FOOLME

(REV)

Description : Make a call :) . But its not a pwn challenge.

Author : 0xrakesh

Hint : Don't try automation like **angr** .
And the flag will be meaningful text.
Did you play my **EeZY** reverse challenge?

File : foolme

Basic Analysis:

file command :

file foolme

```
→ testing  file foolme
foolme: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=09c9defd3791ce
d3729c5635a817146f3bec06b5, for GNU/Linux 3.2.0, stripped
```

It is 64 bit elf binary, dynamically linked and stripped.

Strings command :

strings foolme

```
                        [+]  Interesting Strings  [+]

Hey ,Don't try to callme

Wrong Length.
[+] HINT : Find the Length First
TamilCTF{StRiNgs_C0mP4re5}
                                Correct Passcode
                The Flag is TamilCTF{%s}
Wrong Pass!!!!
Enter the Passcode to get flag :
```

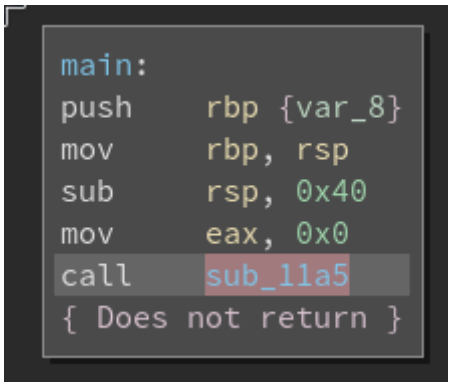
There is one interesting string (`TamilCTF{StRiNgs_C0mP4re5}`)

Execute the binary :

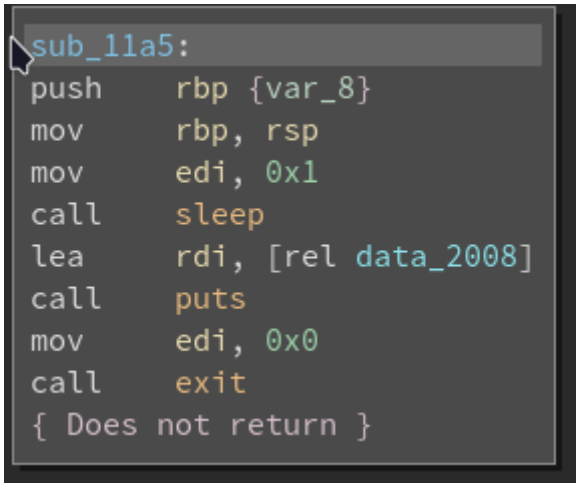
```
./foolme
```

```
→  testing  ./foolme
Hey ,Don't try to callme
→  testing
```

Open the binary in binaryninja :



The main function call the sub_11a5 function and it does not return .



The sub_11a5 function sleep for 1 second and exit the program. So we need to bypass the sub_11a5 function by convert the call instruction to nops .

Step : Right click the call instruction -> Patch option -> convert to Nop.

```
main:
push    rbp {__saved_rbp}
mov     rbp, rsp {__saved_rbp}
sub     rsp, 0x40
mov     eax, 0x0
nop
nop
nop
nop
nop
mov     eax, 0x0
call    sub_11c9
lea     rdi, [rel data_2480]  {"Enter the Passcode to get flag :…"}
mov     eax, 0x0
call    printf
mov     rdx, qword [rel stdin]
lea     rax, [rbp-0x40 {var_48}]
mov     esi, 0x1b
mov     rdi, rax {var_48}
call    fgets
lea     rax, [rbp-0x40 {var_48}]
mov     rdi, rax {var_48}
```

After the convert the call instruction to nops , just save the binay.

Execute the patch binary :

./patch

It ask for passcode to get the flag. Then print Wrong Length and find the length first.

Open the patch binary in Ghidra :

```
undefined8 FUN_001017e6(void)

{
  char local_48 [48];
  undefined8 local_18;
  undefined8 local_10;

  FUN_001011c9();
  printf("Enter the Passcode to get flag : ");
  fgets(local_48,0x1b,stdin);
  local_10 = FUN_001011e1(local_48);
  local_18 = FUN_001014ce(local_10);
  FUN_0010152b(local_18,local_48);
  return 0;
}
```

Main Function

```
Decompile: FUN_001011e1 - (patch)
16   iVar1 = ((int)param_1[0x10] << 3) >> 0x1f;
17   _DAT_001040a0 = (int)*param_1 << 2;
18   _DAT_001040a4 = (int)((char)(param_1[2] ^ 0xbU) / '\x02');
19   _DAT_001040a8 = (int)(char)(param_1[1] ^ 0x1f);
20   _DAT_001040ac = (char)(param_1[3] ^ 0x15) + 0xb;
21   _DAT_001040b0 = param_1[5] + 0xf;
22   _DAT_001040b4 = (int)param_1[6] + 0x45U ^ 2;
23   _DAT_001040b8 = (param_1[4] * 2) / 3;
24   _DAT_001040bc = (int)(char)(param_1[7] ^ 0x29);
25   _DAT_001040c0 = (0xc4 - param_1[9]) / 2;
26   _DAT_001040c4 = (param_1[8] + 7) / 2;
27   _DAT_001040c8 = param_1[0xb] + 5;
28   _DAT_001040cc = param_1[0xc] + -0x16;
29   _DAT_001040d0 = (int)(param_1[10] >> 4);
30   _DAT_001040d4 = (int)(char)(param_1[0x15] >> 2 ^ 0x2d);
31   _DAT_001040d8 = (int)param_1[0xd];
32   _DAT_001040dc = (int)(char)(param_1[0xf] ^ 0x23);
33   _DAT_001040e0 = param_1[0xe] + -0x3d;
34   _DAT_001040e4 = (((int)param_1[0x10] << 3) / 6 + iVar1 >> 1) - iVar1;
35   _DAT_001040e8 = param_1[0x12] + 0x1d;
36   _DAT_001040ec = (int)(param_1[0x11] >> 1);
37   _DAT_001040f0 = (int)(char)(param_1[0x14] ^ 0x46);
38   _DAT_001040f4 = (param_1[0x13] * 4) / 2;
39   _DAT_001040f8 = param_1[0x18] * 2;
40   _DAT_001040fc = (int)(char)(param_1[0x16] ^ 0xd);
41   _DAT_00104100 = (int)param_1[0x17] - 0x1dU ^ 1;
42   _DAT_00104104 = (int)(char)(param_1[0x19] ^ 0x42);
43   return &DAT_001040a0;
```

There are three interesting functions. The first function (FUN_001011e1) do some operation with user input and return the value.

```
undefined * FUN_001014ce(long param_1)

{
  int local_10;
  int local_c;

  local_c = 0x19;
  for (local_10 = 0; local_10 < 0x1a; local_10 = local_10 + 1) {
    *(undefined4 *)(&DAT_00104120 + (long)local_10 * 4) =
         *(undefined4 *)(param_1 + (long)local_c * 4);
    local_c = local_c + -1;
  }
  return &DAT_00104120;
}
```

The second function (FUN_001014ce) reverse the give paramter and return the value.

```
  local_40 = param_1[0x13];
  local_3c = param_1[0x12] - 9;
  local_38 = ((int)param_1[0x14] >> 1) + 0x14;
  local_34 = (int)param_1[0x16] / 2 + -2;
  local_30 = (int)param_1[0x15] / 2 ^ 0x52;
  local_2c = param_1[0x17] * 2 + 0x11;
  local_28 = param_1[0x18] ^ 0x1f;
  local_24 = param_1[0x19] ^ 0xb9;
  for (local_c = 0; local_c < 0x1a; local_c = local_c + 1) {
    local_a9 = (char)local_88[local_c];
    strncat(local_a8,&local_a9,1);
  }
  iVar1 = strcmp(local_a8,"TamilCTF{StRiNgs_C0mP4re5}");
  if (iVar1 == 0) {
    puts(&DAT_00102434);
    printf("\t\tThe Flag is TamilCTF{%s}\n",param_2);
  }
  else {
    puts(&DAT_0010246a);
  }
  return;
}
```

The third function (FUN_0010152b) do some operation as same as first function and finally the value is compare with TamilCTF{StRiNgs_C0mP4re5} .If it's equal ,it print "Correct Flag" , otherwise it print "Wrong Flag".

GOAL :

1. Find the flag by reverse the operation with TamilCTF{StRiNgs_C0mP4re5} string.
2. Don't use angr, because it has many possible flag.

Make a script :

We have the compare string (TamilCTF{StRiNgs_C0mP4re5}) , just reverse the operation.

Change the string into integer and store list.

compare_string = "TamilCTF{StRiNgs_C0mP4re5}"

Reverse the operation done in third function and return the inverse value.

```
def first_encode(par):
        encode = [0]*26
        encode[0] = par[0] ^ 0x29
        encode[1] = par[1] - 0x13
        encode[2] = par[2] ^ 1
        encode[3] = (par[4] ^ 0x4a) << 2
        encode[4] = par[3] ^ 0xd7
        encode[5] = (par[6] - 9) ^ 0x6e
        encode[6] = par[7] ^ 0x7f
        encode[7] = (par[5] ^ 7) + 0x1e
        encode[8] = par[8] - 0x2f
        encode[9] = par[10] ^ 0x4d
        encode[10] = (par[9] - 0x20) << 1
        encode[11] = (par[11] ^ 0x34) // 2
        encode[12] = par[12] ^ 0x56
        encode[13] = (par[14] - 2) ^ 0x60
        encode[14] = (par[13] << 1) - 0x40
        encode[15] =  (par[15] - 3) ^ 0x14
        encode[16] = (par[16] ^ 0x2a) - 0x3f
        encode[17] = par[17] - 0xe
        encode[18] = par[19] + 9
        encode[19] = par[18]
        encode[20] = (par[20] - 0x14) << 1
        encode[21] = (par[22] ^ 0x52) * 2
        encode[22] = (par[21] + 2) * 2
        encode[23] = (par[23] - 0x11) //2
        encode[24] = par[24] ^ 0x1f
        encode[25] = par[25] ^ 0xb9
        return encode[::-1]
```

Reverse the First function operation and print the flag as string

```
def find_me(par):
        flag = [0] * 26
        flag[0] = par[0] >> 2
        flag[1] = par[2] ^ 0x1f
        flag[2] = (par[1] * 2) ^ 0xb
        flag[3] = (par[3] - 0xb) ^ 0x15
        flag[4] = (par[6] * 3) //2
        flag[5] = par[4] - 0xf
        flag[6] = (par[5]^2) - 0x45
        flag[7] = par[7] ^ 0x29
        flag[8] = (par[9] * 2 ) - 7
        flag[9] = 196 - (par[8] * 2)
        flag[10] = par[12] << 4
        flag[11] = par[10] - 5
        flag[12] = par[11] + 0x16
        flag[13] = par[14]
        flag[14] = par[16] + 0x3d
        flag[15] = par[15] ^ 0x23
        flag[16] = (par[17] * 12 ) >> 0x3
        flag[17] = par[19] << 0x1
        flag[18] = par[18] - 0x1d
```

```
flag[19] = (par[21] * 2 ) // 4
flag[20] = par[20] ^ 0x46
flag[21] = (par[13] ^ 0x2d) << 2
flag[22] = par[23] ^ 0xd
flag[23] = (par[24] ^ 1) + 0x1d
flag[24] = par[22] // 2
flag[25] = par[25] ^ 0x42
print ''.join([ chr(i) for i in flag])
```

Output :

15_tH15_eZP_r3vErrE_cHalL?

1.As we already know that the flag is meaningful text.
2.And Hint is “Did you play my **EeZY** reverse challenge?”.

r3vErrE ---> r3vErsE
eZP ----> eZY

Passcode :

15_tH15_eZY_r3vErsE_cHalL?

Execute the patch binary with this passcode:

** Confirm the flag is right or not by submit the flag in website. **