Reverse Engineering to extract password from binary files

3rdcrackme

We've been given with a crack me file from which we must retrieve the password which is embedded in the memory. This file is a binary, so you can't retrieve it by normal means.

For basic knowledge, let us consider a main.c file. For a machine to understand the written code, the file is converted to an object file (.o extension) and further converted to a binary executable. To ensure the following, we do the following.

```
┌─[paraxor@parrot]─[~/Downloads/GCI-fedora]
└──╼ $rabin2 -I 3rdcrackme
arch     x86
baddr    0x0
binsz    14922
bintype  elf
bits     64
canary   false
class    ELF64
compiler GCC: (Debian 9.2.1-19) 9.2.1 20191109
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  AMD x86-64 architecture
maxopsz  16
minopsz  1
nx       true
os       linux
pcalign  0
pic      true
relocs   true
relro    partial
rpath    NONE
sanitiz  false
static   false
stripped false
subsys   linux
va       true
```

I've played reverse engineering challenges before playing Google Code-In, so I have a set of processes in order to understand the binary.

```
└─ $rabin2 -z 3rdcrackme
[Strings]
Num Paddr      Vaddr      Len Size Section  Type  String
000 0x00002008 0x00002008  30  31 (.rodata) ascii You found the secret function!
001 0x00002027 0x00002027   9  10 (.rodata) ascii Congrats!
002 0x00002038 0x00002038  35  36 (.rodata) ascii The password is: FEDORAPASSWORDGCI!
003 0x0000205c 0x0000205c  20  21 (.rodata) ascii Enter the password!
004 0x00002071 0x00002071  22  23 (.rodata) ascii \nChecking password...\n
005 0x00002088 0x00002088  33  34 (.rodata) ascii Successfully logged in!\nGood job!
006 0x000020aa 0x000020aa  13  14 (.rodata) ascii Login failed!
```

I'm not gonna fall for this! Lets dig deeper.

We use gdb-peda for the following. Gdb is a debugger and is inbuilt in Linux distributions. What I've used is an extension for gdb intended for binary exploitation purposes.

```
gdb-peda$ info functions
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001030  puts@plt
0x0000000000001040  printf@plt
0x0000000000001050  memcmp@plt
0x0000000000001060  gets@plt
0x0000000000001070  exit@plt
0x0000000000001080  __cxa_finalize@plt
0x0000000000001090  _start
0x00000000000010c0  deregister_tm_clones
0x00000000000010f0  register_tm_clones
0x0000000000001130  __do_global_dtors_aux
0x0000000000001170  frame_dummy
0x0000000000001175  secret
0x00000000000011a7  main
0x0000000000001260  __libc_csu_init
0x00000000000012c0  __libc_csu_fini
0x00000000000012c4  _fini
```

We see two functions in the following functions list, lets look at them.

Secret:

```
gdb-peda$ disassemble secret
Dump of assembler code for function secret:
   0x0000000000001175 <+0>:     push   rbp
   0x0000000000001176 <+1>:     mov    rbp,rsp
   0x0000000000001179 <+4>:     lea    rdi,[rip+0xe88]        # 0x2008
   0x0000000000001180 <+11>:    call   0x1030 <puts@plt>
   0x0000000000001185 <+16>:    lea    rdi,[rip+0xe9b]        # 0x2027
   0x000000000000118c <+23>:    call   0x1030 <puts@plt>
   0x0000000000001191 <+28>:    lea    rdi,[rip+0xea0]        # 0x2038
   0x0000000000001198 <+35>:    call   0x1030 <puts@plt>
   0x000000000000119d <+40>:    mov    edi,0x0
   0x00000000000011a2 <+45>:    call   0x1070 <exit@plt>
End of assembler dump.
```

Main:

```
Dump of assembler code for function main:
   0x00000000000011a7 <+0>:     push   rbp
   0x00000000000011a8 <+1>:     mov    rbp,rsp
   0x00000000000011ab <+4>:     sub    rsp,0x40
   0x00000000000011af <+8>:     movabs rax,0x306b403136673030
   0x00000000000011b9 <+18>:    movabs rdx,0x313531646e616c30
   0x00000000000011c3 <+28>:    mov    QWORD PTR [rbp-0x20],rax
   0x00000000000011c7 <+32>:    mov    QWORD PTR [rbp-0x18],rdx
   0x00000000000011cb <+36>:    mov    DWORD PTR [rbp-0x10],0x6c656334
   0x00000000000011d2 <+43>:    mov    WORD PTR [rbp-0xc],0x21
   0x00000000000011d8 <+49>:    mov    DWORD PTR [rbp-0x4],0x0
   0x00000000000011df <+56>:    lea    rdi,[rip+0xe76]        # 0x205c
   0x00000000000011e6 <+63>:    mov    eax,0x0
   0x00000000000011eb <+68>:    call   0x1040 <printf@plt>
   0x00000000000011f0 <+73>:    lea    rax,[rbp-0x40]
   0x00000000000011f4 <+77>:    mov    rdi,rax
   0x00000000000011f7 <+80>:    mov    eax,0x0
   0x00000000000011fc <+85>:    call   0x1060 <gets@plt>
   0x0000000000001201 <+90>:    lea    rcx,[rbp-0x20]
   0x0000000000001205 <+94>:    lea    rax,[rbp-0x40]
   0x0000000000001209 <+98>:    mov    edx,0x16
   0x000000000000120e <+103>:   mov    rsi,rcx
   0x0000000000001211 <+106>:   mov    rdi,rax
   0x0000000000001214 <+109>:   call   0x1050 <memcmp@plt>
   0x0000000000001219 <+114>:   test   eax,eax
   0x000000000000121b <+116>:   jne    0x1224 <main+125>
   0x000000000000121d <+118>:   mov    DWORD PTR [rbp-0x4],0x1
   0x0000000000001224 <+125>:   lea    rdi,[rip+0xe46]        # 0x2071
   0x000000000000122b <+132>:   call   0x1030 <puts@plt>
   0x0000000000001230 <+137>:   cmp    DWORD PTR [rbp-0x4],0x0
   0x0000000000001234 <+141>:   je     0x124c <main+165>
   0x0000000000001236 <+143>:   lea    rdi,[rip+0xe4b]        # 0x2088
   0x000000000000123d <+150>:   call   0x1030 <puts@plt>
   0x0000000000001242 <+155>:   mov    edi,0x0
   0x0000000000001247 <+160>:   call   0x1070 <exit@plt>
   0x000000000000124c <+165>:   lea    rdi,[rip+0xe57]        # 0x20aa
   0x0000000000001253 <+172>:   call   0x1030 <puts@plt>
   0x0000000000001258 <+177>:   mov    eax,0x0
   0x000000000000125d <+182>:   leave
   0x000000000000125e <+183>:   ret
End of assembler dump.
```

From the following functions, one can deduce that the secret function is of no use :v

I switched to r2 for a detailed analysis.

```
    0x000011a7      55              push rbp
    0x000011a8      4889e5          mov rbp, rsp
    0x000011ab      4883ec40        sub rsp, 0x40
    0x000011af      48b830306736.   movabs rax, 0x306b403136673030 ; '00g61@k0'
    0x000011b9      48ba306c616e.   movabs rdx, 0x313531646e616c30 ; '0land151'
    0x000011c3      488945e0        mov qword [var_20h], rax
    0x000011c7      488955e8        mov qword [var_18h], rdx
    0x000011cb      c745f0346365.   mov dword [var_10h], 0x6c656334 ; '4cel'
    0x000011d2      66c745f42100    mov word [var_ch], 0x21     ; '!'
    0x000011d8      c745fc000000.   mov dword [var_4h], 0
    0x000011df      488d3d760e00.   lea rdi, qword str.Enter_the_password ; 0x205c ; "Enter the password! " ; const char *format
    0x000011e6      b800000000      mov eax, 0
    0x000011eb      e850feffff      call sym.imp.printf         ; int printf(const char *format)
    0x000011f0      488d45c0        lea rax, qword [var_40h]
    0x000011f4      4889c7          mov rdi, rax                ; char *s
    0x000011f7      b800000000      mov eax, 0
    0x000011fc      e85ffeffff      call sym.imp.gets           ; char *gets(char *s)
    0x00001201      488d4de0        lea rcx, qword [var_20h]
    0x00001205      488d45c0        lea rax, qword [var_40h]
    0x00001209      ba16000000      mov edx, 0x16               ; rdx ; size_t n
    0x0000120e      4889ce          mov rsi, rcx                ; const void *s2
    0x00001211      4889c7          mov rdi, rax                ; const void *s1
    0x00001214      e837feffff      call sym.imp.memcmp         ; int memcmp(const void *s1, const void *s2, size_t n)
    0x00001219      85c0            test eax, eax
,=< 0x0000121b      7507            jne 0x1224
|   0x0000121d      c745fc010000.   mov dword [var_4h], 1
|   ; CODE XREF from main @ 0x121b
`-> 0x00001224      488d3d460e00.   lea rdi, qword str.Checking_password... ; 0x2071 ; "\nChecking password...\n" ; const char *s
    0x0000122b      e800feffff      call sym.imp.puts           ; int puts(const char *s)
    0x00001230      837dfc00        cmp dword [var_4h], 0
,=< 0x00001234      7416            je 0x124c
|   0x00001236      488d3d4b0e00.   lea rdi, qword str.Successfully_logged_in__Good_job ; 0x2088 ; "Successfully logged in!\nGood job!" ; const char *s
|   0x0000123d      e8eefdffff      call sym.imp.puts           ; int puts(const char *s)
|   0x00001242      bf00000000      mov edi, 0                  ; int status
|   0x00001247      e824feffff      call sym.imp.exit           ; void exit(int status)
|   ; CODE XREF from main @ 0x1234
`-> 0x0000124c      488d3d570e00.   lea rdi, qword str.Login_failed ; 0x20aa ; "Login failed!" ; const char *s
    0x00001253      e8d8fdffff      call sym.imp.puts           ; int puts(const char *s)
    0x00001258      b800000000      mov eax, 0
    0x0000125d      c9              leave
    0x0000125e      c3              ret
```

This crackme is THE same as the 2ndcrackme. I expected some heavy recursion bitwise operators, but alas. Task Completed.

```
└─$./3rdcrackme
Enter the password! 00g61@k00land1514cel!

Checking password...

Successfully logged in!
Good job!
```