老司机带你玩转Radare2

hetianlab (/u/708e8568411c) ＋关注
2018.02.28 17:47 字数 1461 阅读 1501 评论 1 喜欢 2
(/u/708e8568411c)

Radare2实战（图片来源：hetianlab.com）

说起逆向，你想到的可能是IDA Pro，OllyDBG。

而Radare2是一款开放源代码的逆向工程平台，它的强大超越你的想象，包括反汇编、分析数据、打补丁、比较数据、搜索、替换、虚拟化等等，同时具备超强的脚本加载能力，并且可以运行在几乎所有主流的平台（GNU/Linux, .Windows *BSD, iOS, OSX, Solaris…）上。可谓是一大神器。

这里我们使用Kali系统来学习它，因为Kali系统自带了这个神器。

打开终端，使用radare2 -h 可以查看其帮助信息

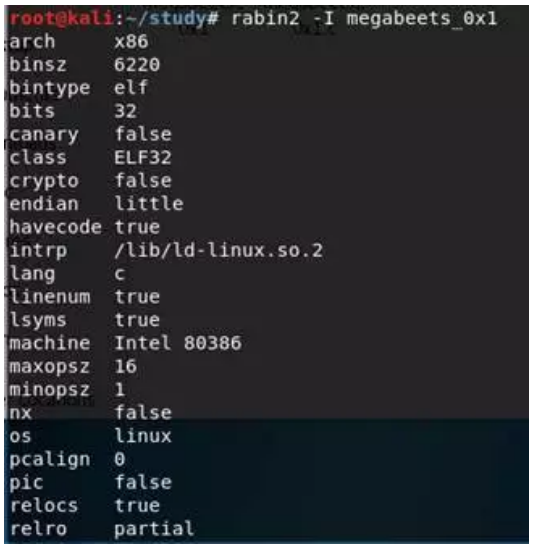
radare2帮助（图片来源：hetianlab.com）

radare2里面有个很牛逼的工具：rabin2

rabin2 可以获取包括ELF, PE, Mach-O, Java CLASS文件的区段、头信息、导入导出表、字符串相关、入口点等等，并且支持几种格式的输出文件。使用man rabin2 可以查看rabin2的使用帮助文档。

然后我们通过破解一个crackme来学习神器radare2的使用。

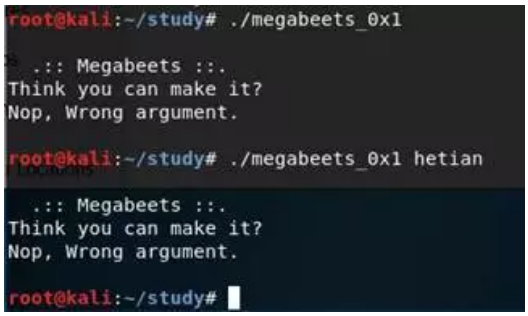
首先使用 rabin2 打印出二进制文件的系统属性、语言、字节序、框架、以及使用了哪些加固技术



rabin2 -I（图片来源：hetianlab.com）

可以看到这是一个32位的 elf 文件，没有剥离符号表并且是动态链接的

接下来尝试运行它



运行（图片来源：hetianlab.com）

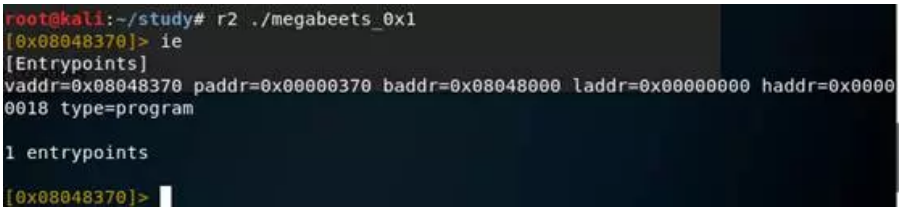
可以看到，不论是否加参数都会显示wrong。

接下来使用radare2来进行破解

（图片来源：hetianlab.com）

可以黄色输出了一个地址 (0x08048370)，这就是它自动识别的程序入口点

或者也可以使用ie命令手动打印出入口点

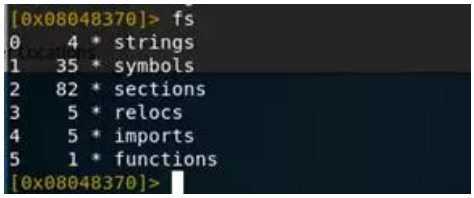


（图片来源：hetianlab.com）

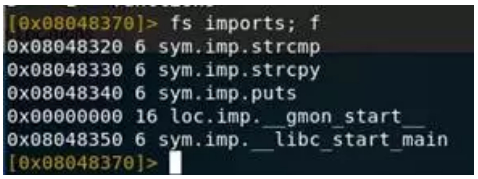
接下来输入aa或者aaa进行细致的分析

分析完成之后， r2会将所有有用的信息和特定的名字绑定在一起，比如区段、函数、符号、字符串，这些都被称作 'flags', flags 被整合进 ，一个 flag 是所有类似特征的集合

接下来看看所有的flag



我们打印出imports下面的信息



（图片来源：hetianlab.com）

为了获取更多的信息，我们可以再列出数据段里的字符串

（图片来源：hetianlab.com）

出现了关键字，一个是success，一个是我们之前运行时的wrong….

那我们接下来就跟着success走，看看哪儿进行了调用

输入命令axt @@ str.*

```
[0x08048370]> axt @@ str.*
[0x08048370]>
```

（图片来源：hetianlab.com）

'axt' 命令用来在 data/code段里找寻某个地址相关的引用（更多的操作，请看 'ax?'）.

'@@'就像一个迭代器，用来在地址空间里不断地匹配后面一系列相关的命令（更多操作，请看 '@@?'）

'str.*' 是一个通配符，用来标记所有以 'str.'开头的信息，不光会列出字符串标志，同时也包括函数名，找到它们到底在哪里以及何处被调用。

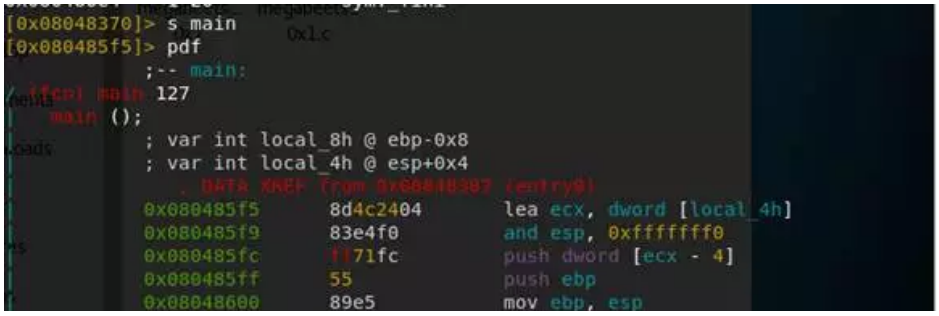
接下来我们看看radare2分析出来哪些函数

```
[0x08048370]> afl
0x080482ec    3 35           sym._init
0x08048320    1 6            sym.imp.strcmp
0x08048330    1 6            sym.imp.strcpy
0x08048340    1 6            sym.imp.puts
0x08048350    1 6            sym.imp.__libc_start_main
0x08048360    1 6            sub.__gmon_start___252_360
0x08048370    1 33           entry0
0x080483a0    1 4            sym.__x86.get_pc_thunk.bx
0x080483b0    4 43           sym.deregister_tm_clones
0x080483e0    4 53           sym.register_tm_clones
0x08048420    3 30           sym.__do_global_dtors_aux
0x08048440    4 43   -> 40    sym.frame_dummy
0x0804846b   19 282          sym.rot13
0x08048585    1 112          sym.beet
0x080485f5    5 127          main
0x08048680    4 93           sym.__libc_csu_init
0x080486e0    1 2            sym.__libc_csu_fini
0x080486e4    1 20           sym._fini
[0x08048370]>
```

（图片来源：hetianlab.com）

看到两个引起我们注意的sym.beet和sym.rot13

接下来我们用 's main' 指令定位到main函数入口处，然后用 'pdf'输出反汇编代码



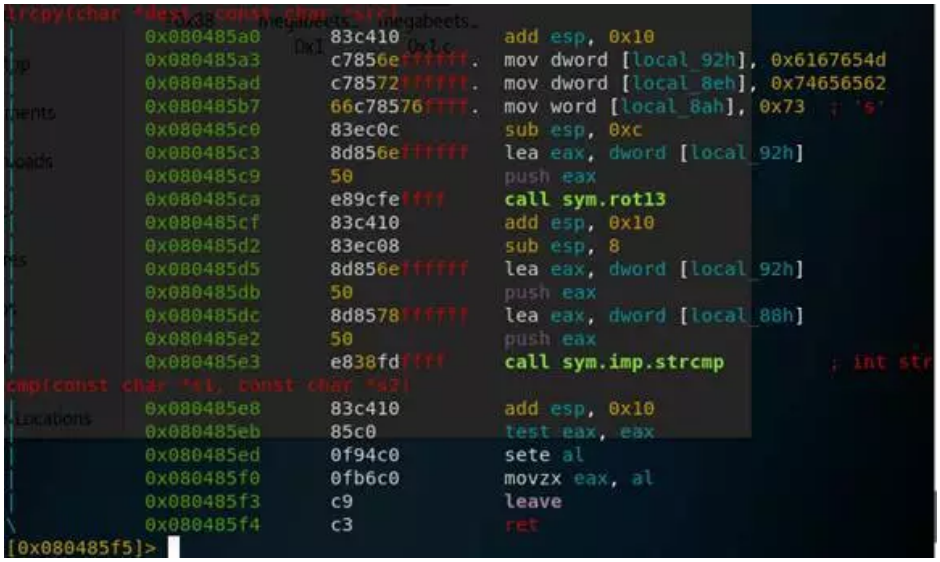
（图片来源：hetianlab.com）

分析函数的执行流程，我们知道二进制程序是通过获取 beet函数的返回结果来判断是否正确

因此我们去beet函数反汇编的部分看看

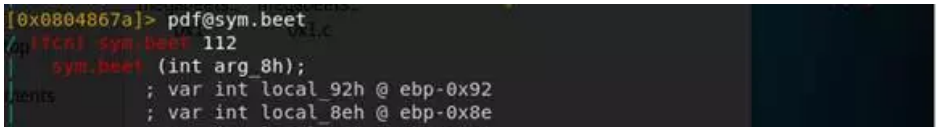
因此我们定位到beet

输入pdf@sym.beet进行跳转



（图片来源：hetianlab.com）

自动跳转到beet函数的反汇编部分

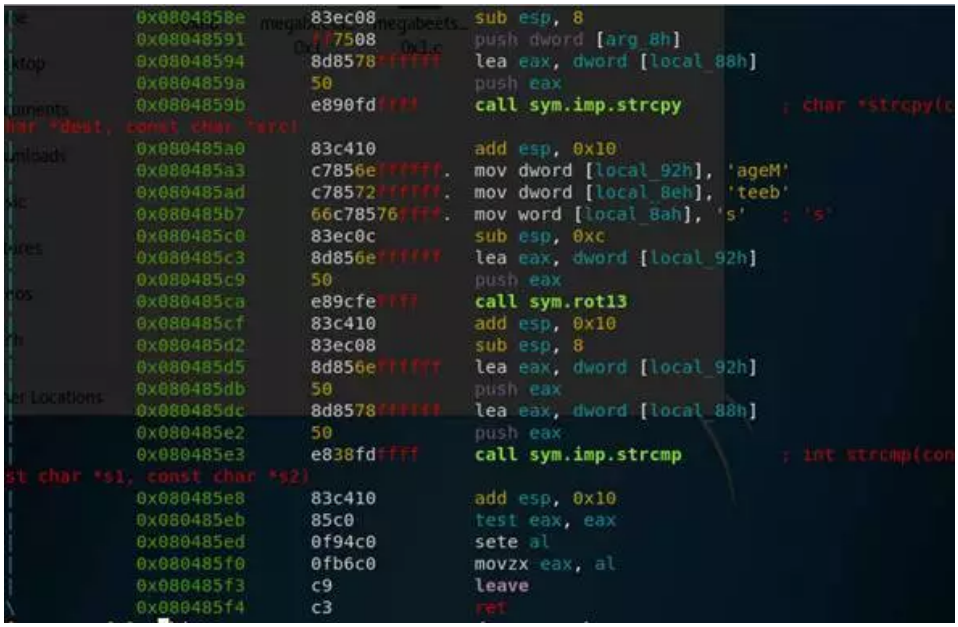


（图片来源：hetianlab.com）

我们看到输入的参数被拷贝到了一个缓存空间里，这个空间的地址是 'ebp – local_88h' 。 'local_88h' 就是十进制的 136。由于4个字节会被用来保存 ebp 的地址，4个字节被用来保存返回地址，所以这个缓冲区得大小是 128个字节.它们加起来刚好是 136. 我们输入的参数被拷贝到缓冲区后被用来和 sym.rot13的返回结果作对比， Rot-13 是一个著名的替换密码算法，在ctf和crackme中被广泛使用，这个函数接受了9个十六进制值作为参数，但是上图中看起来r2好像没有识别出来到底是什么字符，这里我们需要用 'ahi s' 来做些处理.输入

（图片来源：hetianlab.com）

ahi s 是用来设置字符串特定的偏移地址（使用 ahi? 获取更多用法），@@是一个迭代器，可以用来接受后面输入的多个参数,执行完这条命令后，图形视图会自动刷新。



（图片来源：hetianlab.com）

可以看到

0x080485a3

0x080485ad

0x080485b7

后面的字符都已经显示出来了

我们已经看到了之前无法识别的字符串'Megabeets'(根据字节序反向压栈顺序得到).

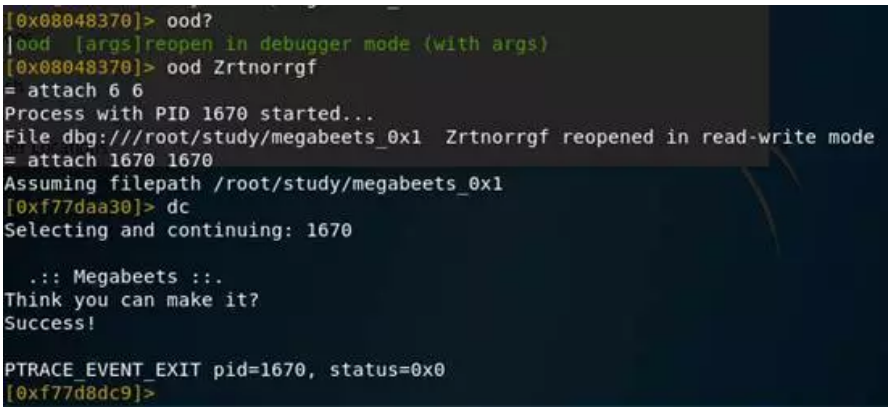
这个二进制文件将我们传入的参数来和经过 rot13 处理后的 'Megabeets' 作比较

接下来我们通过rahash2求出这个字符串的校验值

```
root@kali:~# rahash2 -E rot -S s:13 -s 'Megabeets'
'Zrtnorrgf'root@kali:~#
```

（图片来源：hetianlab.com）

至此，程序的逻辑就很清楚了： 'Zrtnorrgf' 就是用来和我们输入的字符串作比较，成功则返回success。我们验证一下：接下来输入ood?进入调试模式 将Zrtnorrgf作为参数进行调试，输入dc查看结果



（图片来源：hetianlab.com）

输出了success，我们成功破解了这个小软件，也借此掌握了radare2的基本用法。

是不是跃跃欲试了呢？实践才能学到真本领！戳下面的链接即可进行在线实操，无需自己搭建环境噢！

戳它>>Radare2实战 (https://link.jianshu.com?t=http%3A%2F%2Fwww.hetianlab.com%2Fcour.do%3Fw%3D1%26c%3DCee9320adea6e062018011816570500001)<<戳它

(本文来源：合天网安实验室 (https://link.jianshu.com?t=http%3A%2F%2Fwww.hetianlab.com)--一家实操型在线网络安全学习平台)

小礼物走一走，来简书关注我