登录

Radare2使用实战

hackedbylh / 2017-11-23 10:01:00 / 浏览数 3061 技术文章 技术文章 顶(0) 踩(0)

前言

前文讲了一些 radare2 的特性相关的操作方法。本文以一个 crackme 来具体介绍下 radare2 的使用

程序的地址： 在这里

正文

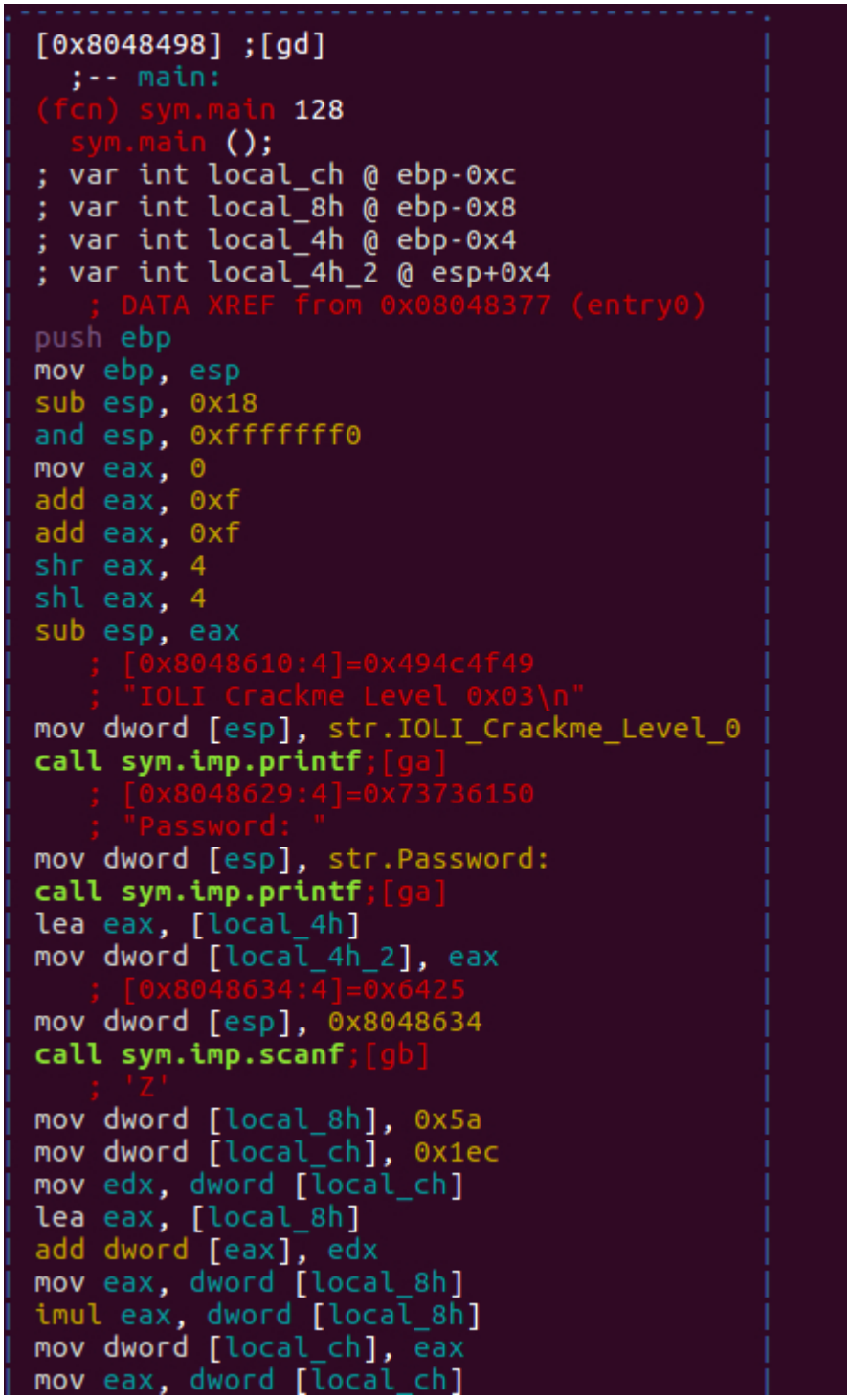
首先使用 radare2 加载该程序。使用了 aaa 分析了程序中的所有函数。使用 iI 查看二进制文件的信息。可以看到是 32 位的。

```
haclh@ubuntu:~/r2_learn$ r2 bin-linux/crackme0x03
 -- Execute commands on a temporary offset by appending '@ offset' to your command.
[0x08048360]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze len bytes of instructions for references (aar)
[x] Analyze function calls (aac)
[x] Use -AA or aaaa to perform additional experimental analysis.
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[0x08048360]> iI
arch     x86
binsz    7580
bintype  elf
bits     32
canary   false
class    ELF32
crypto   false
endian   little
havecode true
intrp    /lib/ld-linux.so.2
lang     c
linenum  true
lsyms    true
machine  Intel 80386
maxopsz  16
minopsz  1
nx       true
os       linux
pcalign  0
pic      false
relocs   true
relro    partial
rpath    NONE
```

使用 aaa分析完程序后，可以使用 afl 查看所有的函数。

```
[0x08048360]> afl
0x080482f8    1 23           sym._init
0x08048320    1 6            sym.imp.__libc_start_main
0x08048330    1 6            sym.imp.scanf
0x08048340    1 6            sym.imp.strlen
0x08048350    1 6            sym.imp.printf
0x08048360    1 33           entry0
0x08048384    3 33           fcn.08048384
0x080483b0    6 47           sym.__do_global_dtors_aux
0x080483e0    4 50           sym.frame_dummy
0x08048414    4 90           sym.shift
0x0804846e    4 42           sym.test
0x08048498    1 128          sym.main
0x08048520    4 99           sym.__libc_csu_init
0x08048590    1 5            sym.__libc_csu_fini
0x08048595    1 4            sym.__i686.get_pc_thunk.bx
0x080485a0    4 35           sym.__do_global_ctors_aux
0x080485c4    1 26           sym._fini
[0x08048360]>
```

直接跳到 main 函数看看逻辑



不习惯看文本模式的汇编的话，可以使用 vV 进入图形化模式

```
[0x8048498] ;[gd]
  ;-- main:
(fcn) sym.main 128
  sym.main ();
; var int local_ch @ ebp-0xc
; var int local_8h @ ebp-0x8
; var int local_4h @ ebp-0x4
; var int local_4h_2 @ esp+0x4
    ; DATA XREF from 0x08048377 (entry0)
push ebp
mov ebp, esp
sub esp, 0x18
and esp, 0xfffffff0
mov eax, 0
add eax, 0xf
add eax, 0xf
shr eax, 4
shl eax, 4
sub esp, eax
    ; [0x8048610:4]=0x494c4f49
    ; "IOLI Crackme Level 0x03\n"
mov dword [esp], str.IOLI_Crackme_Level_0
call sym.imp.printf;[ga]
    ; [0x8048629:4]=0x73736150
    ; "Password: "
mov dword [esp], str.Password:
call sym.imp.printf;[ga]
lea eax, [local_4h]
mov dword [local_4h_2], eax
    ; [0x8048634:4]=0x6425
mov dword [esp], 0x8048634
call sym.imp.scanf;[gb]
    ; 'Z'
mov dword [local_8h], 0x5a
mov dword [local_ch], 0x1ec
mov edx, dword [local_ch]
lea eax, [local_8h]
add dword [eax], edx
mov eax, dword [local_8h]
imul eax, dword [local_8h]
mov dword [local_ch], eax
mov eax, dword [local_ch]
```

拿到个程序，我会首先看函数调用理解程序的大概流程。比如这里先调用了 `printf` 打印了一些提示信息，然后使用 `scanf` 获取我们的输入，分析`scanf`的参数

```
|           0x080484cc      8d45fc         lea eax, [local_4h]
|           0x080484cf      89442404       mov dword [local_4h_2], eax
|           0x080484d3      c70424348604.  mov dword [esp], 0x8048634  ; [0x8048634:4]=0x6425
|           0x080484da      e851feffff     call sym.imp.scanf          ; int scanf(const char *format)
```

我们可以知道`0x8048634` 是我们的第一个参数, `local_4h`是我们的第二个参数。看看 `0x8048634`存放的是什么。

```
[0x08048498]> ps @ 0x8048634
%d
[0x08048498]>
```

所以程序需要我们输入的是一个 整数，然后把它存在 `local_4h`里面了。那我们就可以把 `local_4h` 变量改下名字。这里改成 `input`

```
[0x08048498]> afvn local_4h input
[0x08048498]> pdf
           ;-- main:
/ (fcn) sym.main 128
|   sym.main ();
|           ; var int local_ch @ ebp-0xc
|           ; var int local_8h @ ebp-0x8
|           ; var int input @ ebp-0x4
|           ; var int local_4h_2 @ esp+0x4
|              ; DATA XREF from 0x08048377 (entry0)
|           0x08048498      55              push ebp
|           0x08048499      89e5            mov ebp, esp
|           0x0804849b      83ec18          sub esp, 0x18
|           0x0804849e      83e4f0          and esp, 0xfffffff0
[0x080484cc 16% 170 (0x7:-1=1)]> pd $r @ sym.main+59
|           0x080484cc      8d45fc          lea eax, [input]
|           0x080484cf      89442404        mov dword [local_4h_2], eax
|           0x080484d3   *  c70424348604.  mov dword [esp], 0x8048634  ; [0x8048634:4]=0x6425
|           0x080484da      e851feffff      call sym.imp.scanf          ;[1] ; int scanf(const char *forma
|           0x080484df      c745f85a0000.   mov dword [local_8h], 0x5a  ; 'Z'
|           0x080484e6      c745f4ec0100.   mov dword [local_ch], 0x1ec
|           0x080484ed      8b55f4          mov edx, dword [local_ch]
|           0x080484f0      8d45f8          lea eax, [local_8h]
|           0x080484f3      0110            add dword [eax], edx
|           0x080484f5      8b45f8          mov eax, dword [local_8h]
|           0x080484f8      0faf45f8        imul eax, dword [local_8h]
|           0x080484fc      8945f4          mov dword [local_ch], eax
```

继续往下看发现 `input` 变量后来没有被处理直接传到了 `test` 函数。他的第二个参数是这样生成的

```
      0x080484da      e851feffff      call sym.imp.scanf          ; int scanf(const char *format)
      0x080484df      c745f85a0000.   mov dword [local_8h], 0x5a  ; 'Z'
      0x080484e6      c745f4ec0100.   mov dword [local_ch], 0x1ec
      0x080484ed      8b55f4          mov edx, dword [local_ch]
      0x080484f0      8d45f8          lea eax, [local_8h]
      0x080484f3      0110            add dword [eax], edx
      0x080484f5      8b45f8          mov eax, dword [local_8h]
      0x080484f8      0faf45f8        imul eax, dword [local_8h]
      0x080484fc      8945f4          mov dword [local_ch], eax
      0x080484ff      8b45f4          mov eax, dword [local_ch]
      0x08048502      89442404        mov dword [local_4h_2], eax
      0x08048506      8b45fc          mov eax, dword [input]
      0x08048509      890424          mov dword [esp], eax
      0x0804850c      e85dffffff      call sym.test
      0x08048511      b800000000      mov eax, 0
      0x08048516      c9              leave
      0x08048517      c3              ret
484cc]>
```


为了获得这个参数我们有很多方法，比如 我们可以直接静态分析，或者用 `gdb` 调试这都很容易得到结果。

这里正好试试 `radare`
的模拟执行功能。使用该功能我们需要先分析要模拟执行的代码对环境的依赖，比如寄存器的值，内存的值等，然后根据依赖关系修改内存和寄存器的值来满足代码运行的上

在这里这段代码只对栈的内存进行了处理。那我们就先分配一块内存，然后用 `esp`
刚刚分配的内存。由于这里一开始没有对内存数据进行读取，所以我们直接使用分配的内存就好，不用对他进行处理。

首先我们跳到目标地址，然后使用 `aei` 或者 `aeip` 初始化虚拟机堆栈，然后使用 `aer` 查看寄存器状态。

```
[0x080484df]> s 0x080484df
[0x080484df]> aeip
x
[0x080484df]> aei
[0x080484df]> aer
oeax = 0x00000000
eax = 0x00000000
ebx = 0x00000000
ecx = 0x00000000
edx = 0x00000000
esi = 0x00000000
edi = 0x00000000
esp = 0x00000000
ebp = 0x00000000
eip = 0x080484df
eflags = 0x00000000
[0x080484df]>
```

然后分配一块内存作为栈内存，给程序模拟执行用。

```
[0x080484df]> aeim 0xff0000 0x40000 stack
[0x080484df]> aeim?
Usage: aeim [addr] [size] [name] - initialize ESIL VM stack
Default: 0x100000 0xf0000
See ae? for more help
[0x080484df]> px 0x10 @ 0xff0000
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00ff0000  0000 0000 0000 0000 0000 0000 0000 0000  ................
[0x080484df]>
```

在 `0xff0000` 分配了 `0x40000` 大小的内存。然后把 `esp` 和 `ebp` 指到这块内存里面。

```
[0x080484df]> aer esp=0xff2000
[0x080484df]> aer ebp=0xff2000-0x40
x
[0x080484df]> aer
oeax = 0x00000000
eax = 0x00000000
ebx = 0x00000000
ecx = 0x00000000
edx = 0x00000000
esi = 0x00000000
edi = 0x00000000
esp = 0x00ff2000
ebp = 0x00ff1fc0
eip = 0x080484df
eflags = 0x00000000
[0x080484df]> px 0x10 @ esp
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00ff2000  0000 0000 0000 0000 0000 0000 0000 0000  ................
[0x080484df]>
```

然后我们让模拟器运行到 `0x0804850c` 也就是调用 `test` 函数的位置处，查看他的参数，可以看到第二个参数的值就是 `0x00052b24`

```
[0x080484df]> aesu 0x0804850c
[0x080484df]> aer
oeax = 0x00000000
eax = 0x00000000
ebx = 0x00000000
ecx = 0x00000000
edx = 0x000001ec
esi = 0x00000000
edi = 0x00000000
esp = 0x00ff2000
ebp = 0x00ff1fc0
eip = 0x0804850c
eflags = 0x00000000
[0x080484df]> pxw 0x10 @ esp
0x00ff2000  0x00000000 0x00052b24 0x00000000 0x00000000  ....$+..........
[0x080484df]>
```

最后我们进去 test 函数里面看看

```
[0x804846e] ;[gb]
(fcn) sym.test 42
  sym.test (int arg_8h, int arg_ch);
; arg int arg_8h @ ebp+0x8
; arg int arg_ch @ ebp+0xc
    ; CALL XREF from 0x0804850c (sym.main)
push ebp
mov ebp, esp
sub esp, 8
    ; [0x8:4]=-1
    ; 8
mov eax, dword [arg_8h]
    ; [0xc:4]=-1
    ; 12
cmp eax, dword [arg_ch]
je 0x804848a;[ga]
```

```
f t
```

```
0x804847c ;[ge]
    ; [0x80485ec:4]=0x6479714c
    ; "Lqydolg#Sdvvzrug$"
mov dword [esp], str.Lqydolg_Sdvvzrug_
call sym.shift;[gc]
jmp 0x8048496;[gd]
```

```
0x804848a ;[ga]
    ; JMP XREF from 0x0804847a (sym.test)
    ; [0x80485fe:4]=0x76766453
    ; "Sdvvzrug#RN$$$#=,"
mov dword [esp], str.Sdvvzrug_RN_____
call sym.shift;[gc]
```

就是判断 ■■■ 和 ■■■ 是否一致，所以这个 crackme 的 key 就是 0x00052b24 十进制数表示 338724.

```
[0x0804846e]> ? 0x00052b24
338724 0x52b24 01225444 330.8K 5000:0b24 338724 "$+\x05" 0000010100101011001001000 338724.0 338724.00
338724.000000
[0x0804846e]> q
haclh@ubuntu:~/r2_learn$ ./bin-linux/crackme0x03
IOLI Crackme Level 0x03
Password: 338724
Password OK!!! :)
haclh@ubuntu:~/r2_learn$
```

成功

总结

radare2 的模拟执行功能是通过 esil 来实现的，粗略的试了一下感觉还是挺不错的感觉和 unicorn 有的一拼，不过radare2也是有 unicorn的插件的。

参考：

http://radare.org/r/talks.html

https://github.com/radare/radare2book

https://codeload.github.com/radareorg/r2con/

点击收藏 | 0 关注 | 1

上一篇：LCTF 2017 三道Web题的... 下一篇：使用TensorFlow自动识别验...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录