Description

Furious at the theft of his holy rice, the monk has ascended to his final form: Vengeful Tech Guru. He rebuilt the locker, harder, better, and definitely more cursed. The pigeon mafia failed — now it's your turn.

- Author: hampter
- flag: apoorvctf{h0w_d1d_u_3v3n_f1nd_th1s:0}

Writeup

So we have been given a file called `evil-rice-cooker` executing asks for a password, giving something random says monk laughs today.

So i ran `file evil-rice-cooker`

```
evil-rice-cooker: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=b016e91d1d7ef3231be27f1cc93ed728010e162c, for GNU/Linux
4.4.0, stripped
```

Hence out binary is stripped, so we wont have symbol table to work with.

Now i opened the given binary in *BinaryNinja*(you can use any decompiler of you choice, like Ghidra, ida)

i check the main function and see this.

main

```
00001215    int32_t main(int32_t argc, char** argv, char** envp)

00001220        void* fsbase
00001220        int64_t rax = *(fsbase + 0x28)
0000123e        printf(format: "Enter password: ")
00001264        void var_118
00001264        int32_t result
00001264
00001264        if (__isoc99_scanf(format: "%255s", &var_118) == 1)
000012a6            if (strlen(&var_118) == 0x25)
000012e6                int64_t rax_6 = mmap(addr: nullptr, len: data_4130,
prot: 7, flags: 0x22, fd: 0xffffffff, offset: 0)
000012e6
000012fa                if (rax_6 == -1)
00001306                    perror(s: "mmap")
00001310                    exit(status: 1)
00001310                    noreturn
00001310
00001330                memcpy(rax_6, &data_4080, data_4130)
00001353                sub_11c9(rax_6, data_4130, data_4138)
0000135f                int64_t var_128_1 = rax_6
```

```
00001385                rax_6(&var_118)
00001387                result = 0
000012a6            else
000012b2                puts(str: "monk laughs today")
000012b7                result = 1
00001264        else
00001284            fwrite(buf: "Input error.\n", size: 1, count: 0xd, fp:
stderr)
00001289            result = 1
00001289
00001390        *(fsbase + 0x28)
00001390
00001399        if (rax == *(fsbase + 0x28))
000013a1            return result
000013a1
0000139b        __stack_chk_fail()
0000139b        noreturn
```

right away we can see it asks for password then check the lenght `if (strlen(&var_118) == 0x25)` hence out input length should be **0x25** or **37** in decimal

Hence i gave an input as 37 a's `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa` but

```
Enter password: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
monk laughs today
```

But the monk was still laughing, hence looking forward, i see `memcpy` and `sub_11c9` being called.

looking at `memcpy` it loads `&data_4080`, checking its values,

```
00004080  data_4080:
00004080  42 aa aa aa aa f1 e2 9b  B.......
00004088  63 e2 a5 1c ae a5 aa 62  c......b
00004090  5c 7a 9e ee ae bd 6a 62  \z....jb
00004098  a9 82 62 e2 27 f9 f9 90  ..b.'...
000040a0  ae a0 df be e2 55 6b e2  .....Uk.
000040a8  29 53 8f df 76 e2 27 d9  )S..v.'.
000040b0  d2 10 b3 aa aa aa 41 a6  ......A.
000040b8  e2 27 19 3b aa aa aa 10  .'.;....
000040c0  b9 aa aa aa 12 ab aa aa  ........
000040c8  aa 15 ab aa aa aa a5 af  ........
000040d0  69 3a 3a 3a 3a 3a 3a 3a  i:::::::
000040d8  94 91 90 b3 32 5f 9d 59  ....2_.Y
000040e0  3f 6b b8 4d 77 7a a4 99  ?k.Mwz..
000040e8  66 e1 f2 a3 e4 79 a1 39  f....y.9
000040f0  ce c8 6c a4 84 14 40 e1  ..l...@.
000040f8  aa 0d 17 76 a8 d3 c5 df  ...v....
00004100  8a ce cb 8a d8 c3 c9 cf  ........
00004108  8a cd c5 ce 8a c5 d8 8a  ........
```

```
00004110  dd c5 de 95 a0 aa c7 c5  ........
00004118  c4 c1 8a c6 cb df cd c2  ........
00004120  d9 8a de c5 ce cb d3 a0  ........
00004128  aa 00 00 00 00 00 00 00  ........
```

looking at `sub_11c9` it takes `data_4130` and `data_4138` as parameters

```
000011c9    void* sub_11c9(int64_t arg1, int64_t arg2, char arg3)

0000120f        void* i
0000120f
0000120f        for (i = nullptr; i u< arg2; i += 1)
00001200            *(i + arg1) ^= arg3
00001200
00001214        return i
```

this is just a simple XOR function with key arg3 which is `data_4138` looking at its value, `00004138 char data_4138 = -0x56`

This value `-0x56` is saved as two compliment form when converted to normal we get key as `0xAA`

So basically we load values from `data_4080` apply XOR with `0xAA` so, i wrote a simple python program to xor it,

```
data = [
    0x42, 0xaa, 0xaa, 0xaa, 0xaa, 0xf1, 0xe2, 0x9b,
    0x63, 0xe2, 0xa5, 0x1c, 0xae, 0xa5, 0xaa, 0x62,
    0x5c, 0x7a, 0x9e, 0xee, 0xae, 0xbd, 0x6a, 0x62,
    0xa9, 0x82, 0x62, 0xe2, 0x27, 0xf9, 0xf9, 0x90,
    0xae, 0xa0, 0xdf, 0xbe, 0xe2, 0x55, 0x6b, 0xe2,
    0x29, 0x53, 0x8f, 0xdf, 0x76, 0xe2, 0x27, 0xd9,
    0xd2, 0x10, 0xb3, 0xaa, 0xaa, 0xaa, 0x41, 0xa6,
    0xe2, 0x27, 0x19, 0x3b, 0xaa, 0xaa, 0xaa, 0x10,
    0xb9, 0xaa, 0xaa, 0xaa, 0x12, 0xab, 0xaa, 0xaa,
    0xaa, 0x15, 0xab, 0xaa, 0xaa, 0xaa, 0xa5, 0xaf,
    0x69, 0x3a, 0x3a, 0x3a, 0x3a, 0x3a, 0x3a, 0x3a,
    0x94, 0x91, 0x90, 0xb3, 0x32, 0x5f, 0x9d, 0x59,
    0x3f, 0x6b, 0xb8, 0x4d, 0x77, 0x7a, 0xa4, 0x99,
    0x66, 0xe1, 0xf2, 0xa3, 0xe4, 0x79, 0xa1, 0x39,
    0xce, 0xc8, 0x6c, 0xa4, 0x84, 0x14, 0x40, 0xe1,
    0xaa, 0x0d, 0x17, 0x76, 0xa8, 0xd3, 0xc5, 0xdf,
    0x8a, 0xce, 0xcb, 0x8a, 0xd8, 0xc3, 0xc9, 0xcf,
    0x8a, 0xcd, 0xc5, 0xce, 0x8a, 0xc5, 0xd8, 0x8a,
    0xdd, 0xc5, 0xde, 0x95, 0xa0, 0xaa, 0xc7, 0xc5,
    0xc4, 0xc1, 0x8a, 0xc6, 0xcb, 0xdf, 0xcd, 0xc2,
    0xd9, 0x8a, 0xde, 0xc5, 0xce, 0xcb, 0xd3, 0xa0,
    0xaa, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
]
```

```
key = 0xAA
xored = bytes(b ^ key for b in data)

with open("data.bin", "wb") as f:
    f.write(xored)
```

Since this is loaded into memory we should be able to see it as assembly instructions.

running `ndisasm -b 64 data.bin` we get

```
00000000  E800000000        call 0x5
00000005  5B                pop rbx
00000006  4831C9            xor rcx,rcx
00000009  480FB6040F        movzx rax,byte [rdi+rcx]
0000000E  00C8              add al,cl
00000010  F6D0              not al
00000012  3444              xor al,0x44
00000014  0417              add al,0x17
00000016  C0C803            ror al,byte 0x3
00000019  28C8              sub al,cl
0000001B  488D5353          lea rdx,[rbx+0x53]
0000001F  3A040A            cmp al,[rdx+rcx]
00000022  7514              jnz 0x38
00000024  48FFC1            inc rcx
00000027  4883F925          cmp rcx,byte +0x25
0000002B  75DC              jnz 0x9
0000002D  488D7378          lea rsi,[rbx+0x78]
00000031  BA19000000        mov edx,0x19
00000036  EB0C              jmp short 0x44
00000038  488DB391000000    lea rsi,[rbx+0x91]
0000003F  BA13000000        mov edx,0x13
00000044  B801000000        mov eax,0x1
00000049  BF01000000        mov edi,0x1
0000004E  0F05              syscall
00000050  C3                ret
00000051  90                nop
00000052  90                nop
00000053  90                nop
00000054  90                nop
00000055  90                nop
00000056  90                nop
00000057  90                nop
00000058  3E3B3A            cmp edi,[ds:rdx]
0000005B  1998F537F395      sbb [rax-0x6a0cc80b],ebx
00000061  C112E7            rcl dword [rdx],byte 0xe7
00000064  DDD0              fst st0
00000066  0E                db 0x0e
00000067  33CC              xor ecx,esp
00000069  4B58              pop r8
0000006B  094ED3            or [rsi-0x2d],ecx
0000006E  0B936462C60E      or edx,[rbx+0xec66264]
```

```
00000074  2EBEEA4B00A7      cs mov esi,0xa7004bea
0000007A  BDDC02796F        mov ebp,0x6f7902dc
0000007F  7520              jnz 0xa1
00000081  64                fs
00000082  61                db 0x61
00000083  207269            and [rdx+0x69],dh
00000086  63                db 0x63
00000087  6520676F          and [gs:rdi+0x6f],ah
0000008B  64206F72          and [fs:rdi+0x72],ch
0000008F  20776F            and [rdi+0x6f],dh
00000092  743F              jz 0xd3
00000094  0A00              or al,[rax]
00000096  6D                insd
00000097  6F                outsd
00000098  6E                outsb
00000099  6B206C            imul esp,[rax],byte +0x6c
0000009C  61                db 0x61
0000009D  7567              jnz 0x106
0000009F  687320746F        push qword 0x6f742073
000000A4  64                fs
000000A5  61                db 0x61
000000A6  790A              jns 0xb2
000000A8  00AAAAAAAAAA      add [rdx-0x55555556],ch
000000AE  AA                stosb
000000AF  AA                stosb
```

Looking at this, instructions look fine till `00000050` after which some weird stuff is happening, probably because they are not instruciton.

Looking at instructions one by one.

Initially we call 0x5 which makes program jumps to 0x5 then pop rbx making `rbx = 0x5` then performing `xor rcx rcx` which makes `rcx = 0`.

next we have

```
00000009  480FB6040F        movzx rax,byte [rdi+rcx]
0000000E  00C8              add al,cl
00000010  F6D0              not al
00000012  3444              xor al,0x44
00000014  0417              add al,0x17
00000016  C0C803            ror al,byte 0x3
00000019  28C8              sub al,cl
```

here we move the byte at [rdi+rcx] = [rdi] since rcx is 0, hence `rcx = rdi`, now it performs some transformation, which we will skip for now and come back later,

next we have,

```
0000001B  488D5353          lea rdx,[rbx+0x53]
0000001F  3A040A            cmp al,[rdx+rcx]
00000022  7514              jnz 0x38
```

now we move value [rbx + 0x53] = [0x5 + 0x53] = [0x58] to rdx, next we compare byte at [rdx + rcx] = [0x58 + 0] = [0x58] to our byte at al, if they are not equal it jumps to 0x38 which is probably the fail condition which prints `monk laughs today` which we want to avoid.

next we have,

```
00000024  48FFC1            inc rcx
00000027  4883F925          cmp rcx,byte +0x25
0000002B  75DC              jnz 0x9
```

we increment the value of rcx making `rcx = 0x1` and then cmp with `+0x25 = 37` (remember our input value is 37) meaning we loop through all char in our input apply these transformations

```
0000000E  00C8              add al,cl
00000010  F6D0              not al
00000012  3444              xor al,0x44
00000014  0417              add al,0x17
00000016  C0C803            ror al,byte 0x3
00000019  28C8              sub al,cl
```

then cmp with value at