LAB1

CTF: CRACKMES01 BY GEYSLAN

- We will refer to the crackme.01 challenge (https://github.com/geyslan/crackmes/blob/master/src/crackme.01.c)

```
$ ./crackme.01
Please tell me my password: AAA
No! No! No! No! Try again.
```

- Let's search for the password…

```
$ file crackme.01
crackme.01: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=e0972d05468b45df0d006749f8c3426afe082d9a, not stripped
```

- A binary file dynamically linked (not stripped…info about the starting point available)

LOOKING FOR STRINGS…

→ why opening a file?!?

→Uses ptrace() …couldbe antidebugging…

→2 abutidebugging messages…two different techniques … could be the fopen

→The good message

→The bad message

→Password is in memory…somehow

→Here it is…

→A compare function…

```
$ strings crackme.01
/lib64/ld-linux-x86-64.so.2
libc.so.6

fopen
…
ptrace
…
I'm sorry GDB! You are not allowed!
Tracing is not allowed... Bye
Please tell me my password:
The password is correct!
Congratulations!!!
No! No! No! No! Try again.
;*3$"
…
passwd
…
__libc_csu_init
ptrace@@GLIBC_2.2.5
__bss_start
main
detect_gdb
…
compare
fwrite@@GLIBC_2.2.5
__TMC_END__
```

LET'S TRY TO LTRACE AND STRACE


```
$ ltrace ./crackme.01
__libc_start_main(0x400930, 1, 0x7fffcfbb6ff8, 0x400a00 <unfinished ...>
fopen("/tmp", "r") = 0x16b1010
fileno(0x16b1010) = 3
fclose(0x16b1010) = 0
ptrace(0, 0, 1, 0) = -1
puts("Tracing is not allowed... Bye"Tracing is not allowed... Bye) = 30
exit(1 <no return ...>
+++ exited (status 1) +++
```

- Antidebugging…in action

```
$ strace ./crackme.01
execve("./crackme.01", ["./crackme.01"], [/* 18 vars */]) = 0
...
open("/tmp", O_RDONLY)                  = 3
close(3)                                = 0
ptrace(PTRACE_TRACEME, 0, 0x1, NULL)    = -1 EPERM (Operation not permitted)
fstat(1, {st_mode=S_IFCHR|0660, st_rdev=makedev(4, 1), ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
write(1, "Tracing is not allowed... Bye\n", 30Tracing is not allowed... Bye) = 30
exit_group(1)                           = ?
+++ exited with 1 +++
```

BYPASSING THE ANTIDEBUG

- Debugging in gdb…fails as expected

```
$gdb ./crackme.01
gdb-peda$ start
Tracing is not allowed... Bye
```

- Let's try to overl

```
$ gdb ./crackme.01
gdb-peda$ set environment LD_PRELOAD=./fakeptrace.so
gdb-peda$ start
gdb-peda$ c
Continuing.
Please tell me my password:
```

DISASSEMBLE MAIN


```
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0000000000400930 <+0>:     push   rbp
   0x0000000000400931 <+1>:     mov    rbp,rsp
   0x0000000000400934 <+4>:     sub    rsp,0x10
   0x0000000000400938 <+8>:     mov    rax,QWORD PTR fs:0x28
   0x0000000000400941 <+17>:    mov    QWORD
   0x0000000000400945 <+21>:    xor    eax,ea
   0x0000000000400947 <+23>:    mov    rax,QW
   0x000000000040094e <+30>:    mov    rcx,ra
   0x0000000000400951 <+33>:    mov    edx,0x
   0x0000000000400956 <+38>:    mov    esi,0x
   0x000000000040095b <+43>:    mov    edi,0x
   0x0000000000400960 <+48>:    call   0x4006
   0x0000000000400965 <+53>:    mov    rdx,QW
   0x000000000040096c <+60>:    lea    rax,[r
   0x0000000000400970 <+64>:    mov    esi,0x
   0x0000000000400975 <+69>:    mov    rdi,ra
   0x0000000000400978 <+72>:    call   0x4006
   0x000000000040097d <+77>:    lea    rax,[r
   0x0000000000400981 <+81>:    mov    rdi,ra
   0x0000000000400984 <+84>:    call   0x4008
   0x0000000000400989 <+89>:    lea    rax,[r
   0x000000000040098d <+93>:    mov    esi,0x
   0x0000000000400992 <+98>:    mov    rdi,ra
   0x0000000000400995 <+101>:   call   0x4008
   0x000000000040099a <+106>:   test   eax,ea
   0x000000000040099c <+108>:   jne    0x4009
   0x000000000040099e <+110>:   mov    rax,QW
   0x00000000004009a5 <+117>:   mov    rcx,ra
   0x00000000004009a8 <+120>:   mov    edx,0x
   0x00000000004009ad <+125>:   mov    esi,0x
   0x00000000004009b2 <+130>:   mov    edi,0x
   0x00000000004009b7 <+135>:   call   0x4006
   0x00000000004009bc <+140>:   jmp    0x4009
   0x00000000004009be <+142>:   mov    rax,QW
   0x00000000004009c5 <+149>:   mov    rcx,ra
   0x00000000004009c8 <+152>:   mov    edx,0x
   0x00000000004009cd <+157>:   mov    esi,0x
   0x00000000004009d2 <+162>:   mov    edi,0x
   0x00000000004009d7 <+167>:   call   0x4006
   0x00000000004009dc <+172>:   mov    eax,0x
   0x00000000004009e1 <+177>:   mov    rcx,QW
   0x00000000004009e5 <+181>:   xor    rcx,QWORD PTR fs:0x28
   0x00000000004009ee <+190>:   je     0x4009f5 <main+197>
   0x00000000004009f0 <+192>:   call   0x400680 <__stack_chk_fail@plt>
   0x00000000004009f5 <+197>:   leave
   0x00000000004009f6 <+198>:   ret
End of assembler dump.
```

```
                 mov    rdx,QWORD PTR [rip+0x200724]
                 lea    rax,[rbp-0x10]
                 mov    esi,0x7
                 mov    rdi,rax
                 call   0x4006a0 <fgets@plt>
                 lea    rax,[rbp-0x10]
                 mov    rdi,rax
                 call   0x40088c <xor>
                 lea    rax,[rbp-0x10]
                 mov    esi,0x601078
                 mov    rdi,rax
                 call   0x4008cc <compare>
                 test   eax,eax
                 jne    0x4009be <main+142>
                 mov    rax,QWORD PTR [rip+0x2006d
```

<- read user

<-uses the input in a xor function

<- compare the result with something @0x601078

LET'S HAVE A LOOK TO THE XOR FUNCTION

```
ler code for function xor:
040088c <+0>:     push   rbp
040088d <+1>:     mov    rbp,rsp
0400890 <+4>:     mov    QWORD PTR [rbp-0x18],rdi
0400894 <+8>:     mov    DWORD PTR [rbp-0x4],0x0
040089b <+15>:    jmp    0x4008c3 <xor+55>
040089d <+17>:    mov    eax,DWORD PTR [rbp-0x4]
04008a0 <+20>:    movsxd rdx,eax
04008a3 <+23>:    mov    rax,QWORD PTR [rbp-0x18]
04008a7 <+27>:    add    rax,rdx
04008aa <+30>:    mov    edx,DWORD PTR [rbp-0x4]
04008ad <+33>:    movsxd rcx,edx
04008b0 <+36>:    mov    rdx,QWORD PTR [rbp-0x18]
04008b4 <+40>:    add    rdx,rcx
04008b7 <+43>:    movzx  edx,BYTE PTR [rdx]
04008ba <+46>:    xor    edx,0x6c
04008bd <+49>:    mov    BYTE PTR [rax],dl
04008bf <+51>:    add    DWORD PTR [rbp-0x4],0x1
04008c3 <+55>:    cmp    DWORD PTR [rbp-0x4],0x5
04008c7 <+59>:    jle    0x40089d <xor+17>
04008c9 <+61>:    nop
04008ca <+62>:    pop    rbp
04008cb <+63>:    ret
```

<- jump to xor+55

<-Read from input + counter 1 byte

<-xor with 0x6c

<- inc. counter

<-compares [rbp-0x4] with 5

If less or equal …repeat loop

LET'S HAVE A LOOK TO THE XOR FUNCTION

```
ler code for function xor:
040088c <+0>:     push   rbp
040088d <+1>:     mov    rbp,rsp
0400890 <+4>:     mov    QWORD PTR [rbp-0x18],rdi
0400894 <+8>:     mov    DWORD PTR [rbp-0x4],0x0
040089b <+15>:    jmp    0x4008c3 <xor+55>
040089d <+17>:    mov    eax,DWORD PTR [rbp-0x4]
04008a0 <+20>:
04008a3 <+23>:
04008a7 <+27>:
04008aa <+30>:
04008ad <+33>:
04008b0 <+36>:
04008b4 <+40>:
04008b7 <+43>:
04008ba <+46>:    xor    edx,0x6c
04008bd <+49>:    mov    BYTE PTR [rax],dl
04008bf <+51>:    add    DWORD PTR [rbp-0x4],0x1
04008c3 <+55>:    cmp    DWORD PTR [rbp-0x4],0x5
04008c7 <+59>:    jle    0x40089d <xor+17>
04008c9 <+61>:    nop
04008ca <+62>:    pop    rbp
04008cb <+63>:    ret
```

<- jump to xor+55

It seems that every character from the input is xor-ed with 0x6c before being compared with the passwd

om input +
r 1 byte

<-xor with 0x6c

<- inc. counter

<-compares [rbp-0x4] with 5

If less or equal …repeat loop

DISASSEMBLE MAIN

```
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0000000000400930 <+0>:     push   rbp
   0x0000000000400931 <+1>:     mov    rbp,rsp
   0x0000000000400934 <+4>:     sub    rsp,0x10
   0x0000000000400938 <+8>:     mov    rax,QWORD PTR fs:0x28
   0x0000000000400941 <+17>:    mov    QWORD
   0x0000000000400945 <+21>:    xor    eax,ea
   0x0000000000400947 <+23>:    mov    rax,QW
   0x000000000040094e <+30>:    mov    rcx,ra
   0x0000000000400951 <+33>:    mov    edx,0x
   0x0000000000400956 <+38>:    mov    esi,0x
   0x000000000040095b <+43>:    mov    edi,0x
   0x000000000040098d <+93>:    mov    esi,0x
   0x0000000000400992 <+98>:    mov    rdi,ra
   0x0000000000400995 <+101>:   call   0x4008
   0x000000000040099a <+106>:   test   eax,ea
   0x000000000040099c <+108>:   jne    0x4009
   0x000000000040099e <+110>:   mov    rax,QW
   0x00000000004009a5 <+117>:   mov    rcx,ra
   0x00000000004009a8 <+120>:   mov    edx,0x
   0x00000000004009ad <+125>:   mov    esi,0x
   0x00000000004009b2 <+130>:   mov    edi,0x
   0x00000000004009b7 <+135>:   call   0x4006
   0x00000000004009bc <+140>:   jmp    0x4009
   0x00000000004009be <+142>:   mov    rax,QW
   0x00000000004009c5 <+149>:   mov    rcx,ra
   0x00000000004009c8 <+152>:   mov    edx,0x
   0x00000000004009cd <+157>:   mov    esi,0x
   0x00000000004009d2 <+162>:   mov    edi,0x
   0x00000000004009d7 <+167>:   call   0x4006
   0x00000000004009dc <+172>:   mov    eax,0x
   0x00000000004009e1 <+177>:   mov    rcx,QW
   0x00000000004009e5 <+181>:   xor    rcx,QWORD PTR fs:0x28
   0x00000000004009ee <+190>:   je     0x4009f5 <main+197>
   0x00000000004009f0 <+192>:   call   0x400680 <__stack_chk_fail@plt>
   0x00000000004009f5 <+197>:   leave
   0x00000000004009f6 <+198>:   ret
End of assembler dump.
```

```
                 rdx,QWORD PTR [rip+0x200724]
lea              rax,[rbp-0x10]
mov              esi,0x7
                                              t>
call             0x40088c <xor>
lea              rax,[rbp-0x10]
mov              esi,0x601078
mov              rdi,rax
call             0x4008cc <compare>
test             eax,eax
jne              0x4009be <main+142>
mov              rax,QWORD PTR [rip+0x2006d
```

The compare function takes the xor-ed input and a “passwd”

<- read user

<-uses the input in a xor function

<- compare the result with something @0x601078

LET'S HAVE A LOOK TO THE COMPARE FUNCTION


```
ler code for function compare:
04008cc <+0>:     push   rbp
04008cd <+1>:     mov    rbp,rsp
04008d0 <+4>:     mov    QWORD PTR [rbp-0x8],rdi
04008d4 <+8>:     mov    QWORD PTR [rbp-0x10],rsi
04008d8 <+12>:    jmp    0x4008fa <compare+46>
04008da <+14>:    mov    rax,QWORD PTR [rbp-0x8]
04008de <+18>:    movzx  eax,BYTE PTR [rax]
04008e1 <+21>:    test   al,al
04008e3 <+23>:    je     0x40090c <compare+64>
04008e5 <+25>:    mov    rax,QWORD PTR [rbp-0x10]
04008e9 <+29>:    movzx  eax,BYTE PTR [rax]
04008ec <+32>:    test   al,al
04008ee <+34>:    je     0x40090c <compare+64>
04008f0 <+36>:    add    QWORD PTR [rbp-0x8],0x1
04008f5 <+41>:    add    QWORD PTR [rbp-0x10],0x1
04008fa <+46>:    mov    rax,QWORD PTR [rbp-0x8]
04008fe <+50>:    movzx  edx,BYTE PTR [rax]
0400901 <+53>:    mov    rax,QWORD PTR [rbp-0x10]
0400905 <+57>:    movzx  eax,BYTE PTR [rax]
0400908 <+60>:    cmp    dl,al
040090a <+62>:    je     0x4008da <compare+14>
040090c <+64>:    mov    rax,QWORD PTR [rbp-0x8]
0400910 <+68>:    movzx  eax,BYTE PTR [rax]
0400913 <+71>:    test   al,al
0400915 <+73>:    jne    0x400929 <compare+93>
0400917 <+75>:    mov    rax,QWORD PTR [rbp-0x10]
040091b <+79>:    movzx  eax,BYTE PTR [rax]
040091e <+82>:    test   al,al
0400920 <+84>:    jne    0x400929 <compare+93>
0400922 <+86>:    mov    eax,0x0
0400927 <+91>:    jmp    0x40092e <compare+98>
0400929 <+93>:    mov    eax,0xffffffff
040092e <+98>:    pop    rbp
040092f <+99>:    ret
```

- The two inputs are compared until equal or end-of-string
- Let's set a breakpoint at the start of compare and check the inputs

```
gdb-peda$ break *compare
Breakpoint 2 at 0x4008cc
gdb-peda$ start
gdb-peda$ c
Continuing.
Please tell me my password: aaa
Breakpoint 2, 0x00000000004008cc in
compare ()
gdb-peda$
```

RETRIEVING THE PASSWORD

- At this point RSI should point at the xor-ed password and RDI at our input xor-ed…

```
RSI: 0x601078 --> 0x185c0215041b
RDI: 0x7fffffffee2e0
```

- let's check it…

```
gdb-peda$ x/4db $rdi
0x7fffffffee2e0: 13      13      13      102
```

- 13^108 (0x6c) = 97 -> ascii for 'a' ….GREAT
- Thus the xor-ed password is

```
gdb-peda$ x/10db 0x601078
0x601078 <passwd>:      27      4       21      2       92      24      0       0
gdb-peda$ q
$ python2 -c "print(chr(27^108)+chr(4^108)
+chr(21^108)+chr(2^108)+chr(92^108)+chr(24^108))"
whyn0t
```

```
$ ./crackme.01
Please tell me my password: whyn0t
The password is correct!
Congratulations!!!
```

Lab2

CTF: CRACKMES02 BY GEYSLAN

- We will refer to the crackme.02.64 challenge a modified version of crackme.02 by Geysaln (https://github.com/geyslan/crackmes/blob/master/src/crackme.02.c), for x64 architectures
- Additional challenges can be found at https://github.com/geyslan/crackmes

```
./crackme.02
Please tell me my password: aaa
No! No! No! No! Try again.
```

SOME STATIC ANALYSIS

- ELF has been compromised

```
$ file crackme.02
crackme.02: ELF 64-bit LSB executable, x86-64, (SYSV), too many
section (65535)
```

- Let's try with radare2 toolset

```
$ rabin2 -I ./crackme.02
Warning: Cannot initialize section headers
Warning: Cannot initialize strings table
Warning: Cannot initialize dynamic strings
bintype  elf
bits     64
canary   true
class    ELF64
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
nx       true
os       linux
pcalign  0
pic      false
relocs   true
relro    partial
rpath    NONE
static   false
stripped false
subsys   linux
va       true
```

CONTINUING WITH STATIC

- File is not stripped, we can get the starting point

```
$ rabin2 -M ./crackme.02
[Main]
vaddr=0x00400967 paddr=0x00000967
```

- Let's have a look to strings… (strings ./crackme.02)

```
$ rabin2 -zz ./crackme.02
000 0x00000238 0x00400238  27  28 (INTERP) ascii /lib64/ld-linux-x86-64.so.2
001 0x000003f9 0x004003f9   9  10 (LOAD0) ascii libc.so.6
...
003 0x00000408 0x00400408   5   6 (LOAD0) ascii fopen
...
026 0x00000ac0 0x00400ac0  35  36 (LOAD0) ascii I'm sorry GDB! You are not allowed!
027 0x00000ae8 0x00400ae8  30  31 (LOAD0) ascii Tracing is not allowed... Bye!
028 0x00000b07 0x00400b07  28  29 (LOAD0) ascii Please tell me my password:
029 0x00000b28 0x00400b28  44  45 (LOAD0) ascii The password is correct!\nCongratulations!!!\n
030 0x00000b55 0x00400b55  27  28 (LOAD0) ascii No! No! No! No! Try again.\n
...
057 0x000019c6 0x000019c6   6   7 () ascii passwd
...
060 0x000019f2 0x000019f2   9  10 () ascii antidebug
...
075 0x00001adb 0x00001adb   4   5 () ascii main
...
080 0x00001b20 0x00001b20   7   8 () ascii compare
...
```

CONTINUING WITH STATIC

- File is not stripped, we can get the starting point

```
$ rabin2 -M ./crackme.02
[Main]
vaddr=0x00400967 paddr=0x00000967
```

- Let's have a look to strings… (strings ./crackme.02)

```
$ rabin2 -zz ./crackme.02
000 0x00000238 0x00400238  27  28 (INTERP) ascii /lib64/ld-linux-x86-64.so.2
001 0x000003f9 0x004003f9   9  10 (LOAD0) ascii libc.so.6
...
003 0x00000408 0x00400408   5   6 (LOAD0) ascii fopen
...
026 0x00000ac0 0x00400ac0  35  36 (LOAD0) ascii I'm sorry GDB! You are not allowed!
027 0x00000ae8 0x00400ae8  30  31 (LOAD0) ascii Tracing is not allowed... Bye!
028 0x00000b07 0
029 0x00000b28 0                                                    ongratulations!!!\n
030 0x00000b55 0
...
057 0x000019c6 0
...
060 0x000019f2 0x000019f2   9  10 () ascii antidebug
...
075 0x00001adb 0x00001adb   4   5 () ascii main
...
080 0x00001b20 0x00001b20   7   8 () ascii compare
...
```

No ptrace…but still two messages for antidebugging
Again we have dynamic linking

LET'S LTRACE AND STRACE

- …we can strace/ltrace and see that antidebug is still there…

```
$ strace ./crackme.02
...
open("/tmp", O_RDONLY)                  = 3
close(3)                                = 0
ptrace(PTRACE_TRACEME, 0, NULL, NULL)   = -1 EPERM (Operation not permitted)
fstat(1, {st_mode=S_IFCHR|0660, st_rdev=makedev(4, 1), ...}) = 0
ioctl(1, TCGETS, {B38400 opost isig icanon echo ...}) = 0
```

- If we try to open with r2 and break at main…we are kicked off

```
$ r2 -Ad ./crackme.02
[0x7ff37da00c30]> db main
[0x7ff37da00c30]> dc
Tracing is not allowed... Bye!
[0x7ff37d6fc748]>
```

HUNTING ANTIDEBUGGING FUNCTION

- Let's check for accesses to the antidebugging output message

```
$ r2 -Ad ./crackme.02
[0x7f12daa00c30]> izz
026 0x00000ac0 0x00400ac0  35  36 (LOAD0) ascii I'm sorry GDB! You are not ...
027 0x00000ae8 0x00400ae8  30  31 (LOAD0) ascii Tracing is not allowed... Bye!
...
```

- We can check references to the address

```
[0x7f12daa00c30]> axt 0x00400ae8
[0x7f12daa00c30]>
```

- Nothing is found…let's check for the address in the binary..(it is little endian)

```
[0x7febac200c30]> s main
[0x00400967]> /x e80a40
Searching 3 bytes in [0x400000-0x401000]
hits: 1
0x00400862 hit0_0 e80a40
```

JUMPING IN THE ANTIDEBUG

- Seek at the hitting address (`s 0x00400862`) and turn in Visual mode (Vp)
- Turn in cursor mode (c) and explore upward …you will find:
  - A reference to the debugger message 2 (0x400ae8) “Tracing is not allowed... Bye!“
  - A reference to the debugger message 1 (0x400ac0) “I'm sorry GDB! You are not allowed!”
    - Both references are followed by the printing of the message…
  - The start of a function @0x004007b6 …this should be the debugging function

```
0x004007b6   *  55              push rbp
0x004007b7      4889e5          mov rbp, rsp
0x004007ba      4883ec10        sub rsp, 0x10
0x004007be      beb80a4000      mov esi, 0x400ab8
0x004007c3      bfba0a4000      mov edi, 0x400aba
0x004007c8      e8b3feffff      call sym.imp.fopen
0x004007cd      488945f8        mov qword [rbp - 8], rax
0x004007d1      488b45f8        mov rax, qword [rbp - 8]
0x004007d5      4889c7          mov rdi, rax
0x004007d8      e893feffff      call sym.imp.fileno
0x004007dd      83f805          cmp eax, 5
,=< 0x004007e0  7e14            jle 0x4007f6
|   0x004007e2  bfc00a4000      mov edi, 0x400ac0
|   0x004007e7  e834feffff      call sym.imp.puts
```

- The end of the function is after the print of the second antidebugging message

```
,=< 0x0040085f      7914            jns 0x400875                ;[7]
|   0x00400861  ~   bfe80a4000      mov edi, 0x400ae8
|   ;-- hit0_0:
|   0x00400862      e80a4000e8      call 0xffffffffe840ba71     ;[8]
|   0x00400867      b5fd            mov ch, 0xfd                ; 253
|   0x00400869      ff              invalid
|   0x0040086a      ff              invalid
|   0x0040086b      bf01000000      mov edi, 1
|   0x00400870      e81bfeffff      call sym.imp.exit           ;[9] ; void exit(int status)
`-> 0x00400875      90              nop
    0x00400876      c9              leave
    0x00400877      c3              ret
```

<- here we are kicked off with the message

<- here the function finishes (0x00400877)

- Let's patch the function so that we enter and soon finish… @ 0x004007b6 hit A and write:
  - jmp 0x00400877 then Enter/enter to save exit the cursor mode (c) and hit F2 to put a breakpoint here

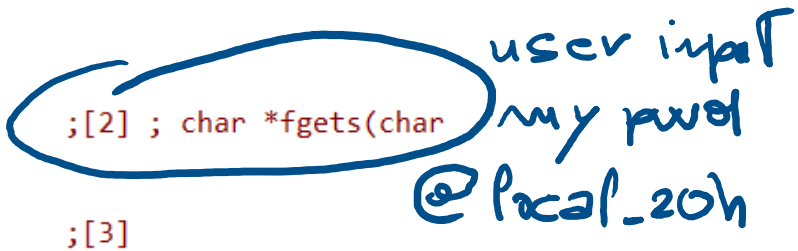
- Execute until the break point (: dc)

```
;-- rip:
,=< 0x004007b6 b    e9bc000000      jmp 0x400877
|   0x004007bb      83ec10          sub esp, 0x10
|   0x004007be      beb80a4000      mov esi, 0x400ab8
|   0x004007c3      bfba0a4000      mov edi, 0x400aba
```

- F7 to step-in F8 to step-out till reaching the main…we are in…

```
/ (fcn) main 199
|   main ();
|           ; var int local_20h @ rbp-0x20
|           ; var int local_8h @ rbp-0x8
|           ; DATA XREF from 0x004006dd (entry0)
|           0x00400967      55              push rbp
|           0x00400968      4889e5          mov rbp, rsp
|           ;-- rip:
|           0x0040096b      4883ec20        sub rsp, 0x20
|           0x0040096f      64488b042528.   mov rax, qword fs:[0x28]    ; [0x28:8]=-1 ; '(' ; 4
|           0x00400978      488945f8        mov qword [local_8h], rax
```

```
/ (fcn) main 199
|   main ();
|           ; var int local_20h @ rbp-0x20
|           ; var int local_8h @ rbp-0x8
|           ; DATA XREF from 0x004006dd (entry0)
|           0x00400967      55              push rbp
|           0x00400968      4889e5          mov rbp, rsp
|           ;-- rip:
|           0x0040096b      4883ec20        sub rsp, 0x20
|           0x0040096f      64488b042528.   mov rax, qword fs:[0x28]    ; [0x28:8]=-1 ; '(' ; 4
|           0x00400978      488945f8        mov qword [local_8h], rax
|           0x0040097c      31c0            xor eax, eax
|           0x0040097e      488b05fb0620.   mov rax, qword obj.stdout    ; [0x601080:8]=0x7f1dcb
|           0x00400985      4889c1          mov rcx, rax
|           0x00400988      ba1c000000      mov edx, 0x1c               ; 28
|           0x0040098d      be01000000      mov esi, 1
|           0x00400992      bf070b4000      mov edi, 0x400b07
|           0x00400997      e804fdffff      call sym.imp.fwrite         ;[1] ; size_t fwrite(cor
|           0x0040099c      488b15ed0620.   mov rdx, qword obj.stdin    ; [0x601090:8]=0x7f1dcbb
|           0x004009a3      488d45e0        lea rax, [local_20h]
|           0x004009a7      be0a000000      mov esi, 0xa
|           0x004009ac      4889c7          mov rdi, rax
|           0x004009af      e8acfcffff      call sym.imp.fgets          ;[2] ; char *fgets(char
|           0x004009b4      488d45e0        lea rax, [local_20h]
|           0x004009b8      4889c7          mov rdi, rax
|           0x004009bb      e8f1feffff      call 0x4008b1               ;[3]
|           0x004009c0      488d45e0        lea rax, [local_20h]
|           0x004009c4      be70106000      mov esi, 0x601070
|           0x004009c9      4889c7          mov rdi, rax
|           0x004009cc      e832ffffff      call 0x400903               ;[4]
|           0x004009d1      85c0            test eax, eax
|       ,=< 0x004009d3      7520            jne 0x4009f5                ;[5]
|       |   0x004009d5      488b05a40620.   mov rax, qword obj.stdout    ; [0x601080:8]=0x7f1dcb
|       |   0x004009dc      4889c1          mov rcx, rax
|       |   0x004009df      ba2c000000      mov edx, 0x2c               ; ',' ; 44
|       |   0x004009e4      be01000000      mov esi, 1
|       |   0x004009e9      bf280b4000      mov edi, 0x400b28
|       |   0x004009ee      e8adfcffff      call sym.imp.fwrite         ;[1] ; size_t fwrite(cor
|       `==< 0x004009f3      eb1e            jmp 0x400a13                ;[6]
```



```
    0x004009b4      488d45e0        lea rax, [local_20h]
    0x004009b8      4889c7          mov rdi, rax
    0x004009bb      e8f1feffff      call 0x4008b1                   ;[3]
    0x004009c0      488d45e0        lea rax, [local_20h]
    0x004009c4      be70106000      mov esi, 0x601070
    0x004009c9      4889c7          mov rdi, rax
    0x004009cc      e832ffffff      call 0x400903                   ;[4]
    0x004009d1      85c0            test eax, eax
,=< 0x004009d3      7520            jne 0x4009f5                    ;[5]
|   0x004009d5      488b05a40620.   mov rax, qword obj.stdout       ; [0x601080:8]=0x7f1dcbbf5620 ;
|   0x004009dc      4889c1          mov rcx, rax
|   0x004009df      ba2c000000      mov edx, 0x2c                   ; ',' ; 44
|   0x004009e4      be01000000      mov esi, 1
|   0x004009e9      bf280b4000      mov edi, 0x400b28
|   0x004009ee      e8adfcffff      call sym.imp.fwrite             ;[1] ; size_t fwrite(const void *
==< 0x004009f3      eb1e            jmp 0x400a13                    ;[6]
`-> 0x004009f5      488b05840620.   mov rax, qword obj.stdout       ; [0x601080:8]=0x7f1dcbbf5620 ;
    0x004009fc      4889c1          mov rcx, rax
    0x004009ff      ba1b000000      mov edx, 0x1b                   ; 27
    0x00400a04      be01000000      mov esi, 1
    0x00400a09      bf550b4000      mov edi, 0x400b55
    0x00400a0e      e88dfcffff      call sym.imp.fwrite             ;[1] ; size_t fwrite(const void *
    ; JMP XREF from 0x004009f3 (main)
--> 0x00400a13      b800000000      mov eax, 0
```

do something with mypwd

do something with result of 3 and something stored @0x601070

based on result of [4]

ok!

ko!

```
...
026 0x00000ac0 0x00400ac0  35  36 (LOAD0) ascii I'm sorry GDB! You are not allowed!
027 0x00000ae8 0x00400ae8  30  31 (LOAD0) ascii Tracing is not allowed... Bye!
028 0x00000b07 0x00400b07  28  29 (LOAD0) ascii Please tell me my password:
029 0x00000b28 0x00400b28  44  45 (LOAD0) ascii The password is correct!\nCongratulations!!!\n
030 0x00000b55 0x00400b55  27  28 (LOAD0) ascii No! No! No! No! Try again.\n
...
```

LET'S CHECK THE CONTENT OF FUNCTION [3] BY HITTING '3'

- rdi -> mypwd
- We can recognize a loop executed 9 times (length of the password)
  - [rbp-20]->mypwd
  - [rbp-0xc]->count
- Takes char by char @[mypwd+count]
- And invokes [2] passing the char
- Let's go deeper by hitting '2'

```
0x004008b1      55              push rbp
0x004008b2      4889e5          mov rbp, rsp
0x004008b5      53              push rbx
0x004008b6      4883ec18        sub rsp, 0x18
0x004008ba      48897de0        mov qword [rbp - 0x20], rdi
0x004008be      c745f4000000.   mov dword [rbp - 0xc], 0
,=< 0x004008c5      eb2e            jmp 0x4008f5                ;[1]
.--> 0x004008c7      8b45f4          mov eax, dword [rbp - 0xc]
:|  0x004008ca      4863d0          movsxd rdx, eax
:|  0x004008cd      488b45e0        mov rax, qword [rbp - 0x20]
:|  0x004008d1      488d1c02        lea rbx, [rdx + rax]
:|  0x004008d5      8b45f4          mov eax, dword [rbp - 0xc]
:|  0x004008d8      4863d0          movsxd rdx, eax
:|  0x004008db      488b45e0        mov rax, qword [rbp - 0x20]
:|  0x004008df      4801d0          add rax, rdx                ; '('
:|  0x004008e2      0fb600          movzx eax, byte [rax]
:|  0x004008e5      0fbec0          movsx eax, al
:|  0x004008e8      89c7            mov edi, eax
:|  0x004008ea      e889ffffff      call 0x400878               ;[2]
:|  0x004008ef      8803            mov byte [rbx], al
:|  0x004008f1      8345f401        add dword [rbp - 0xc], 1
:`-> 0x004008f5      837df408        cmp dword [rbp - 0xc], 8    ; [0x8:4
`==< 0x004008f9      7ecc            jle 0x4008c7                ;[3]
0x004008fb      90              nop
0x004008fc      4883c418        add rsp, 0x18
0x00400900      5b              pop rbx
```

- edi=the input char
- There is an antireverse trick: jmp in the middle of an instraction

the actual code is here…

again antireverse…

the real operation:
the char is or-ed with 0x90

```
                0x00400878      55              push rbp
                0x00400879      4889e5          mov rbp, rsp
                0x0040087c      89f8            mov eax, edi
                0x0040087e      8845fc          mov byte [local_4h], al
                0x00400881      0fbe45fc        movsx eax, byte [local_4h]
                0x00400885      890511082000    mov dword [0x0060109c], eax    ; [0
            ,=< 0x0040088b      eb02            jmp loc.0040088f            ;[1]
            |   0x0040088d      c9              leave
            |   0x0040088e  ~   35528b1425      xor eax, 0x25148b52
loc.0040088f 10
 loc.0040088f ();
            |   ; JMP XREF from 0x0040088b (step2)
            `-> 0x0040088f      52              push rdx
                0x00400890      8b14259c1060.   mov edx, dword [0x60109c]    ; [0x6
            ,=< 0x00400897      eb02            jmp loc.0040089b            ;[2]
            |   0x00400899  ~   085a81          or byte [rdx - 0x7f], bl
loc.0040089b 22
 loc.0040089b ();
            |   ; JMP XREF from 0x00400897 (loc.0040088f)
            `-> 0x0040089b      81ca90000000    or edx, 0x90
                0x004008a1      8914259c1060.   mov dword [0x60109c], edx    ; [0x6
                0x004008a8      5a              pop rdx
                0x004008a9      8b05ed072000    mov eax, dword [0x0060109c]    ; [0
                0x004008af      5d              pop rbp
                0x004008b0      c3              ret
```

BY HITTING 'Q' TWO TIMES, LET'S GO BACK TO MAIN

```
0x004009b4      488d45e0        lea rax, [local_20h]
0x004009b8      4889c7          mov rdi, rax
0x004009bb      e8f1feffff      call 0x4008b1
0x004009c0      488d45e0        lea rax, [local_20h]
0x004009c4      be70106000      mov esi, 0x601070
0x004009c9      4889c7          mov rdi, rax
0x004009cc      e832ffffff      call 0x400903
0x004009d1      85c0            test eax, eax
                                jne 0x4009f5                    ;[5]
0x004009d5      488b05a40620.   mov rax, qword obj.stdout       ; [0x601080:8]=0x7f1dcbbf5620 ;
0x004009dc      4889c1          mov rcx, rax
0x004009df      ba2c000000      mov edx, 0x2c                   ; ',' ; 44
```

do something with mypwd ;[3]

do something with result of 3 and something stored @0x601070 ;[4]

based on result of [4]

ko

ok!

[4] must be the comparison of the given pwd or-ed with 0x90 and the real one stored (or-ed) @0x601070

```
:> px @0x601070
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00601070  f7f8 f1f4 f1f8 b3fc fc00 0000 0000 0000  ................
0x00601080  2056 bfcb 1d7f 0000 0000 0000 0000 0000   V..............
0x00601090  e048 bfcb 1d7f 0000 0000 0000 0000 0000  .H..............
```

```
...
026 0x00000ac0 0x00400ac0  35  36 (LOAD0) ascii I'm sorry GDB! You are not allowed!
027 0x00000ae8 0x00400ae8  30  31 (LOAD0) ascii Tracing is not allowed... Bye!
028 0x00000b07 0x00400b07  28  29 (LOAD0) ascii Please tell me my password:
029 0x00000b28 0x00400b28  44  45 (LOAD0) ascii The password is correct!\nCongratulations!!!\n
030 0x00000b55 0x00400b55  27  28 (LOAD0) ascii No! No! No! No! Try again.\n
...
```

- **f7 f8 f1 f4 f1 f8 b3 fc fc** must be the password or-ed with 0x90 char by char
- To retrieve the password we have to xor with 0x90 every byte…

```
$ python3 -c "for i in [0xf7,0xf8,0xf1,0xf4,0xf1,0xf8,0xb3,0xfc,0xfc]:
print(chr(i^0x90),end='')"
ghadah#ll
```

- Let's try…

```
$ ./crackme.02
Please tell me my password: ghadah#ll
The password is correct!
Congratulations!!!
```

BEFORE LEAVING…LET'S HAVE A LOOK TO THE SOURCES

- crackme.02.x64.c
- Cleaner64.c

MORE FUN?

- http://csapp.cs.cmu.edu/3e/bomb.tar
- Writeup: http://csapp.cs.cmu.edu/2e/datalab.pdf
  - This version of the bomb is disconnected by the grading server…

- Play hard…working with **stripped** binaries:
  - https://medium.com/@faisal48/working-with-stripped-binaries-in-gdb-cacacd7d5a33
  - https://ocw.cs.pub.ro/courses/cns/labs/lab-03

REFERENCES

- Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, chap 15
  - http://staff.ustc.edu.cn/~bjhua/courses/security/2014/readings/anti-disas.pdf
- https://reverseengineering.stackexchange.com/questions/1531/what-is-overlapping-instructions-obfuscation
- Code Encryption, http://phrack.org/issues/58/5.html
- Beginners Guide to Basic Linux Anti Anti Debugging Techniques, http://www.stonedcoder.org/~kd/lib/14-61-1-PB.pdf
- Anti Debugging Protection Techniques With Examples (MS-Win) https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software
- Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software
- Playing with PTRACE parts 1 and 2, https://www.linuxjournal.com/article/6100
- Thread Local Storage, http://www.nynaeve.net/?tag=tls
- https://www.apriorit.com/dev-blog/367-anti-reverse-engineering-protection-techniques-to-use-before-releasing-software
- https://github.com/geyslan/crackmes
  - https://sgros-students.blogspot.com/2015/08/reverse-engineering-with-radare2.html