SIGFLAG

DEC 30, 2017 • ARMIN WEIHBOLD • @KOYAAN5

Write-Up: m0rph from 34C3 CTF

This Write-Up is about solving the `m0rph` challenge from 34C3CTF using IDA Pro and radare2. I chose this to start on, because it looked easy to me and was marked *easy*. So I planned on doing a detailed step-by-step guide.

Getting to know the target

Get and unpack m0rph.

```
$ wget https://34c3ctf.ccc.ac/uploads/m0rph-9d6440cf8e1e4c6825b2efa16b3f
$ tar -xzvf m0rph-9d6440cf8e1e4c6825b2efa16b3f993d.tar.gz
$ cp m0rph/morph .
$ sha256sum morph
426070d85dd517363f328690d39c5399b833b5f2d7057980915f9012967774bc  morph
```

First thing we notice is that the target is quite small.

```
$ ls -lhS morph
```

```
-rwxr-xr-x 1 koyaan koyaan 10K Dez 27 17:52 morph
$ file morph
morph: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamical
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=1c81eb4bc8b981ed39ef79801d6fef03d4d81056, stripped
```

So, `file` output suggest we got a stripped 64-bit ELF executable. Starting it with various inputs gives us nothing but a non-zero exit-code.

```
$ ./morph
koyaan@meld: ~/ccc/rev3/m0rph 1
$ ./morph `python -c 'print("A"*1024)'`
koyaan@meld: ~/ccc/rev3/m0rph 1
$
```

`strings` reveals one interesting string:

```
$ strings morph
/lib64/ld-linux-x86-64.so.2
libc.so.6
exit
srand
puts
[...]
What are you waiting for, go submit that flag!
[...]
.comment
```

It looks like `morph` might just check a flag we submit for validity! Next, we are going to load it up in `radare2` .

Getting layout

First we disable ASLR to make our life easier:

```
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Then we start `morph` in `radare2` and print the entrypoint with `ie` .

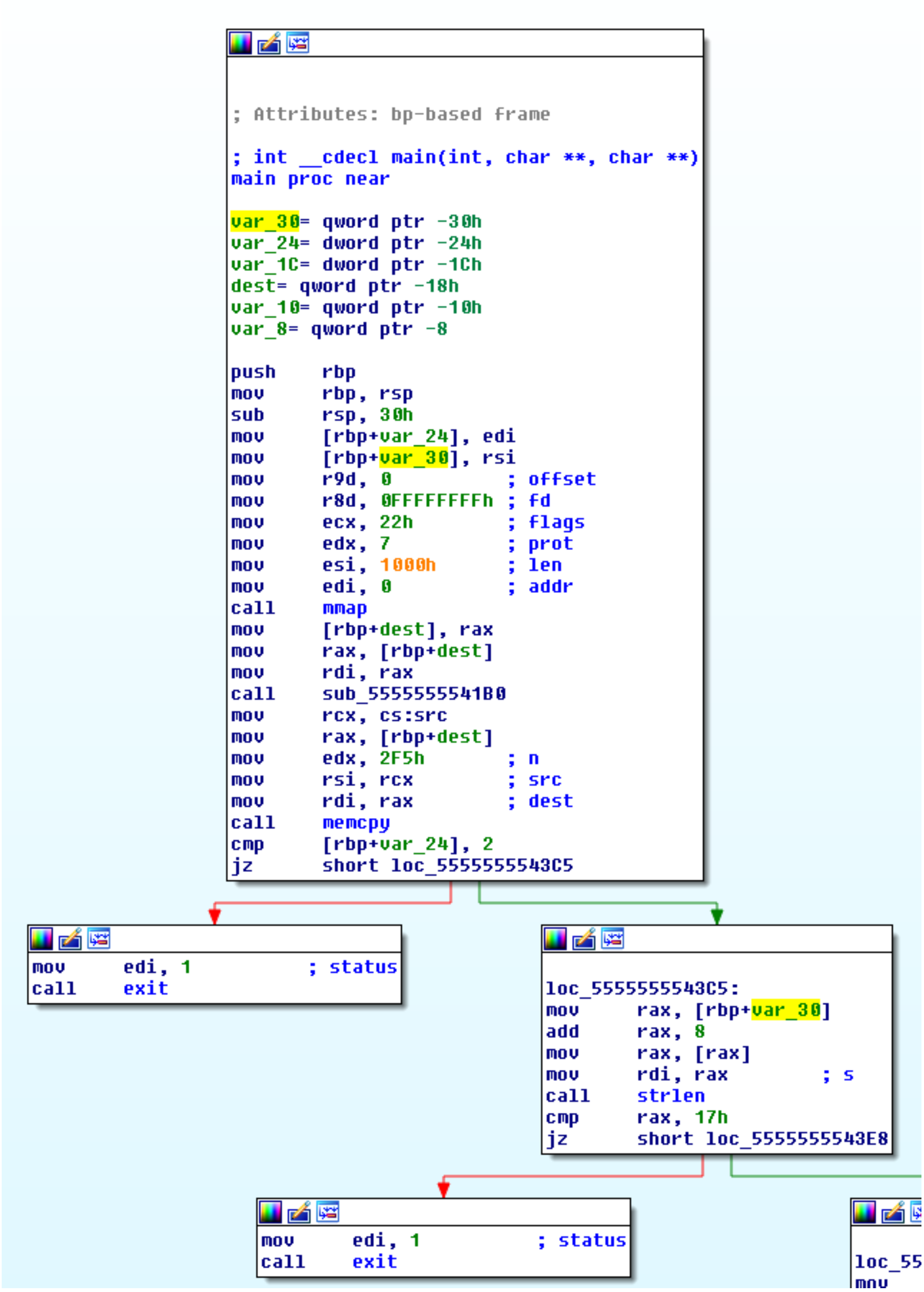
```
$ radare2 -d ./morph
Process with PID 21623 started...
= attach 21623 21623
bin.baddr 0x555555554000
Using 0x5455555554000
asm.bits 64
[0x7ffff7dd7c30]> ie
[Entrypoints]
vaddr=0x5555555547a0 paddr=0x000007a0 baddr=0x555555554000 laddr=0x00000
haddr=0x00000018 type=program

1 entrypoints

[0x7ffff7dd7c30]>
```

We see, that the base address of our executable is at `0x555555554000` and the entrypoint is at `0x5555555547a0` . Next step is to disassemble `morph` with IDA Pro.

Disassemble morph

First thing we do after letting the auto-analysis finish, is rebasing the program, such that all addresses correspond with the ones observed in the debugger. To do this we use *Edit* | *Segments* | *Rebase program…* and entered the base address we observed in `radare2` , `0x555555554000` .

After rebasing, we start examining the `main` function and can immediately see two checks that look a lot like checking `argc` and the length of `argv[1]` - and which get assigned at the very beginning (see Figure 1). So we can rename `var_24` to `argc` and `var_30` to `argv`. Now we can assume that `morph` takes one argument and it has to be 23 characters long.



```
; Attributes: bp-based frame

; int __cdecl main(int, char **, char **)
main proc near

var_30= qword ptr -30h
var_24= dword ptr -24h
var_1C= dword ptr -1Ch
dest= qword ptr -18h
var_10= qword ptr -10h
var_8= qword ptr -8

push    rbp
mov     rbp, rsp
sub     rsp, 30h
mov     [rbp+var_24], edi
mov     [rbp+var_30], rsi
mov     r9d, 0          ; offset
mov     r8d, 0FFFFFFFFh ; fd
mov     ecx, 22h        ; flags
mov     edx, 7          ; prot
mov     esi, 1000h      ; len
mov     edi, 0          ; addr
call    mmap
mov     [rbp+dest], rax
mov     rax, [rbp+dest]
mov     rdi, rax
call    sub_5555555541B0
mov     rcx, cs:src
mov     rax, [rbp+dest]
mov     edx, 2F5h       ; n
mov     rsi, rcx        ; src
mov     rdi, rax        ; dest
call    memcpy
cmp     [rbp+var_24], 2
jz      short loc_5555555543C5
```

```
mov     edi, 1          ; status
call    exit
```

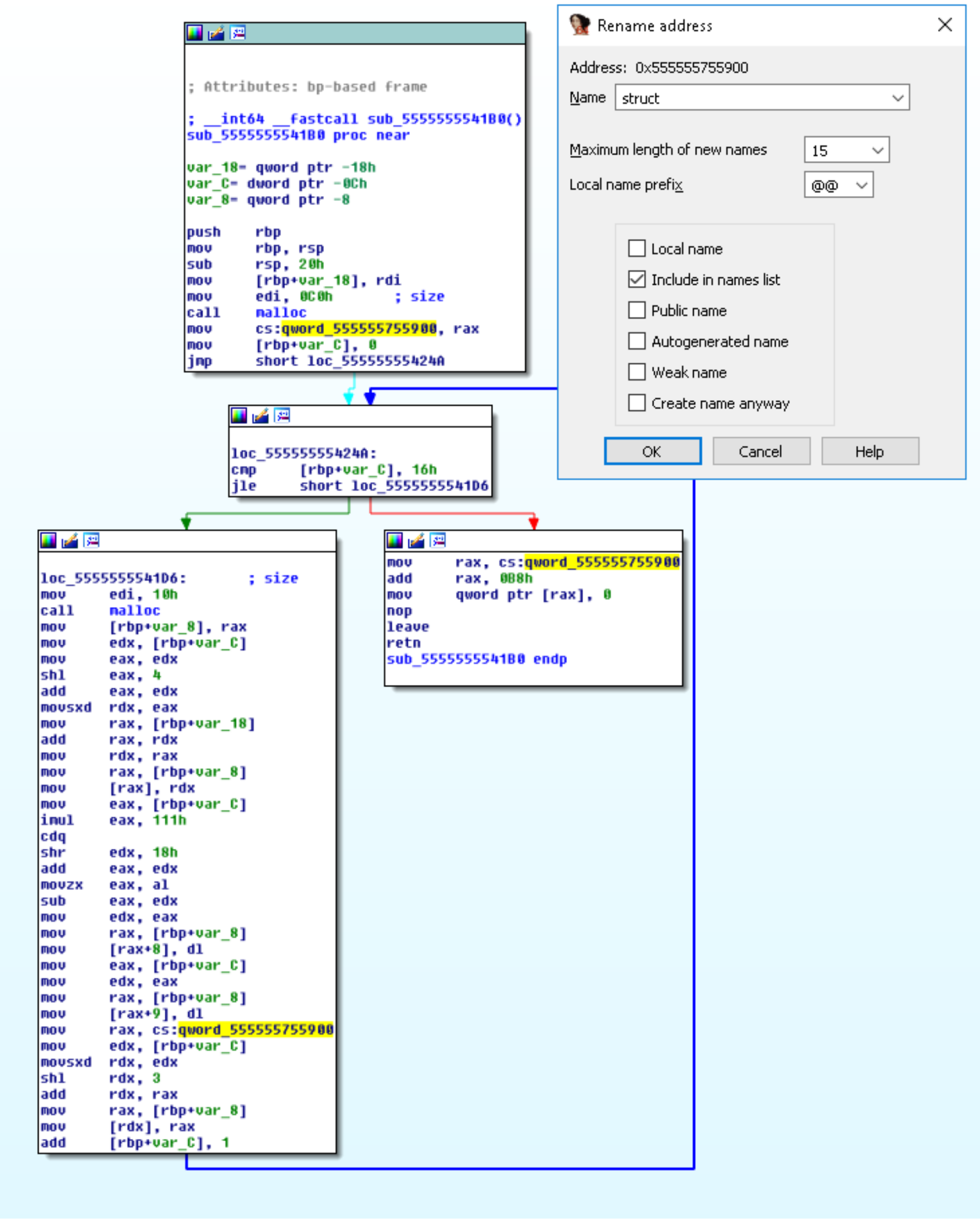
```
loc_5555555543C5:
mov     rax, [rbp+var_30]
add     rax, 8
mov     rax, [rax]
mov     rdi, rax        ; s
call    strlen
cmp     rax, 17h
jz      short loc_5555555543E8
```

```
mov     edi, 1          ; status
call    exit
```

```
loc_55
mov
```

Figure 1 - Use of `argv` coming from `rsi` getting stored into `var_30

The function `sub_5555555541B0` allocates and sets up some structure with 23 elements at the location `qword_555555755900` . We rename this location to `struct` , call this function `setupstruct` and also change its type to `__fastcall` (see Figure 2).



```
; Attributes: bp-based frame

; __int64 __fastcall sub_5555555541B0()
sub_5555555541B0 proc near

var_18= qword ptr -18h
var_C= dword ptr -0Ch
var_8= qword ptr -8

push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+var_18], rdi
mov     edi, 0C0h       ; size
call    malloc
mov     cs:qword_555555755900, rax
mov     [rbp+var_C], 0
jmp     short loc_55555555424A
```

```
loc_55555555424A:
cmp     [rbp+var_C], 16h
jle     short loc_5555555541D6
```

```
loc_5555555541D6:       ; size
mov     edi, 10h
call    malloc
mov     [rbp+var_8], rax
mov     edx, [rbp+var_C]
mov     eax, edx
shl     eax, 4
add     eax, edx
movsxd  rdx, eax
mov     rax, [rbp+var_18]
add     rax, rdx
mov     rdx, rax
mov     rax, [rbp+var_8]
mov     [rax], rdx
mov     eax, [rbp+var_C]
imul    eax, 111h
cdq
shr     edx, 18h
add     eax, edx
movzx   eax, al
sub     eax, edx
mov     edx, eax
mov     rax, [rbp+var_8]
mov     [rax+8], dl
mov     eax, [rbp+var_C]
mov     edx, eax
mov     rax, [rbp+var_8]
mov     [rax+9], dl
mov     rax, cs:qword_555555755900
mov     edx, [rbp+var_C]
movsxd  rdx, edx
shl     rdx, 3
add     rdx, rax
mov     rax, [rbp+var_8]
mov     [rdx], rax
add     [rbp+var_C], 1
```

```
mov     rax, cs:qword_555555755900
add     rax, 0B8h
mov     qword ptr [rax], 0
nop
leave
retn
sub_5555555541B0 endp
```

Rename address

Address: 0x555555755900

Name: struct

Maximum length of new names: 15

Local name prefix: @@

☐ Local name
☑ Include in names list
☐ Public name
☐ Autogenerated name
☐ Weak name
☐ Create name anyway

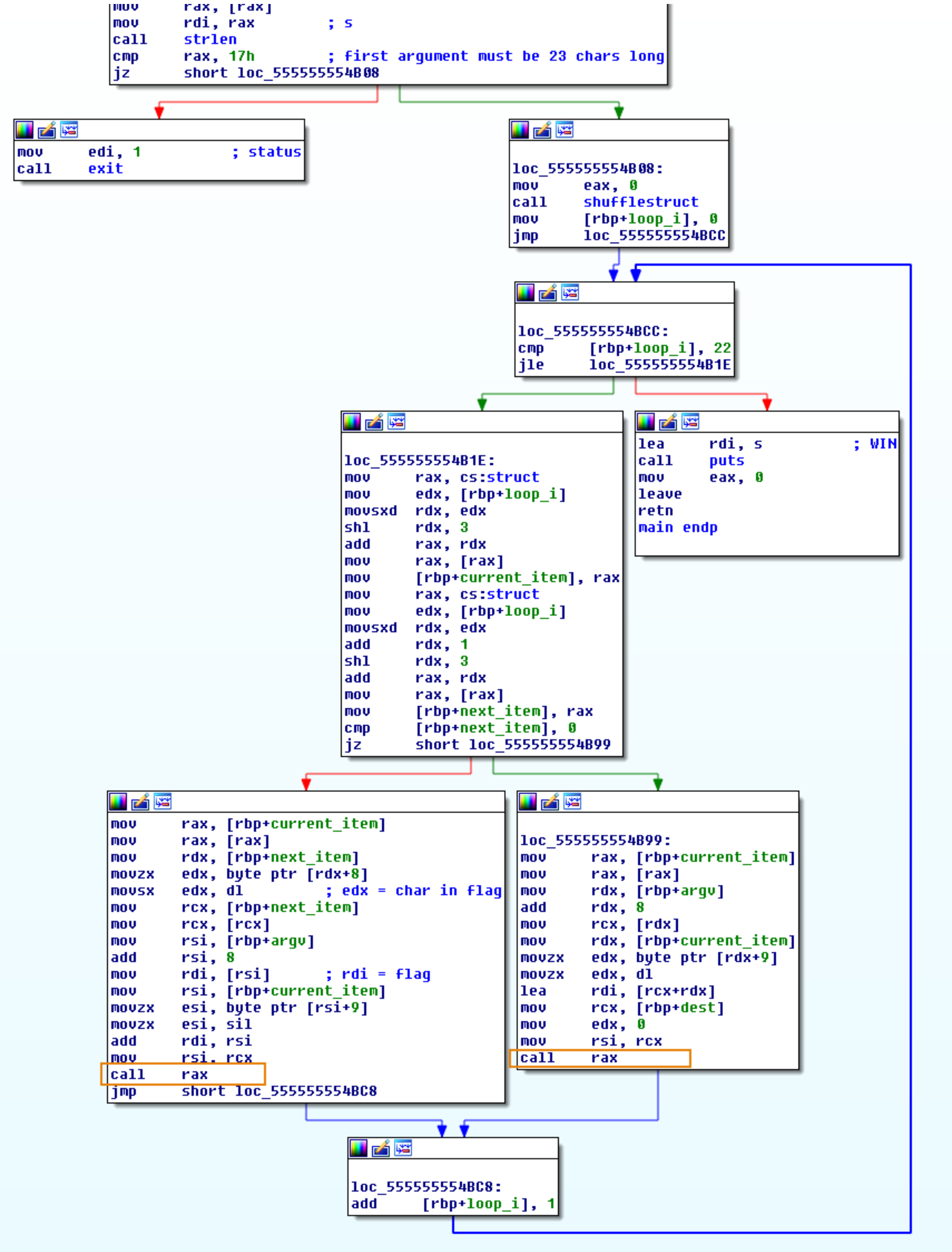
Figure 2 - Function `setupstruct`

We could now immediately start creating a proper structure in IDA Pro, but let's try to get an overview of the program first.

Examining `sub_555555554267` , we see that it just shuffles the elements of `struct` randomly.

```
void shufflestruct()
{
    unsigned int v0; // eax@1
    int rand1; // ST10_4@2
    int rand2_pre; // eax@2
    __int64 temp; // ST18_8@2
    signed __int64 rand2; // rcx@2
    signed int i; // [sp+Ch] [bp-14h]@1

    v0 = time(0LL);
    srand(v0);
    for ( i = 0; i <= 255; ++i )
    {
        rand1 = rand() % 22 + 1;
        rand2_pre = rand();
        temp = *(_QWORD *)(8LL * rand1 + struct);
        rand2 = 8LL * (rand2_pre % 22 + 1);
        *(_QWORD *)(struct + 8LL * rand1) = *(_QWORD *)(rand2 + struct);
        *(_QWORD *)(struct + rand2) = temp;
    }
}
```

In the the lower part of `main` , we can see a loop and rename the loop variable `loop_i` .



```
mov     rax, [rax]
mov     rdi, rax        ; s
call    strlen
cmp     rax, 17h        ; first argument must be 23 chars long
jz      short loc_555555554B08
```

```
mov     edi, 1          ; status
call    exit
```

```
loc_555555554B08:
mov     eax, 0
call    shufflestruct
mov     [rbp+loop_i], 0
jmp     loc_555555554BCC
```

```
loc_555555554BCC:
cmp     [rbp+loop_i], 22
jle     loc_555555554B1E
```

```
loc_555555554B1E:
mov     rax, cs:struct
mov     edx, [rbp+loop_i]
movsxd  rdx, edx
shl     rdx, 3
add     rax, rdx
mov     rax, [rax]
mov     [rbp+current_item], rax
mov     rax, cs:struct
mov     edx, [rbp+loop_i]
movsxd  rdx, edx
add     rdx, 1
shl     rdx, 3
add     rax, rdx
mov     rax, [rax]
mov     [rbp+next_item], rax
cmp     [rbp+next_item], 0
jz      short loc_555555554B99
```

```
lea     rdi, s          ; WIN
call    puts
mov     eax, 0
leave
retn
main endp
```

```
mov     rax, [rbp+current_item]
mov     rax, [rax]
mov     rdx, [rbp+next_item]
movzx   edx, byte ptr [rdx+8]
movsx   edx, dl         ; edx = char in flag
mov     rcx, [rbp+next_item]
mov     rcx, [rcx]
mov     rsi, [rbp+argv]
add     rsi, 8
mov     rdi, [rsi]      ; rdi = flag
mov     rsi, [rbp+current_item]
movzx   esi, byte ptr [rsi+9]
movzx   esi, sil
add     rdi, rsi
mov     rsi, rcx
call    rax
jmp     short loc_555555554BC8
```

```
loc_555555554B99:
mov     rax, [rbp+current_item]
mov     rax, [rax]
mov     rdx, [rbp+argv]
add     rdx, 8
mov     rcx, [rdx]
mov     rdx, [rbp+current_item]
movzx   edx, byte ptr [rdx+9]
movzx   edx, dl
lea     rdi, [rcx+rdx]
mov     rcx, [rbp+dest]
mov     edx, 0
mov     rsi, rcx
call    rax
```

```
loc_555555554BC8:
add     [rbp+loop_i], 1
```

Figure 3 - Checking loop in `main`

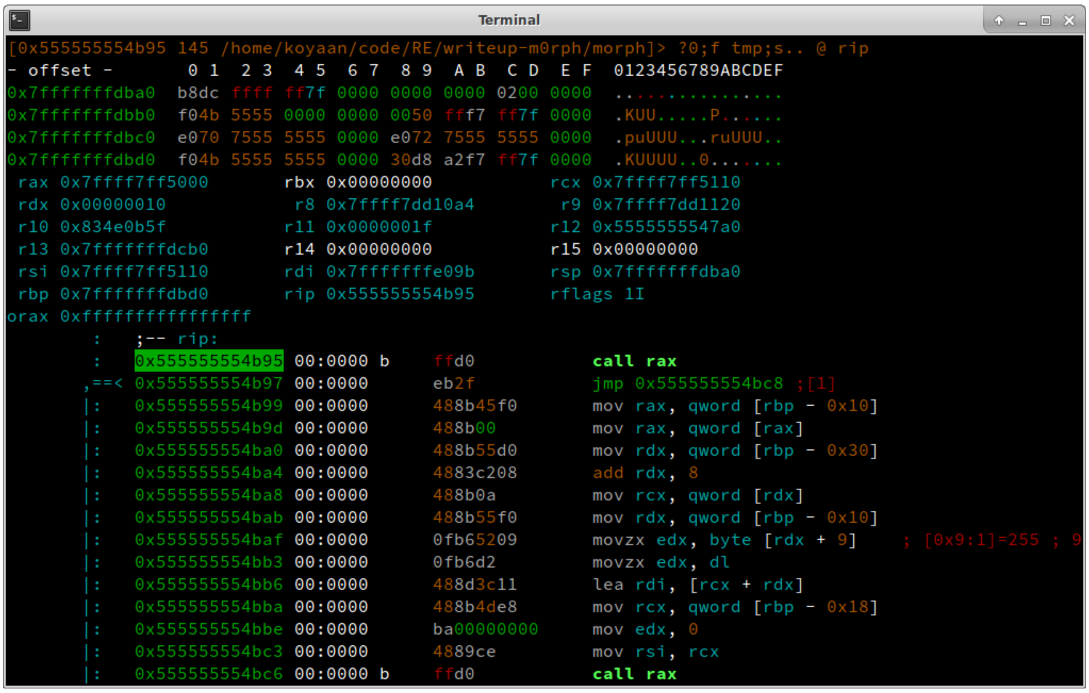
What we cannot follow here are the two `call eax` we see in Figure 3. Let's figure out what happens there with `radare2`. From the normal view in IDA Pro we can see they happen at `0x0000555555554B95` and `0x0000555555554BC6`.

Debugging with `radare2`

We start `morph` with an initial guess based on the known flag format, set breakpoints on the `call eax` instructions with `db` and continue the program with `dc`.

```
$ radare2 -d ./morph 34C3_AAAAAAAAAAAAAAAAAA
Process with PID 2544 started...
= attach 2544 2544
bin.baddr 0x555555554000
Using 0x555555554000
asm.bits 64
[0x7ffff7dd7c30]> db 0x0000555555554B95
[0x7ffff7dd7c30]> db 0x0000555555554BC6
[0x7ffff7dd7c30]> dc
hit breakpoint at: 555555554b95
[0x555555554b95]>
```

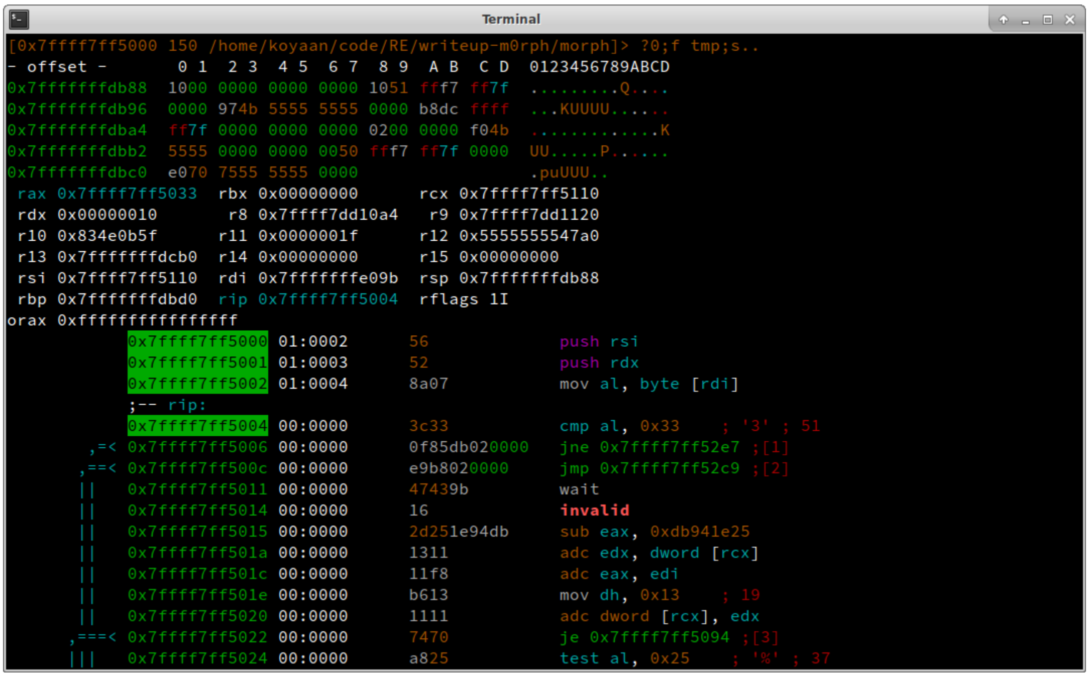
We then use `V` to go to visual mode and `p` two times to cycle to debugger view (`P` cycles backwards). This leaves us at the view of Figure 4.



```
[0x555555554b95 145 /home/koyaan/code/RE/writeup-m0rph/morph]> ?0;f tmp;s.. @ rip
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x7fffffffdba0  b8dc ffff ff7f 0000 0000 0000 0200 0000  ................
0x7fffffffdbb0  f04b 5555 0000 0000 0050 fff7 ff7f 0000  .KUU.....P......
0x7fffffffdbc0  e070 7555 5555 0000 e072 7555 5555 0000  .puUUU...ruUUU..
0x7fffffffdbd0  f04b 5555 5555 0000 30d8 a2f7 ff7f 0000  .KUUUU..0.......
 rax 0x7ffff7ff5000        rbx 0x00000000          rcx 0x7ffff7ff5110
 rdx 0x00000010             r8 0x7ffff7dd10a4       r9 0x7ffff7dd1120
 r10 0x834e0b5f            r11 0x0000001f          r12 0x5555555547a0
 r13 0x7fffffffdcb0        r14 0x00000000          r15 0x00000000
 rsi 0x7ffff7ff5110        rdi 0x7ffffffffe09b      rsp 0x7fffffffdba0
 rbp 0x7fffffffdbd0        rip 0x555555554b95      rflags 1I
orax 0xffffffffffffffff
        :   ;-- rip:
        :   0x555555554b95 00:0000 b     ffd0           call rax
       ,==< 0x555555554b97 00:0000       eb2f           jmp 0x555555554bc8 ;[1]
       |:   0x555555554b99 00:0000       488b45f0       mov rax, qword [rbp - 0x10]
       |:   0x555555554b9d 00:0000       488b00         mov rax, qword [rax]
       |:   0x555555554ba0 00:0000       488b55d0       mov rdx, qword [rbp - 0x30]
       |:   0x555555554ba4 00:0000       4883c208       add rdx, 8
       |:   0x555555554ba8 00:0000       488b0a         mov rcx, qword [rdx]
       |:   0x555555554bab 00:0000       488b55f0       mov rdx, qword [rbp - 0x10]
       |:   0x555555554baf 00:0000       0fb65209       movzx edx, byte [rdx + 9]     ; [0x9:1]=255 ; 9
       |:   0x555555554bb3 00:0000       0fb6d2         movzx edx, dl
       |:   0x555555554bb6 00:0000       488d3c11       lea rdi, [rcx + rdx]
       |:   0x555555554bba 00:0000       488b4de8       mov rcx, qword [rbp - 0x18]
       |:   0x555555554bbe 00:0000       ba00000000     mov edx, 0
       |:   0x555555554bc3 00:0000       4889ce         mov rsi, rcx
       |:   0x555555554bc6 00:0000 b     ffd0           call rax
```

Figure 4 - At the call

We now step into the call with `F7` and step more times until we reach the `cmp` instruction (see Figure 5).



```
[0x7ffff7ff5000 150 /home/koyaan/code/RE/writeup-m0rph/morph]> ?0;f tmp;s..
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  0123456789ABCD
0x7fffffffdb88  1000 0000 0000 0000 1051 fff7 ff7f  .........Q....
0x7fffffffdb96  0000 974b 5555 5555 0000 b8dc ffff  ...KUUUU......
0x7fffffffdba4  ff7f 0000 0000 0000 0200 0000 f04b  .............K
0x7fffffffdbb2  5555 0000 0000 0050 fff7 ff7f 0000  UU.....P......
0x7fffffffdbc0  e070 7555 5555 0000                 .puUUU..
 rax 0x7ffff7ff5033  rbx 0x00000000      rcx 0x7ffff7ff5110
 rdx 0x00000010       r8 0x7ffff7dd10a4   r9 0x7ffff7dd1120
 r10 0x834e0b5f      r11 0x0000001f      r12 0x5555555547a0
 r13 0x7fffffffdcb0  r14 0x00000000      r15 0x00000000
 rsi 0x7ffff7ff5110  rdi 0x7ffffffffe09b  rsp 0x7fffffffdb88
 rbp 0x7fffffffdbd0  rip 0x7ffff7ff5004  rflags 1I
orax 0xffffffffffffffff
           0x7ffff7ff5000 01:0002      56             push rsi
           0x7ffff7ff5001 01:0003      52             push rdx
           0x7ffff7ff5002 01:0004      8a07           mov al, byte [rdi]
           ;-- rip:
           0x7ffff7ff5004 00:0000      3c33           cmp al, 0x33     ; '3' ; 51
       ,=< 0x7ffff7ff5006 00:0000      0f85db020000   jne 0x7ffff7ff52e7 ;[1]
      ,==< 0x7ffff7ff500c 00:0000      e9b8020000     jmp 0x7ffff7ff52c9 ;[2]
      ||   0x7ffff7ff5011 00:0000      47439b         wait
      ||   0x7ffff7ff5014 00:0000      16             invalid
      ||   0x7ffff7ff5015 00:0000      2d251e94db     sub eax, 0xdb941e25
      ||   0x7ffff7ff501a 00:0000      1311           adc edx, dword [rcx]
      ||   0x7ffff7ff501c 00:0000      11f8           adc eax, edi
      ||   0x7ffff7ff501e 00:0000      b613           mov dh, 0x13     ; 19
      ||   0x7ffff7ff5020 00:0000      1111           adc dword [rcx], edx
     ,===< 0x7ffff7ff5022 00:0000      7470           je 0x7ffff7ff5094 ;[3]
     |||   0x7ffff7ff5024 00:0000      a825           test al, 0x25    ; '%' ; 37
```

Figure 5 - Passing check

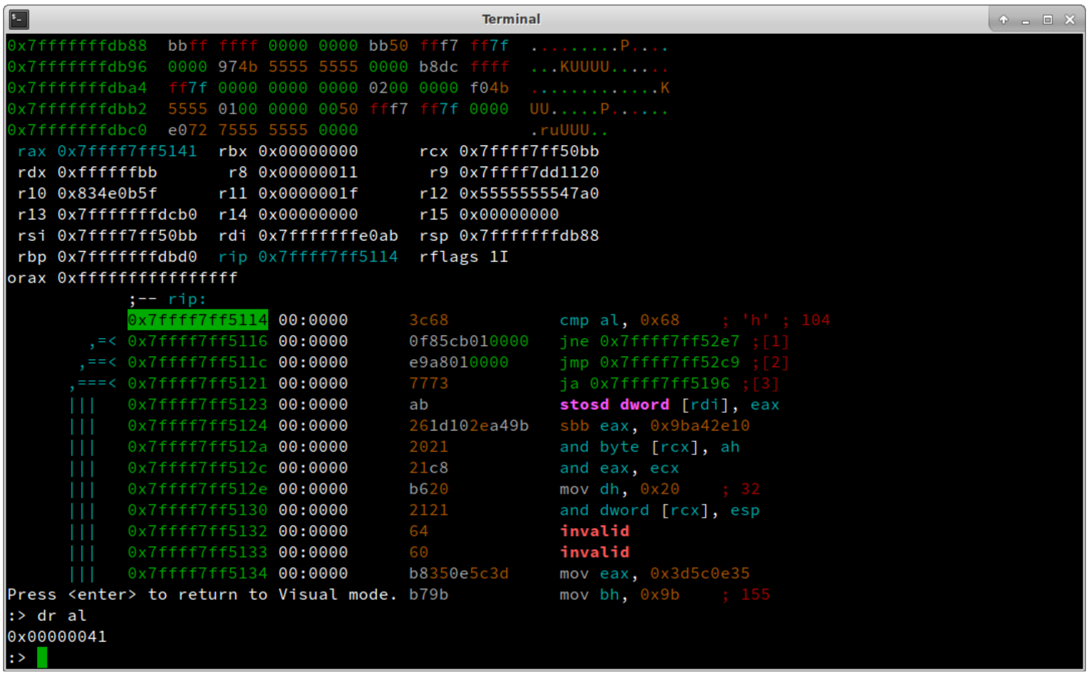
We hit `:` to get into command mode and enter `px 23 @ rdi` to dump 23 bytes in hex-format starting from the address stored in `rdi` . We can confirm this is our flag under scrutiny!

```
:> px 23 @ rdi
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  0123456789ABCD
0x7ffffffe09b  3334 4333 5f41 4141 4141 4141 4141  34C3_AAAAAAAAA
0x7ffffffe0a9  4141 4141 4141 4141 41              AAAAAAAAA
```

Since this check is fine, let's continue with `dc` , step 4 times (`4ds`), seek to `rip` (`s rip`) and hit enter then to go back to visual mode (see Figure 6). We see one of the 'A's is getting compared to 'h' (`cmp al, 0x68`). We see which one by printing `px 23 @ rdi` again. Subtracting this from the known start address gives us the numerical offset into the flag with the evaluate command `?` .

```
:> dc
child stopped with signal 28
[+] SIGNAL 28 errno=0 addr=0x00000000 code=128 ret=0
hit breakpoint at: 555555554b95
:> 4ds
:> s rip
:> dr al
0x00000041
:> px 23 @ rdi
- offset -       0 1  2 3  4 5  6 7  8 9  A B  C D  0123456789ABCD
0x7ffffffe0ab  4141 4141 4141 4100 5844 475f 5654  AAAAAAA.XDG_VT
0x7ffffffe0b9  4e52 3d37 004c 435f 50              NR=7.LC_P
:> ? rdi-0x7ffffffe09b
16 0x10 020 16 0000:0010 16 "\x10" 0b00010000 16.0 16.000000f 16.000000
:>
```

This shows that the 17th character of the flag should be an `h`



```
0x7fffffffdb88  bbff ffff 0000 0000 bb50 fff7 ff7f  ........P....
0x7fffffffdb96  0000 974b 5555 5555 0000 b8dc ffff  ...KUUUU.....
0x7fffffffdba4  ff7f 0000 0000 0000 0200 0000 f04b  ............K
0x7fffffffdbb2  5555 0100 0000 0050 fff7 ff7f 0000  UU.....P.....
0x7fffffffdbc0  e072 7555 5555 0000                .ruUUU..
 rax 0x7ffff7ff5141  rbx 0x00000000      rcx 0x7ffff7ff50bb
 rdx 0xffffffbb       r8 0x00000011       r9 0x7ffff7dd1120
 r10 0x834e0b5f      r11 0x0000001f      r12 0x5555555547a0
 r13 0x7fffffffdcb0  r14 0x00000000      r15 0x00000000
 rsi 0x7ffff7ff50bb  rdi 0x7ffffffe0ab  rsp 0x7fffffffdb88
 rbp 0x7fffffffdbd0  rip 0x7ffff7ff5114  rflags 1I
orax 0xffffffffffffffff
            ;-- rip:
            0x7ffff7ff5114 00:0000      3c68         cmp al, 0x68    ; 'h' ; 104
        ,=< 0x7ffff7ff5116 00:0000      0f85cb010000  jne 0x7ffff7ff52e7 ;[1]
       ,==< 0x7ffff7ff511c 00:0000      e9a8010000   jmp 0x7ffff7ff52c9 ;[2]
      ,===< 0x7ffff7ff5121 00:0000      7773         ja 0x7ffff7ff5196 ;[3]
      |||   0x7ffff7ff5123 00:0000      ab           stosd dword [rdi], eax
      |||   0x7ffff7ff5124 00:0000      261d102ea49b  sbb eax, 0x9ba42e10
      |||   0x7ffff7ff512a 00:0000      2021         and byte [rcx], ah
      |||   0x7ffff7ff512c 00:0000      21c8         and eax, ecx
      |||   0x7ffff7ff512e 00:0000      b620         mov dh, 0x20    ; 32
      |||   0x7ffff7ff5130 00:0000      2121         and dword [rcx], esp
      |||   0x7ffff7ff5132 00:0000      64           invalid
      |||   0x7ffff7ff5133 00:0000      60           invalid
      |||   0x7ffff7ff5134 00:0000      b8350e5c3d   mov eax, 0x3d5c0e35
Press <enter> to return to Visual mode. b79b         mov bh, 0x9b    ; 155
:> dr al
0x00000041
:>
```

Figure 6 - Failing check

Iterate

No we can just restart the process with our new guess `ood 34C3_AAAAAAAAAAAhAAAAAA` , skip over the first break and repeat these steps:

- Step 4 times `4ds`
- Print the `cmp` instruction that is next `pd 1 @ rip`
- Print the character being compared `dr al`
- Print the offset into the flag (`?  rdi-0x7ffffffe09b~[:2]` using the "mini grep" `~` with python-like indexing for the result to just get the integer value)

```
:> ood 34C3_AAAAAAAAAAAhAAAAAA
[0x7ffff7dd7c30]> dc
hit breakpoint at: 555555554b95
[0x555555554b95]> dc
hit breakpoint at: 555555554b95
[0x555555554b95]> 4ds
```

```
[0x7ffff7ff5176]> pd 1 @ rip
;-- rip:
0x7ffff7ff517a 00:0000      3c30           cmp al, 0x30    ; '0' ; 48
[0x7ffff7ff5176]> dr al
0x00000041
[0x7ffff7ff5176]> ? rdi-0x7fffffffe09b~[:2]
22
[0x7ffff7ff5176]>
```

We see that the 23rd character should be a `0`, so we restart with `ood 34C3_AAAAAAAAAAAhAAAAA0` and so on, reconstructing the flag character by character.

```
$ ./morph  34C3_M1GHTY_M0RPh1nG_g0
What are you waiting for, go submit that flag!
```

Thinkspace theme by Heiswayi Nrird | Last modified Saturday, 02 June 2018 | © 2018 SIGFLAG