Linux下pwn从入门到放弃

niexinming / 2017-12-15 16:30:00 / 浏览数 3923 新手 入门资料 顶(0) 踩(0)

0x0 简介

pwn，在安全领域中指的是通过二进制/系统调用等方式获得目标主机的shell。
虽然web系统在互联网中占有比较大的分量，但是随着移动端，ioT的逐渐流行，传统的缓冲区溢出又一次有了用武之处

0x01 工欲善其事，必先利其器

Linux下的pwn常用到的工具有：
（1）gdb：Linux调试中必要用到的
（2）gdb-peda：gdb方便调试的工具，类似的工具有gef，gdbinit，这些工具的安装可以参考：http://blog.csdn.net/gatieme/article/details/63254211
（3）pwntools:写exp和poc的利器
（4）checksec：可以很方便的知道elf程序的安全性和程序的运行平台
（5）objdump和readelf：可以很快的知道elf程序中的关键信息
（6）ida pro ：强大的反编译工具
（7）ROPgadget：强大的rop利用工具
（8）one_gadget：可以快速的寻找libc中的调用exec('bin/sh')的位置
（9）libc-database： 可以通过泄露的libc的某个函数地址查出远程系统是用的哪个libc版本

0x02 检测elf的安全性:

（1)拿到efl，首先要用checksec来检测elf运行于哪个平台，开启了什么安全措施，如果用gcc的编译后，默认会开启所有的安全措施。

【1】RELRO：RELRO会有Partial RELRO和FULL RELRO，如果开启FULL RELRO，意味着我们无法修改got表
【2】Stack：如果栈中开启Canary found，那么就不能用直接用溢出的方法覆盖栈中返回地址，而且要通过改写指针与局部变量、leak canary、overwrite canary的方法来绕过
【3】NX：NX enabled如果这个保护开启就是意味着栈中数据没有执行权限，以前的经常用的call esp或者jmp esp的方法就不能使用，但是可以利用rop这种方法绕过
【4】PIE：PIE enabled如果程序开启这个地址随机化选项就意味着程序每次运行的时候地址都会变化，而如果没有开PIE的话那么No PIE (0x400000)，括号内的数据就是程序的基地址
【5】FORTIFY：FORTIFY_SOURCE机制对格式化字符串有两个限制(1)包含%n的格式化字符串不能位于程序内存中的可写地址。(2)当使用位置参数时，必须使用范围内的用

0x03 调试技巧

gdb常用的调试指令：
n： 执行一行源代码但不进入函数内部
ni: 执行一行汇编代码但不进入函数内部
s： 执行一行源代码而且进入函数内部
si: 执行一行汇编代码而且进入函数内部
c: 继续执行到下一个断点
b *地址: 下断点
directory+源码所在目录：加载程序源码
set follow-fork-mode parent ：只调试主进程
stack: 显示栈信息
x ： 按十六进制格式显示内存数据，其中x/{字节数}x 以16进制显示指定地址处的数据;{字节数}表示字节数制定（b 单字节；h 双字节；w 四字节；g 八字节；默认为四字节）

程序没有开启地址随机化：

```
def debug(addr):
   raw_input('debug:')
   gdb.attach(r, "b *" + addr)
```

在程序运行时调用这个函数就可以调试了

程序开启地址随机化：

```
wordSz = 4
hwordSz = 2
bits = 32
PIE = 0
mypid=0
def leak(address, size):
  with open('/proc/%s/mem' % mypid) as mem:
     mem.seek(address)
```

```
    return mem.read(size)

def findModuleBase(pid, mem):
  name = os.readlink('/proc/%s/exe' % pid)
  with open('/proc/%s/maps' % pid) as maps:
     for line in maps:
        if name in line:
           addr = int(line.split('-')[0], 16)
           mem.seek(addr)
           if mem.read(4) == "\x7fELF":
              bitFormat = u8(leak(addr + 4, 1))
              if bitFormat == 2:
                 global wordSz
                 global hwordSz
                 global bits
                 wordSz = 8
                 hwordSz = 4
                 bits = 64
              return addr
  log.failure("Module's base address not found.")
  sys.exit(1)

def debug(addr = 0):
   global mypid
   mypid = proc.pidof(r)[0]
   raw_input('debug:')
   with open('/proc/%s/mem' % mypid) as mem:
       moduleBase = findModuleBase(mypid, mem)
       gdb.attach(r, "set follow-fork-mode parent\nb *" + hex(moduleBase+addr))
```

由于开启地址随机化之后ida
pro打开程序后，显示的是程序的偏移地址，而不是实际的地址，当程序加载后程序的程序的实际地址是：基地址+偏移地址，调用debug函数的时候只要把偏移地址传递进

0x4 泄露libc地址和版本的方法

【1】利用格式化字符串漏洞泄露栈中的数据，从而找到libc的某个函数地址，再利用libc-database来判断远程libc的版本，之后再计算出libc的基址，一般做题我喜欢找__li
【2】利用write这个函数，pwntools有个很好用的函数DynELF去利用这个函数计算出程序的各种地址，包括函数的基地址，libc的基地址，libc中system的地址
【3】利用printf函数，printf函数输出的时候遇到0x00时候会停止输出，如果输入的时候没有在最后的字节处填充0x00，那么输出的时候就可能泄露栈中的重要数据，比如

0x05 简单的栈溢出

程序没有开启任何保护:
方法一：传统的教材思路是把shellcode写入栈中，然后查找程序中或者libc中有没有call esp或者jmp esp，比如这个题目：
http://blog.csdn.net/niexinming/article/details/76893510
方法二：但是现代操作系统中libc中会开启地址随机化，所以先寻找程序中system的函数，再布局栈空间，调用gets(.bss)，最后调用system('/bin/sh')
比如这个题目：http://blog.csdn.net/niexinming/article/details/78796408
方法三：覆盖虚表方式利用栈溢出漏洞，这个方法是m4x师傅教我的方法，我觉得很巧妙，比如这个题目：http://blog.csdn.net/niexinming/article/details/78144301

0x06 开启nx的程序

开启nx之后栈和bss段就只有读写权限，没有执行权限了，所以就要用到rop这种方法拿到系统权限，如果程序很复杂，或者程序用的是静态编译的话，那么就可以使用ROP
http://blog.csdn.net/niexinming/article/details/78259866

0x07 开启canary的程序

开启canary后就不能直接使用普通的溢出方法来覆盖栈中的函数返回地址了，要用一些巧妙的方法来绕过或者利canary本身的弱点来攻击
【1】利用canary泄露flag，这个方法很巧妙的运用了canary本身的弱点，当stack_check_fail时，会打印出正在运行中程序的名称，所以，我们只要将libc_argv[0]覆盖为fla
http://blog.csdn.net/niexinming/article/details/78522682
【2】利用printf函数泄露一个子进程的Canary，再在另一个子进程栈中伪造Canary就可以绕过Canary的保护了，比如这个题目：http://blog.csdn.net/niexinming/article

0x08 开启PIE的程序

【1】利用printf函数尽量多打印一些栈中的数据，根据泄露的地址来计算程序基地址，libc基地址，system地址，比如这篇文章中echo2的wp：
http://blog.csdn.net/niexinming/article/details/78512274
【2】利用write泄露程序的关键信息，这样的话可以很方便的用DynELF这个函数了，比如这个文章中的rsbo2的题解：http://blog.csdn.net/niexinming/article/details/7

0x09 全部保护开启

如果程序的栈可以被完全控制，那么程序的保护全打开也会被攻破，比如这个题目：http://blog.csdn.net/niexinming/article/details/78666941

0x0a 格式化字符串漏洞

格式化漏洞现在很难在成熟的软件中遇到，但是这个漏洞却很有趣
【1】pwntools有很不错的函数FmtStr和fmtstr_payload来自动计算格式化漏洞的利用点，并且自动生成payload，比如这个题目：http://blog.csdn.net/niexinming/arti
和 http://blog.csdn.net/niexinming/article/details/78512274 中echo的题解
【2】格式化漏洞也是信息泄露的好伴侣，比如这个题目中制造格式化字符串漏洞泄露各种数据 http://blog.csdn.net/niexinming/article/details/78768850

0x0b uaf漏洞

如果把堆释放之后，没有把指针指针清0，还让指针保存下来，那么就会引发很多问题，比如这个题目
http://blog.csdn.net/niexinming/article/details/78598635

0x0c 任意位置写

如果程序可以在内存中的任意位置写的话，那么威力绝对很大
【1】虽然只能写一个字节，但是依然可以控制程序的并getshell，比如这个题目 http://blog.csdn.net/niexinming/article/details/78542089
【2】修改got表是个控制程序流程的好办法，很多ctf题目只要能通过各种方法控制got的写入，就可以最终得到胜利，比如这个题目：
http://blog.csdn.net/niexinming/article/details/78542089
【3】如果能计算出libc的基地址的话，控制top_chunk指针也是解题的好方法，比如这个题目： http://blog.csdn.net/niexinming/article/details/78759363

上一篇：Web应用程序安全测试备忘录 下一篇：域渗透——Pass The Has...
1. 0 条回复
   - 动动手指，沙发就是你的了！