SWRhapsody

Sh4ll8 crackme

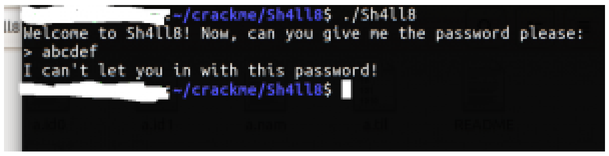
于10月 9, 2018由**SWRhapsody**发布

Introduction

This is a write up for a simple crackme, you can download this crackme from here: https://crackmes.one/crackme/5b506ba633c5d41c0b8ae522.

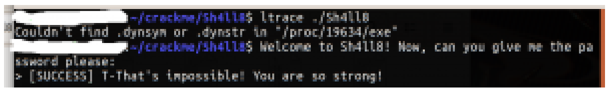
Code

Open the binary in IDA does not give anything, it's obviously packed. We have two options here, unpack it or run it and dump the memory, and in this post I just use the second method. But before we doing this let's see how program runs.

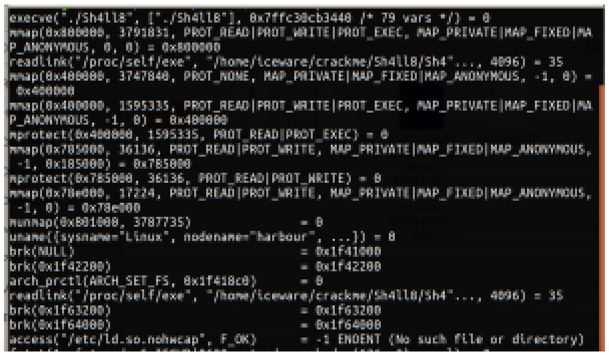


```
[redacted]~/crackme/Sh4ll8$ ./Sh4ll8
Welcome to Sh4ll8! Now, can you give me the password please:
> abcdef
I can't let you in with this password!
[redacted]~/crackme/Sh4ll8$
```

It is a very simple crackme, input a serial and check whether is correct. Before we dump it let's analyze it a bit more.



```
[redacted]~/crackme/Sh4ll8$ ltrace ./Sh4ll8
Couldn't find .dynsym or .dynstr in "/proc/19634/exe"
[redacted]~/crackme/Sh4ll8$ Welcome to Sh4ll8! Now, can you give me the pa
ssword please:
> [SUCCESS] T-That's impossible! You are so strong!
```



```
execve("./Sh4ll8", ["./Sh4ll8"], 0x7ffc30cb3440 /* 79 vars */) = 0
mmap(0x800000, 3791831, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MA
P_ANONYMOUS, 0, 0) = 0x800000
readlink("/proc/self/exe", "/home/iceware/crackme/Sh4ll8/Sh4"..., 4096) = 35
mmap(0x400000, 3747840, PROT_NONE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
 0x400000
mmap(0x400000, 1595335, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MA
P_ANONYMOUS, -1, 0) = 0x400000
mprotect(0x400000, 1595335, PROT_READ|PROT_EXEC) = 0
mmap(0x785000, 36136, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
 -1, 0x185000) = 0x785000
mprotect(0x785000, 36136, PROT_READ|PROT_WRITE) = 0
mmap(0x78e000, 17224, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
 -1, 0) = 0x78e000
munmap(0x801000, 3787735)               = 0
uname({sysname="Linux", nodename="harbour", ...}) = 0
brk(NULL)                               = 0x1f41000
brk(0x1f42200)                          = 0x1f42200
arch_prctl(ARCH_SET_FS, 0x1f418c0)      = 0
readlink("/proc/self/exe", "/home/iceware/crackme/Sh4ll8/Sh4"..., 4096) = 35
brk(0x1f63200)                          = 0x1f63200
brk(0x1f64000)                          = 0x1f64000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
```

SWRhapsody

So the crackme solved itself when we use `ltrace`? It seems that this crackme check one char at a time and only tell you wrong when you enter something wrong, for example if serial is abcdef and you enter abcd it will say you win. But I don't really check if this is true in code.

And program complains when we use `strace` in the second picture, we also find this crackme use `ptrace` which is a basic anti debug technique.

As we gether some basic information about this crackme, we can start solve it. I will unpack it, use Snowman to decompile it in IDA, find the algorithm about serial and the crack it.

First let `ptrace` return 1 to bypass the anti-debug. Run crackme in gdb.

```
catch syscall ptrace
```

```
gef➤  r
Starting program: ~/crackme/Sh4ll8/Sh4ll8
Welcome to Sh4ll8! Now, can you give me the password please:
> abcd
[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────[ registers ]────
$rax   : 0xffffffffffffffda
$rbx   : 0x0000000000400310  →  0x00c0c74808ec8348
$rcx   : 0x00000000004f060e  →  0x2a77fffff0003d48 ("H="?)
$rdx   : 0x0000000000000001
$rsp   : 0x00007fffffffc7e8  →  0x0000000000400ee0  →  0x84c0940ffff88348
$rbp   : 0x00007fffffffc7f0  →  0x00007fffffffc950  →  0x000000000078c018  →  0x00000000004d18
$rsi   : 0x0000000000000000
$rdi   : 0x0000000000000000
$rip   : 0x00000000004f060e  →  0x2a77fffff0003d48 ("H="?)
$r8    : 0x00000000ffffffff
$r9    : 0x0000000000000000
$r10   : 0x0000000000000000
$r11   : 0x0000000000000282
$r12   : 0x0000000000499200  →  0x00785980be415641
$r13   : 0x0000000000499290  →  0x8148007859d8bb53
$r14   : 0x0000000000000000
$r15   : 0x00007fffffffce78  →  0x000000000040000c  →  0x003e000290c3050f
$eflags: [zero carry parity adjust SIGN trap INTERRUPT direction overflow resume virtualx86 id
$ds: 0x0000  $gs: 0x0000  $ss: 0x002b  $cs: 0x0033  $fs: 0x0000  $es: 0x0000
────────────────────────────────────────────────────────────────────[ stack ]────
0x00007fffffffc7e8│+0x00: 0x0000000000400ee0  →  0x84c0940ffff88348  ← $rsp
0x00007fffffffc7f0│+0x08: 0x00007fffffffc950  →  0x000000000078c018  →  0x00000000004d18c0  →
0x00007fffffffc7f8│+0x10: 0x0000000000401459  →  0xc084c0940ffff883
0x00007fffffffc800│+0x18: 0x0000000000788720  →  0x0000000000000008
0x00007fffffffc808│+0x20: 0x000000000045ad09  →  0x48595a1024448b48
0x00007fffffffc810│+0x28: 0x000000000078fa60  →  0x0000000000787080  →  0x0000000000407730  →
0x00007fffffffc818│+0x30: 0x00007fffffffc820  →  0x00000000007a8700  →  0x000002f7000002fe
0x00007fffffffc820│+0x38: 0x00000000007a8700  →  0x000002f7000002fe
──────────────────────────────────────────────────────────[ code:i386:x86-64 ]────
     0x4f0603                  cmova  r10, rcx
```

SWRhapsody

```
     0x4f0619                       js     0x4f0634
     0x4f061b                       cmp    r8d, 0x2
     0x4f061f                       ja     0x4f0634
──────────────────────────────────────────────────────────────────────────────[ threads ]────
[#0] Id 1, Name: "Sh4ll8", stopped, reason: BREAKPOINT
────────────────────────────────────────────────────────────────────────────────[ trace ]────
[#0] 0x4f060e → cmp rax, 0xfffffffffffff000
[#1] 0x400ee0 → cmp rax, 0xffffffffffffffff
[#2] 0x7fffffffc950 → sbb al, al
[#3] 0x401459 → cmp eax, 0xffffffff
[#4] 0x788720 → or BYTE PTR [rax], al
[#5] 0x45ad09 → mov rax, QWORD PTR [rsp+0x10]
[#6] 0x78fa60 → xor BYTE PTR [rax+0x78], 0x0
[#7] 0x7fffffffc820 → add BYTE PTR [rdi+0x7a], al
[#8] 0x7a8700 → inc BYTE PTR [rdx]
[#9] 0x7a8754 → add BYTE PTR [rax], al
─────────────────────────────────────────────────────────────────────────────────────────────


Catchpoint 1 (call to syscall ptrace), 0x00000000004f060e in ?? ()
gef➤
```

`ptrace` is called after we enter serial

```
ni
set $rax=1
ni
```

anti-debug is gone. We can dump the binary here.

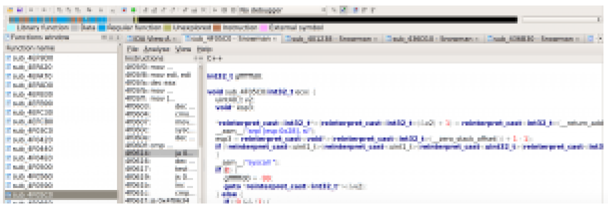
```
gef➤info proc mappings
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
            0x400000           0x586000   0x186000        0x0
            0x586000           0x785000   0x1ff000        0x0
            0x785000           0x7b6000    0x31000        0x0 [heap]
            0x800000           0x801000     0x1000        0x0
      0x7ffff7ffa000     0x7ffff7ffd000     0x3000        0x0 [vvar]
      0x7ffff7ffd000     0x7ffff7fff000     0x2000        0x0 [vdso]
      0x7ffffffdd000     0x7ffffffff000    0x22000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

Our current assembly code is in the first section, dump it

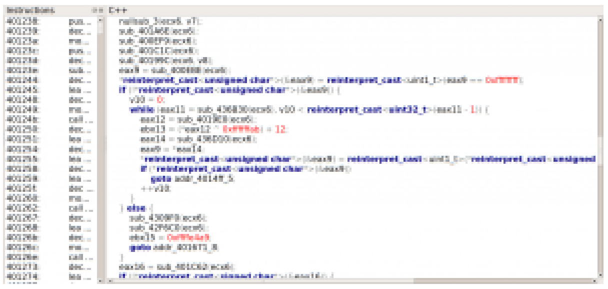
```
dump binary memory a 0x400000 0x586000
```

Use IDA to open the file and it will complain what you select do not have valid header, just ignore it.

I use Snowman to decompile the code, so the function will be different from IDA commercial decompile result.



Continue run the code and after several return, we reach here



```
Instructions        C++
401238:  pus...     [illegible]
401239:  dec...     sub_401468(ecx6);
40123a:  mo...      sub_4085F9(ecx6);
40123c:  pus...     sub_403C1C(ecx6);
40123d:  dec...     sub_40399C(ecx6, v8);
40123e:  sub...     eax9 = sub_408888(ecx6);
401244:  dec...     *reinterpret_cast<unsigned char*>(&eax9) = reinterpret_cast<uint1_t>(eax9 == 0xffffff);
401245:  lea...     if (!*reinterpret_cast<unsigned char*>(&eax9)) {
401248:  dec...         v30 = 0;
401249:  mo...          while (eax11 = sub_436830(ecx6), v30 < reinterpret_cast<uint32_t>(eax11 - 1)) {
40124b:  call...            eax12 = sub_4019E0(ecx6);
401250:  dec...             ebx13 = (*eax12 ^ 0xffff5ab) + 12;
401251:  lea...             eax14 = sub_436D00(ecx6);
401254:  dec...             eax9 = *eax14;
401255:  lea...             *reinterpret_cast<unsigned char*>(&eax9) = reinterpret_cast<uint1_t>(*reinterpret_cast<unsigned
401258:  dec...             if (*reinterpret_cast<unsigned char*>(&eax9))
401259:  lea...                 goto addr_4014ff_5;
40125f:  dec...             ++v30;
401260:  mo...          }
401262:  call...    } else {
401267:  dec...         sub_4308F9(ecx6);
401268:  lea...         sub_42F6C0(ecx6);
40126b:  dec...         ebx15 = 0xffffa4a9;
40126c:  mo...          goto addr_4016T1_8;
40126e:  call...    }
401273:  dec...     eax16 = sub_401C82(ecx6);
401274:  lea...     if (*reinterpret_cast<signed char*>(&eax16)) {
```

and if you playing around the `while` loop you will find out that eax14 is point to the serial we enter. As eax14 is compared with eax13, this should be the correct serial we are looking for, and you can see what eax12 is in gdb

```
[ Legend: Modified register | Code | Heap | Stack | String ]
────────────────────────────────────────────────────────────────────────────────────────────────────
$rax   : 0x00007fffffffc820  →  0x00000000007a8700  →  0x000002f7000002fe
$rbx   : 0x0000000000000000
$rcx   : 0x00000000004f060e  →  0x2a77fffff0003d48 ("H="?)
$rdx   : 0x0000000000000000
$rsp   : 0x00007fffffffc800  →  0x0000000000788720  →  0x0000000000000008
$rbp   : 0x00007fffffffc950  →  0x000000000078c018  →  0x00000000004d18c0  →  0x4ffa8348c0ef0f
$rsi   : 0x0000000000000000
$rdi   : 0x00007fffffffc820  →  0x00000000007a8700  →  0x000002f7000002fe
$rip   : 0x00000000004014cf  →  0x83008b0000050ce8
$r8    : 0x00000000ffffffff
$r9    : 0x0000000000000000
$r10   : 0x0000000000000000
$r11   : 0x0000000000000282
$r12   : 0x0000000000499200  →  0x00785980be415641
$r13   : 0x0000000000499290  →  0x8148007859d8bb53
$r14   : 0x0000000000000000
$r15   : 0x00007fffffffce78  →  0x000000000040000c  →  0x003e000290c3050f
$eflags: [zero carry parity adjust sign trap INTERRUPT direction overflow resume virtualx86 id
$gs: 0x0000  $cs: 0x0033  $ds: 0x0000  $es: 0x0000  $fs: 0x0000  $ss: 0x002b
────────────────────────────────────────────────────────────────────────────────────────────────────
0x00007fffffffc800│+0x00: 0x0000000000788720  →  0x0000000000000008  ← $rsp
0x00007fffffffc808│+0x08: 0x000000000045ad09  →  0x48595a1024448b48
0x00007fffffffc810│+0x10: 0x000000000078fa60  →  0x0000000000787080  →  0x0000000000407730  →
0x00007fffffffc818│+0x18: 0x00007fffffffc820  →  0x00000000007a8700  →  0x000002f7000002fe
0x00007fffffffc820│+0x20: 0x00000000007a8700  →  0x000002f7000002fe  ← $rax, $rdi
0x00007fffffffc828│+0x28: 0x00000000007a8754  →  0x0000000000000000
0x00007fffffffc830│+0x30: 0x00000000007a8780  →  0x0000000000000000
0x00007fffffffc838│+0x38: 0x000000000078f4a0  →  0x00000000007875e8  →  0x00000000004076e0  →
────────────────────────────────────────────────────────────────────────────────────────────────────
     0x4014c2                  lea    rax, [rbp-0x130]
     0x4014c9                  mov    rsi, rdx
     0x4014cc                  mov    rdi, rax
 →   0x4014cf                  call   0x4019e0
   ↳    0x4019e0                  push   rbp
        0x4019e1                  mov    rbp, rsp
        0x4019e4                  mov    QWORD PTR [rbp-0x8], rdi
        0x4019e8                  mov    QWORD PTR [rbp-0x10], rsi
```

```
    $rsi = 0x0000000000000000,
    $rdx = 0x0000000000000000
)
──────────────────────────────────────────────────────────────────────────────
[#0] Id 1, Name: "Sh4ll8", stopped, reason: BREAKPOINT
──────────────────────────────────────────────────────────────────────────────
[#0] 0x4014cf → call 0x4019e0
[#1] 0x788720 → or BYTE PTR [rax], al
[#2] 0x45ad09 → mov rax, QWORD PTR [rsp+0x10]
[#3] 0x78fa60 → xor BYTE PTR [rax+0x78], 0x0
[#4] 0x7ffffffc820 → add BYTE PTR [rdi+0x7a], al
[#5] 0x7a8700 → inc BYTE PTR [rdx]
[#6] 0x7a8754 → add BYTE PTR [rax], al
[#7] 0x7a8780 → add BYTE PTR [rax], al
[#8] 0x78f4a0 → call 0x796d1a
[#9] 0x7a8160 → outs dx, BYTE PTR ds:[rsi]
──────────────────────────────────────────────────────────────────────────────

Breakpoint 2, 0x00000000004014cf in ?? ()
gef➤  x/30xw 7a8700
Invalid number "7a8700".
gef➤  x/30xw 0x7a8700
0x7a8700:   0x000002fe  0x000002f7  0x000002fe  0x000002f7
0x7a8710:   0x000002f8  0x000002c9  0x000002c8  0x000002fd
0x7a8720:   0x000002c8  0x000002f3  0x000002c6  0x000002fc
0x7a8730:   0x000002fe  0x000002c9  0x000002fc  0x000002cd
0x7a8740:   0x000002fe  0x000002fc  0x000002f4  0x000002ca
0x7a8750:   0x000002f2  0x00000000  0x00000000  0x00000000
0x7a8760:   0x00000000  0x00000000  0x00000000  0x00000000
0x7a8770:   0x00000000  0x00000000
```

Now we only need to decrpty it. Don't forget these are int vars.

```python
ta=[0x000002fe,0x000002f7,0x000002fe,0x000002f7,0x000002f8,0x000002c9,
0x000002c8, 0x000002fd, 0x000002c8, 0x000002f3, 0x000002c6, 0x000002fc,
 0x000002fe, 0x000002c9, 0x000002fc, 0x000002cd,
0x000002fe, 0x000002fc, 0x000002f4, 0x000002ca, 0x000002f2]

for i in range(0,len(ta)):
    print(chr(((ta[i]^0xffffffab)+0xc)&0xff),end=' ')
```

```
a h a h _ n o b o d y c a n c r a c k m e
```

分类: EXERCISE

0 条评论

SWRhapsody

发表评论

名称 *

电子邮件 *

网站

在想些什么?

发表评论

近期文章

携程Apollo YAML 反序列化

CVE-2020-5410

CodeQL部分源码简读

服务器与网关的不一致

CodeQL 部分使用记录

近期评论

SWRhapsody

分类目录

SWRhapsody

HackTheBox

Penetration Test

Uncategorized

相关文章

EXERCISE

CodeQL部分源码简读

Introduction CodeQL 也用了不少时间了，从最初的不会 阅读更多…

CHEAT SHEET

CodeQL 部分使用记录

前言 CodeQL 是一个代码分析引擎，主要原理是通过对代码进行构建并 阅读更多…

EXERCISE

Java 反编译工具

在复现 CVE-2019-15012 遇到了一个非常坑的地方，使用 J 阅读更多…

SWRhapsody

ABOUT