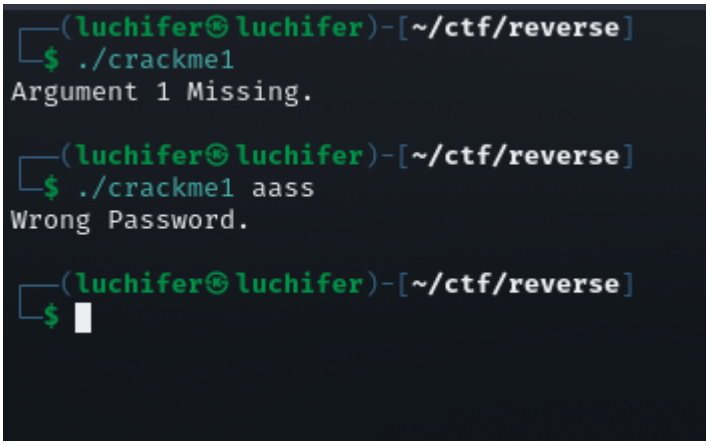
Reverse Engineering (crackme1)

Step by Step Procedure to Crack Software

Step 1:



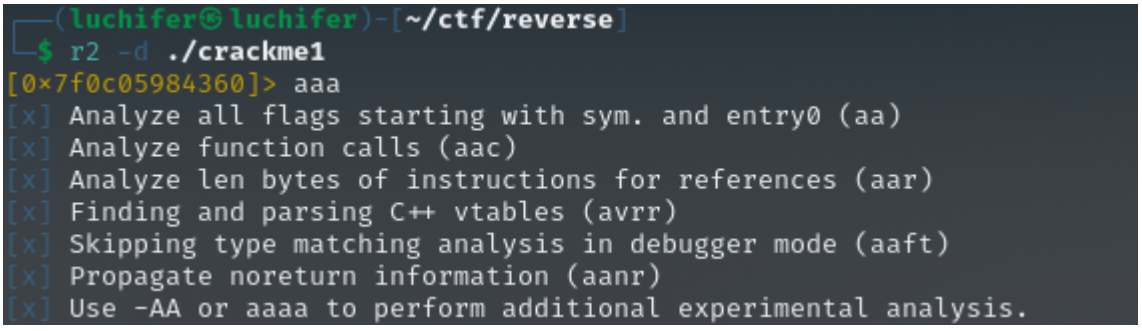
So, the software takes one argument. If we give the argument (initially, we don't know the correct argument, so we give a random one).
I will be using radare2. You can pick anything you want.

Step 2:

To use radare2 in Linux we use:

```
r2 -d ./filename
```

The we use -> aaa to analyze function name.



To know about function in software we use: ***afl***

afl means -> **Analyze Function List**

*Then we use : **db main***

DB means -> **Debugging Mode**

DC means->**Debugging continue**

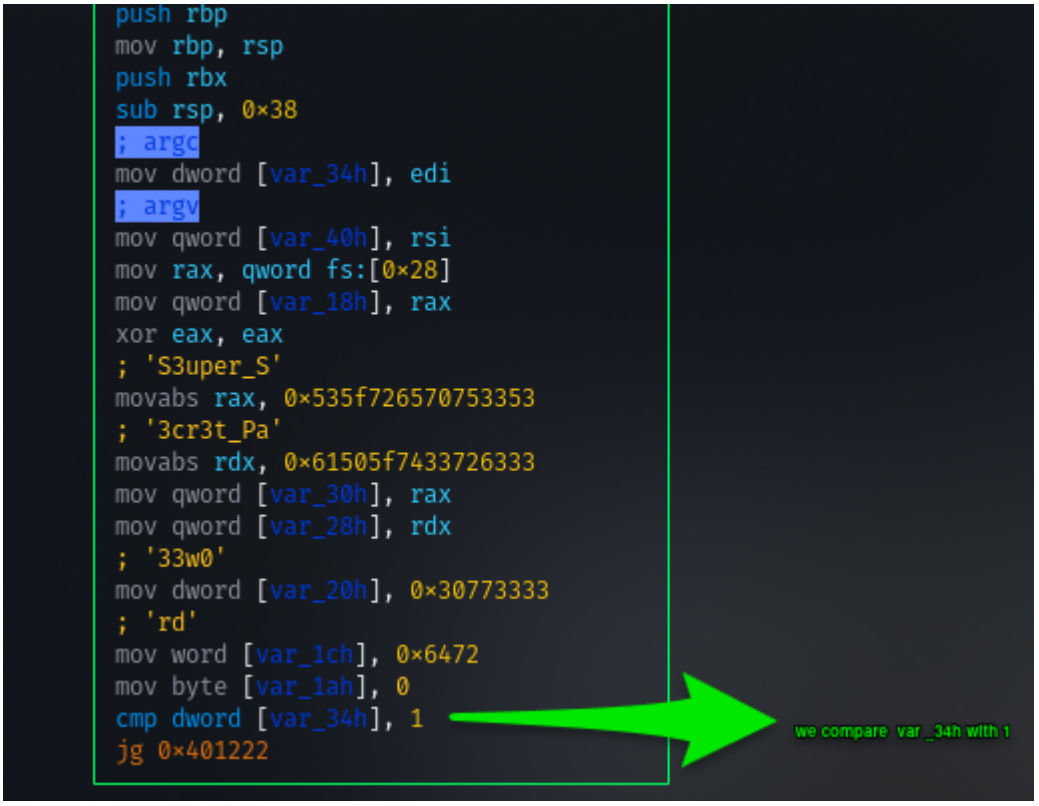
```
┌──(luchifer㉿luchifer)-[~/ctf/reverse]
└─$ r2 -d ./crackme1
[0×7f6b378c2360]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0×7f6b378c2360]> afl
0×004010d0    1 47           entry0
0×00401110    4 33   → 31    sym.deregister_tm_clones
0×00401140    4 49           sym.register_tm_clones
0×00401180    3 33   → 32    sym.__do_global_dtors_aux
0×004011b0    1 6            entry.init0
0×00401330    1 5            sym.__libc_csu_fini
0×00401338    1 13           sym._fini
0×004012c0    4 101          sym.__libc_csu_init
0×00401100    1 5            sym._dl_relocate_static_pie
0×004011b6   10 257          main
0×00401000    3 27           sym._init
0×00401080    1 11           sym.imp.puts
0×00401090    1 11           sym.imp.strlen
0×004010a0    1 11           sym.imp.__stack_chk_fail
0×004010b0    1 11           sym.imp.strcmp
0×004010c0    1 11           sym.imp.exit
[0×7f6b378c2360]> db main
[0×7f6b378c2360]> dc
hit breakpoint at: 0×4011b6
[0×004011b6]> VV
```

we will be using visual graph mode so we use VV for that.

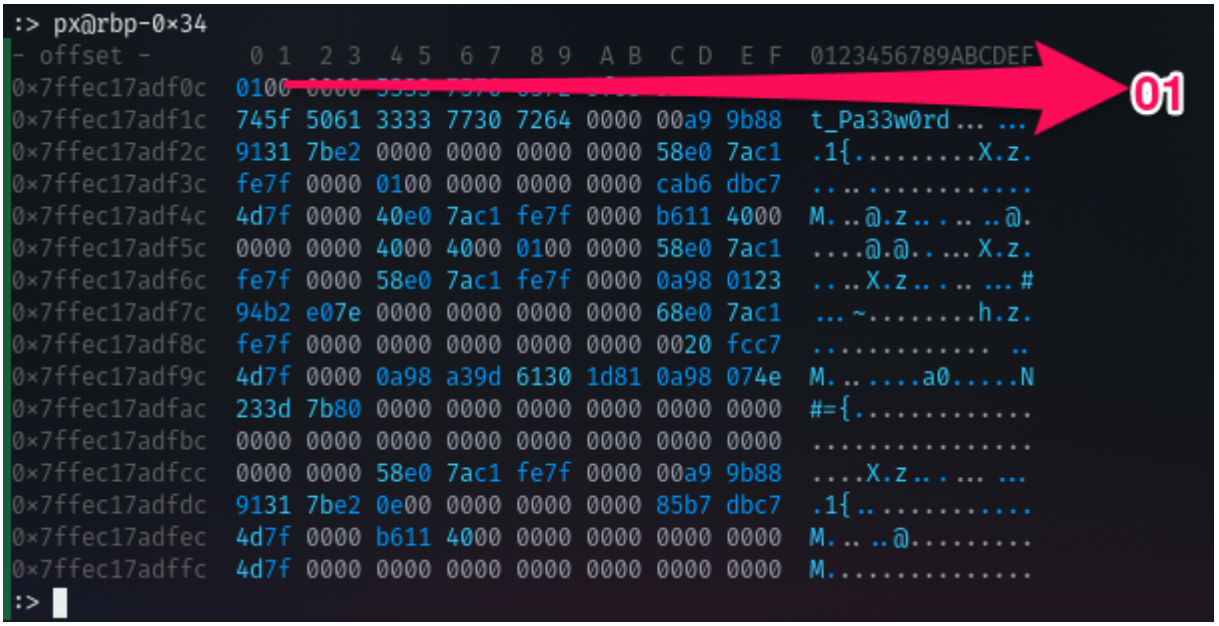
Step 3:

to use command in visual mode type [shift + ;]

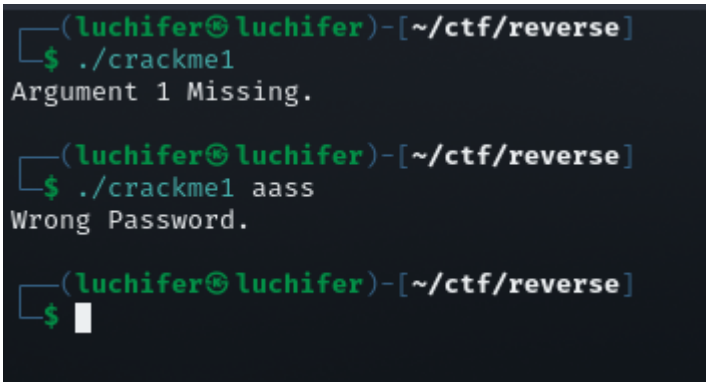
```
 0×4011b6 [oa]
  ; DATA XREF from entry0 @ 0×4010f1
257: int main (int argc, char **argv);
; var int64_t var_40h @ rbp-0×40
; var int64_t var_34h @ rbp-0×34
; var int64_t var_30h @ rbp-0×30
; var int64_t var_28h @ rbp-0×28
; var int64_t var_20h @ rbp-0×20
; var int64_t var_1ch @ rbp-0×1c
; var int64_t var_1ah @ rbp-0×1a
; var int64_t var_18h @ rbp-0×18
; arg int argc @ rdi
; arg char **argv @ rsi
endbr64
```



to know whats the value of '@rbp-0x34' we can use px command

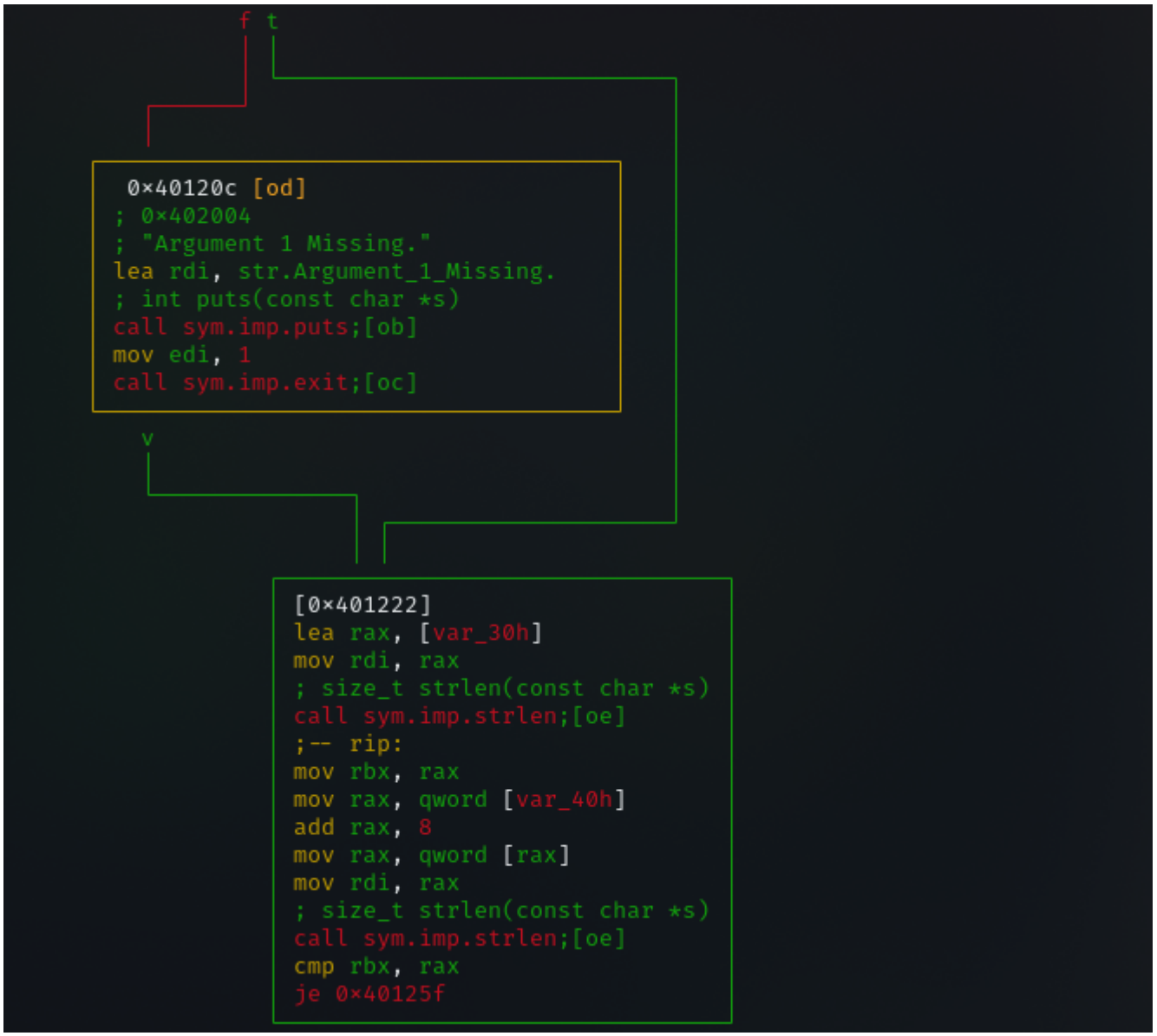


so the value is 1 . so we comparing 1 & (1 meaning the arguments). so, we have to give one argument if we so not give it will print argument missing.



Step 4:

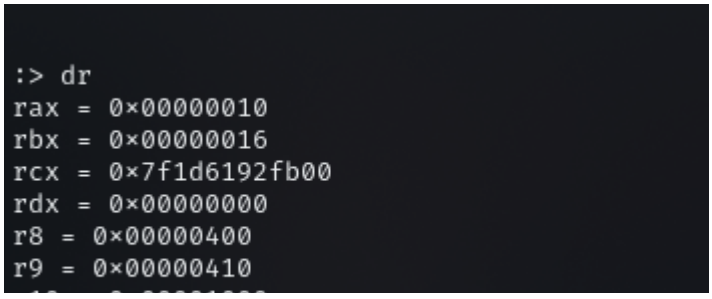
now lets rerun the program with arguments 9 lenght.



here cmp is comparing password length and our input string length.well how to i know it?

to know about register value value we type -dr

to know about more information we type -drr(hex to decimel value)



```
r10 = 0x00001000
r11 = 0x00000202
r12 = 0x00000000
r13 = 0x7ffcde19de70
r14 = 0x00000000
r15 = 0x7f1d61a66000
rsi = 0x00f492a0
rdi = 0x7f1d61a0da30
rsp = 0x7ffcde19dd00
rbp = 0x7ffcde19dd40
rip = 0x00401255
rflags = 0x00000202
orax = 0xffffffffffffffff
```

'rax' is our input string length value and 'rbx' is password length value.(10 and 22 in decimal)

now re-run the program with 22 string argument

now we go to the next level.

if password is same it print correct password if is wrong it prints wrong password.

now we have to know the rsi and rdu value.

we can use ->dr command.

```
:> drr
role reg     value           refstr
-------------------------------------
R0   rax     7ffdc066ba70    [stack] rax,rdi stack R W 0x535f726570753353 S3uper_S3cr3t_Pa33w0rd
     rbx     16              22 .comment rbx
A3   rcx     7f33c4b24840    /usr/lib/x86_64-linux-gnu/libc.so.6 rcx library R W 0x7f33c4b26300
A2   rdx     7ffdc066cf0f    [stack] rdx,rsi stack R W 0x3837363534333231 1234567890123456789012
A4   r8      401330          4199216 /home/luchifer/ctf/reverse/crackme1 .text __libc_csu_fini,r8 sym.__libc_csu_fini program R X 'endbr64' 'crackme1'
A5   r9      7f33c4b51b10    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2 r9 library R X 'push r15' 'ld-linux-x86-64.so.2'
     r10     7f33c4960328    /usr/lib/x86_64-linux-gnu/libc.so.6 r10 library R 0x10001a00007068 hp
     r11     7f33c4aa6140    /usr/lib/x86_64-linux-gnu/libc.so.6 r11 library R X 'mov eax, edi' 'libc.so.6'
     r12     0               0
     r13     7ffdc066bbd0    [stack] r13 stack R W 0x7ffdc066cf26
     r14     0               0
     r15     7f33c4b7f000    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2 r15 library R W 0x7f33c4b802d0
A1   rsi     7ffdc066cf0f    [stack] rdx,rsi stack R W 0x3837363534333231 1234567890123456789012
A0   rdi     7ffdc066ba70    [stack] rax,rdi stack R W 0x535f726570753353 S3uper_S3cr3t_Pa33w0rd
SP   rsp     7ffdc066ba60    [stack] rsp stack R W 0x7ffdc066bbb8
BP   rbp     7ffdc066baa0    [stack] rbp stack R W 0x2
PC   rip     401274          4199028 /home/luchifer/ctf/reverse/crackme1 .text rip main program R X 'call 0x4010b0' 'crackme1'
     cs      33              51 .symtab ascii ('3')
     rflags  216             534 .symtab rflags
```

here rsi value is **123456789012345678901**
and rdi value is **S3uper_S3cr3t_Pa33w0rd**

so,we get our password. it is "**S3uper_S3cr3t_Pa33w0rd**"

the program we reverse is

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char const *argv[])
{
        char password[] = "S3uper_S3cr3t_Pa33w0rd";
        if (argc < 2) {
                puts("Argument 1 Missing.");
                exit(1);
        }
        if (strlen(password) != strlen(argv[1])) {
                puts("Wrong Password.");
                exit(1);
        }
        if (!(strcmp(password,argv[1]))) {
                puts("Correct Password.");
        }
        else {
                puts("Wrong Password.");
        }
        return 0;
}
```

credit: https://www.youtube.com/watch?v=IXNWmc1u3Kg&t=1481s