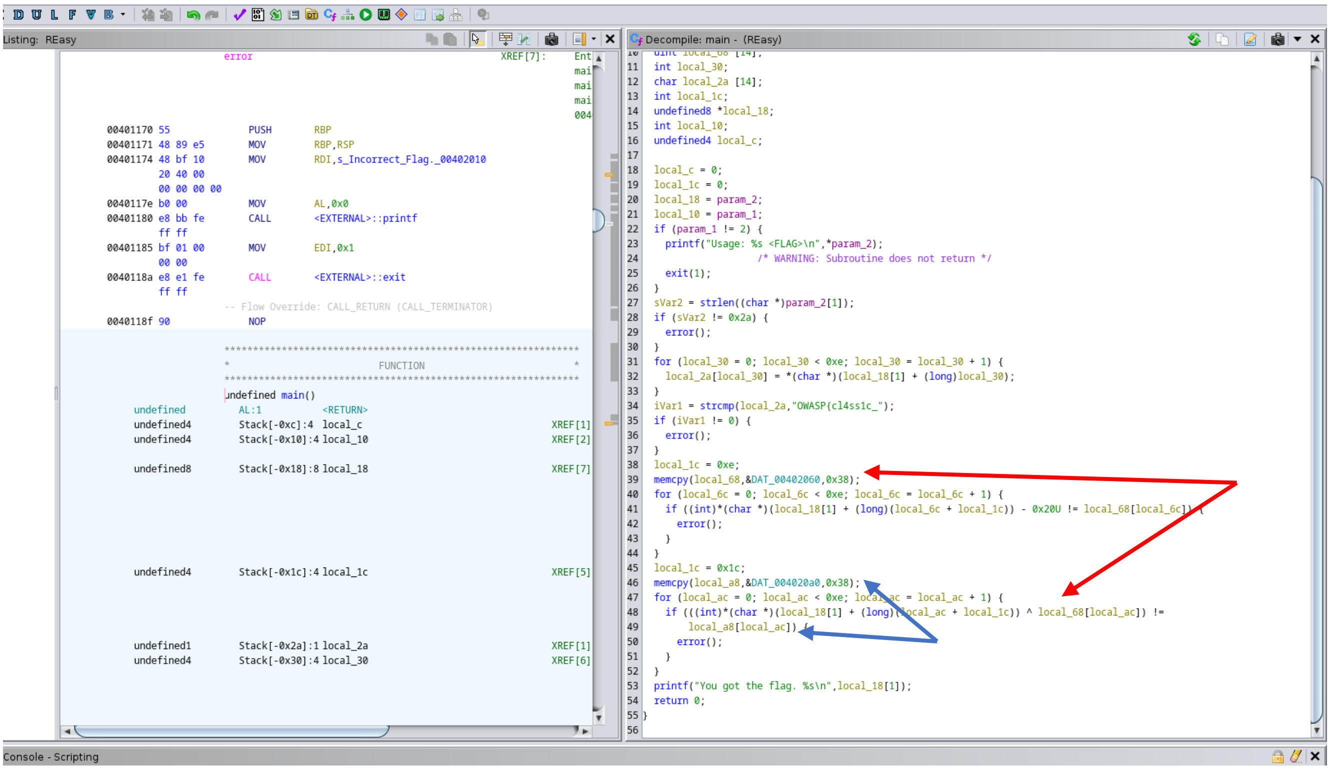
- We need to check the values in “**DAT_00402060**”



```
                    DAT_00402060                                       XREF[1]:     main:00
00402060 52         ??          52h    R
00402061 00         ??          00h
00402062 00         ??          00h
00402063 00         ??          00h
00402064 13         ??          13h
00402065 00         ??          00h
00402066 00         ??          00h
00402067 00         ??          00h
00402068 56         ??          56h    V
00402069 00         ??          00h
0040206a 00         ??          00h
0040206b 00         ??          00h
0040206c 13         ??          13h
0040206d 00         ??          00h
0040206e 00         ??          00h
0040206f 00         ??          00h
00402070 52         ??          52h    R
00402071 00         ??          00h
00402072 00         ??          00h
00402073 00         ??          00h
00402074 53         ??          53h    S
00402075 00         ??          00h
00402076 00         ??          00h
00402077 00         ??          00h
00402078 11         ??          11h
00402079 00         ??          00h
0040207a 00         ??          00h
0040207b 00         ??          00h
0040207c 4e         ??          4Eh    N
0040207d 00         ??          00h
0040207e 00         ??          00h
0040207f 00         ??          00h
00402080 47         ??          47h    G
00402081 00         ??          00h
00402082 00         ??          00h
00402083 00         ??          00h
00402084 3f         ??          3Fh    ?
00402085 00         ??          00h
00402086 00         ??          00h
00402087 00         ??          00h
00402088 43         ??          43h    C
00402089 00         ??          00h
0040208a 00         ??          00h
0040208b 00         ??          00h
```

We got the same word “Reversing” but it missing some characters.

- On line 46 we see that again it is coping from stack “**DAT_004020a0**” to “**local_a8**”.



Listing: REasy

```
                              error                                    XREF[7]:     Ent
                                                                                    mai
                                                                                    mai
                                                                                    mai
                                                                                    004
00401170 55              PUSH       RBP
00401171 48 89 e5        MOV        RBP,RSP
00401174 48 bf 10        MOV        RDI,s_Incorrect_Flag._00402010
         20 40 00
         00 00 00 00
0040117e b0 00           MOV        AL,0x0
00401180 e8 bb fe        CALL       <EXTERNAL>::printf
         ff ff
00401185 bf 01 00        MOV        EDI,0x1
         00 00
0040118a e8 e1 fe        CALL       <EXTERNAL>::exit
         ff ff
                      -- Flow Override: CALL_RETURN (CALL_TERMINATOR)
0040118f 90              NOP

                      **************************************************************
                      *                          FUNCTION                          *
                      **************************************************************
                      undefined main()
     undefined          AL:1            <RETURN>
     undefined4         Stack[-0xc]:4   local_c                            XREF[1]
     undefined4         Stack[-0x10]:4  local_10                           XREF[2]

     undefined8         Stack[-0x18]:8  local_18                           XREF[7]


     undefined4         Stack[-0x1c]:4  local_1c                           XREF[5]


     undefined1         Stack[-0x2a]:1  local_2a                           XREF[1]
     undefined4         Stack[-0x30]:4  local_30                           XREF[6]
```

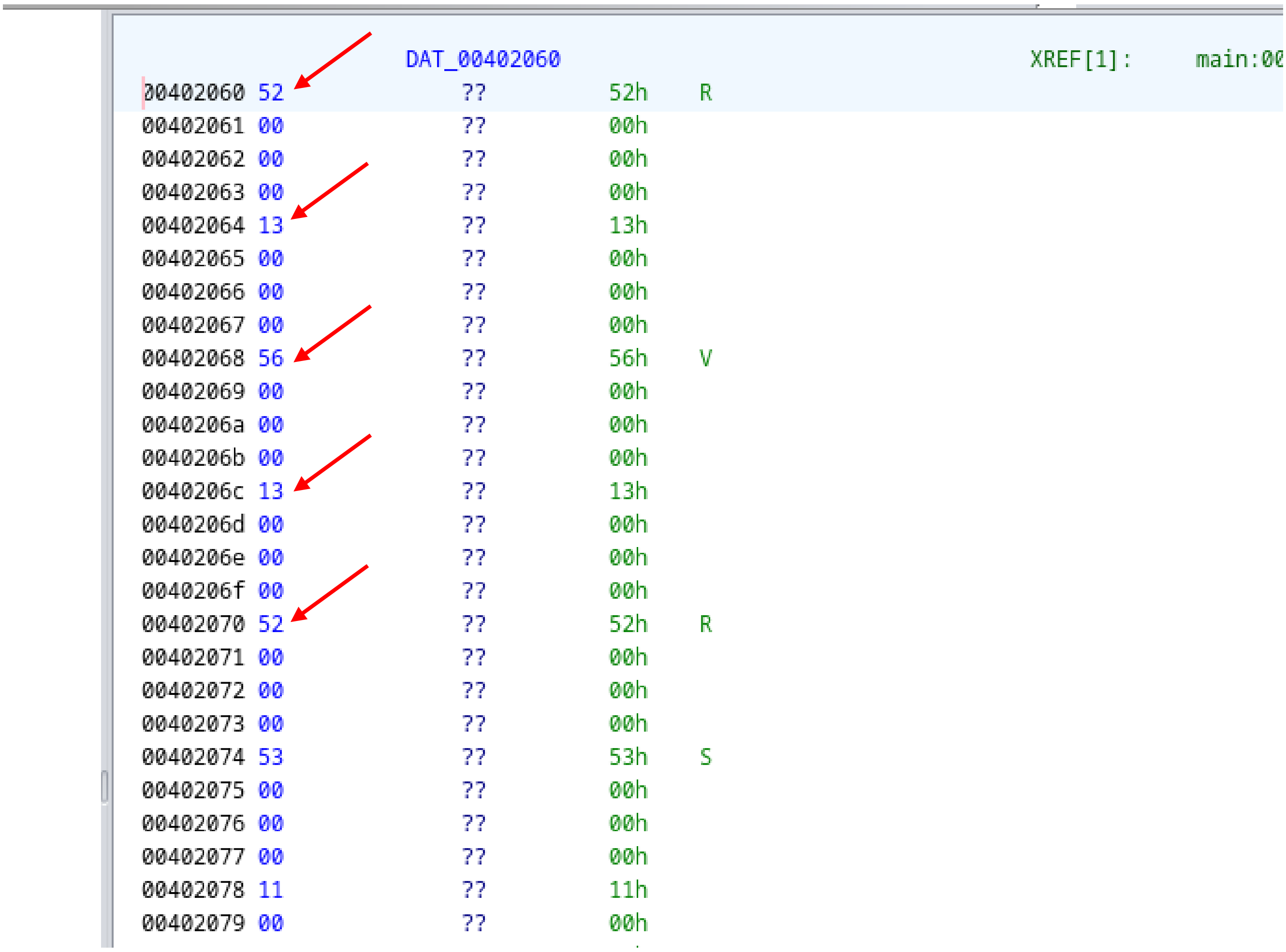
Decompile: main - (REasy)

```
10    uint local_68 [14];
11    int local_30;
12    char local_2a [14];
13    int local_1c;
14    undefined8 *local_18;
15    int local_10;
16    undefined4 local_c;
17  
18    local_c = 0;
19    local_1c = 0;
20    local_18 = param_2;
21    local_10 = param_1;
22    if (param_1 != 2) {
23      printf("Usage: %s <FLAG>\n",*param_2);
24                      /* WARNING: Subroutine does not return */
25      exit(1);
26    }
27    sVar2 = strlen((char *)param_2[1]);
28    if (sVar2 != 0x2a) {
29      error();
30    }
31    for (local_30 = 0; local_30 < 0xe; local_30 = local_30 + 1) {
32      local_2a[local_30] = *(char *)(local_18[1] + (long)local_30);
33    }
34    iVar1 = strcmp(local_2a,"OWASP{cl4ss1c_");
35    if (iVar1 != 0) {
36      error();
37    }
38    local_1c = 0xe;
39    memcpy(local_68,&DAT_00402060,0x38);
40    for (local_6c = 0; local_6c < 0xe; local_6c = local_6c + 1) {
41      if ((int)*(char *)(local_18[1] + (long)(local_6c + local_1c)) - 0x20U != local_68[local_6c]) {
42        error();
43      }
44    }
45    local_1c = 0x1c;
46    memcpy(local_a8,&DAT_004020a0,0x38);
47    for (local_ac = 0; local_ac < 0xe; local_ac = local_ac + 1) {
48      if (((int)*(char *)(local_18[1] + (long)(local_ac + local_1c)) ^ local_68[local_ac]) !=
49          local_a8[local_ac]) {
50        error();
51      }
52    }
53    printf("You got the flag. %s\n",local_18[1]);
54    return 0;
55  }
56  
```

Console - Scripting

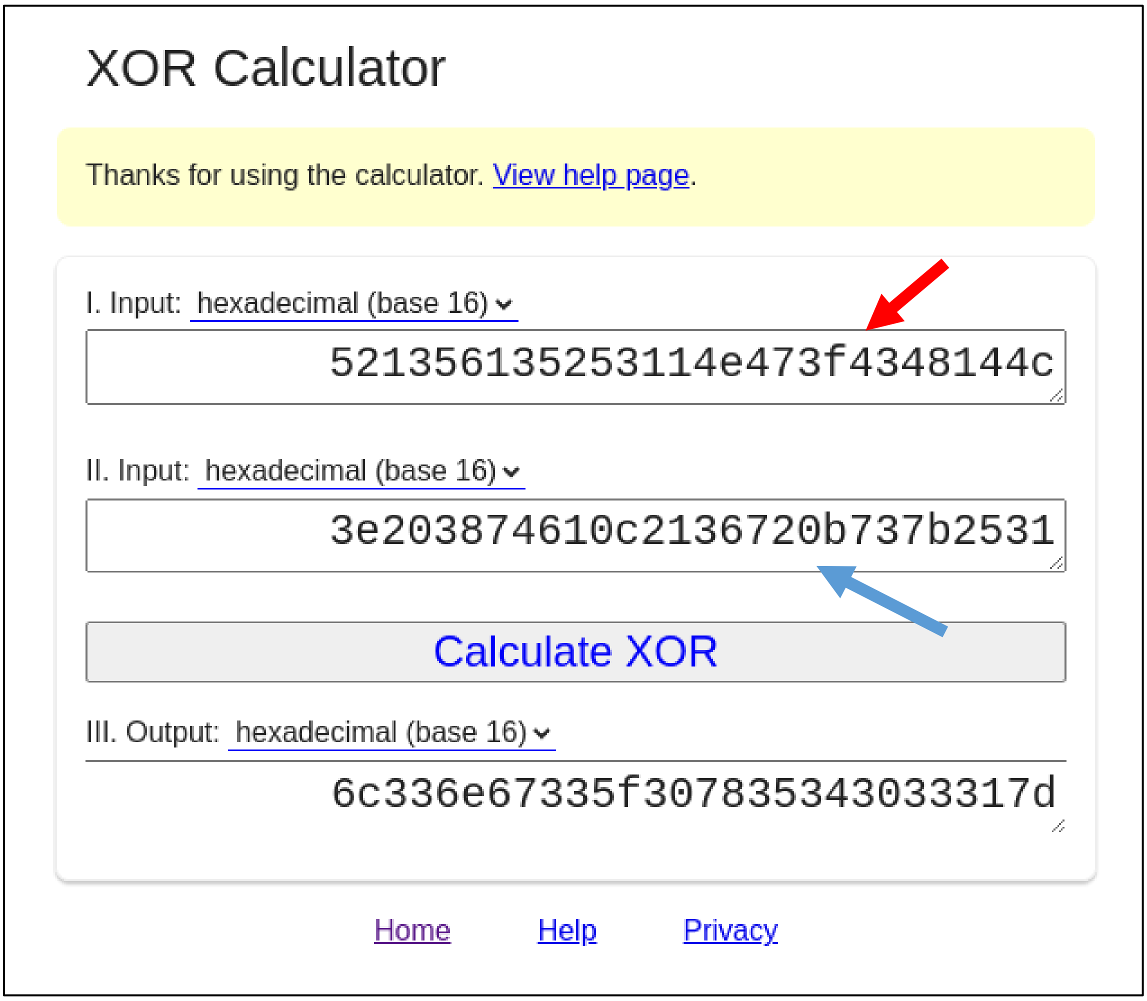
But, on line 48 we see that it is performing **XOR** of some calculation with “**local_68**” which is having values of stack “**DAT_00402060**” and it is checking whether it is equal to element of “**local_a8**” which is having the values of “**DAT_004020a0**”.

- Now we can manually take the hex values in of both stack “**DAT_00402060**” and “**DAT_004020a0**” and **XOR** it.



```
                        DAT_00402060                           XREF[1]:     main:00
00402060 52             ??         52h    R
00402061 00             ??         00h
00402062 00             ??         00h
00402063 00             ??         00h
00402064 13             ??         13h
00402065 00             ??         00h
00402066 00             ??         00h
00402067 00             ??         00h
00402068 56             ??         56h    V
00402069 00             ??         00h
0040206a 00             ??         00h
0040206b 00             ??         00h
0040206c 13             ??         13h
0040206d 00             ??         00h
0040206e 00             ??         00h
0040206f 00             ??         00h
00402070 52             ??         52h    R
00402071 00             ??         00h
00402072 00             ??         00h
00402073 00             ??         00h
00402074 53             ??         53h    S
00402075 00             ??         00h
00402076 00             ??         00h
00402077 00             ??         00h
00402078 11             ??         11h
00402079 00             ??         00h
```

- Calculating XOR



XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

```
521356135253114e473f4348144c
```

II. Input: hexadecimal (base 16)

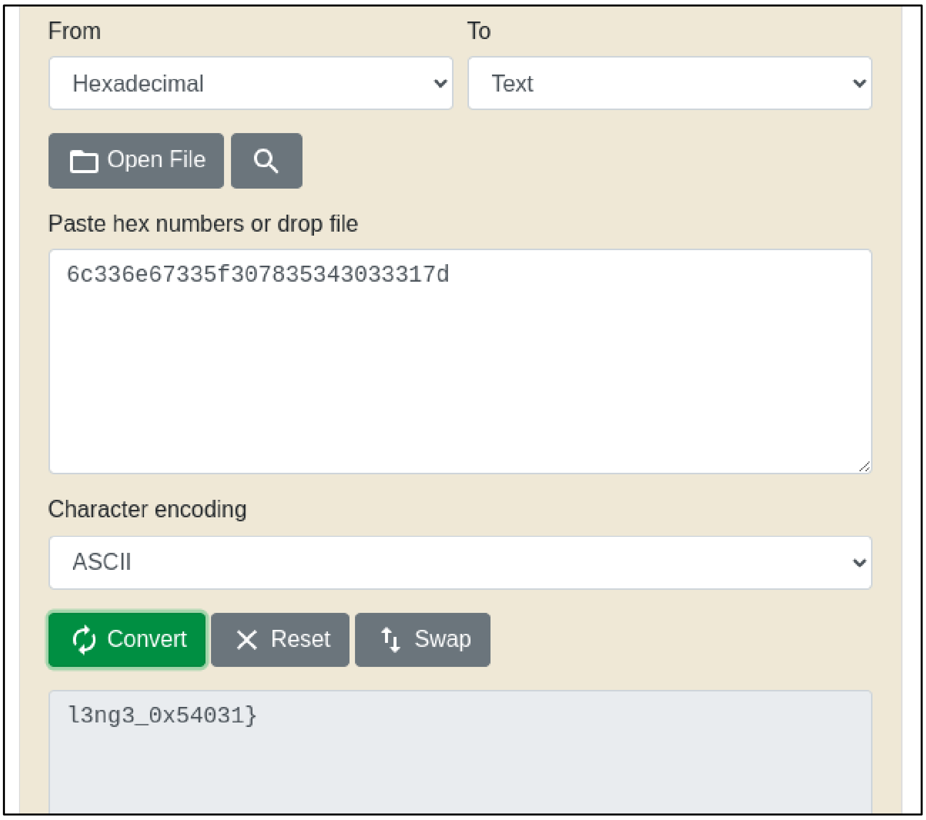
```
3e203874610c2136720b737b2531
```

III. Output: hexadecimal (base 16)

```
6c336e67335f307835343033317d
```

Red arrow hex values of "**DAT_00402060**"
Blue arrow hex values of "**DAT_004020a0**"

- Converting from Hex to ASCII



From: Hexadecimal

To: Text

Paste hex numbers or drop file

```
6c336e67335f307835343033317d
```

Character encoding: ASCII

```
l3ng3_0x54031}
```

Got the remaining part of flag.

- Now the flag is “**OWASP{cl4ss1c_r3v3rs1ng?ch4ll3ng3_0x54031}**”

I tried putting flag it gave me error. Then I change “**?**” to “**_**” and tried it worked.

Flag: “**OWASP{cl4ss1c_r3v3rs1ng_ch4ll3ng3_0x54031}**”