玩转堆-堆漏洞的利用技巧

讲师：Atum

内容简介

INTRODUCTION

01 基础知识

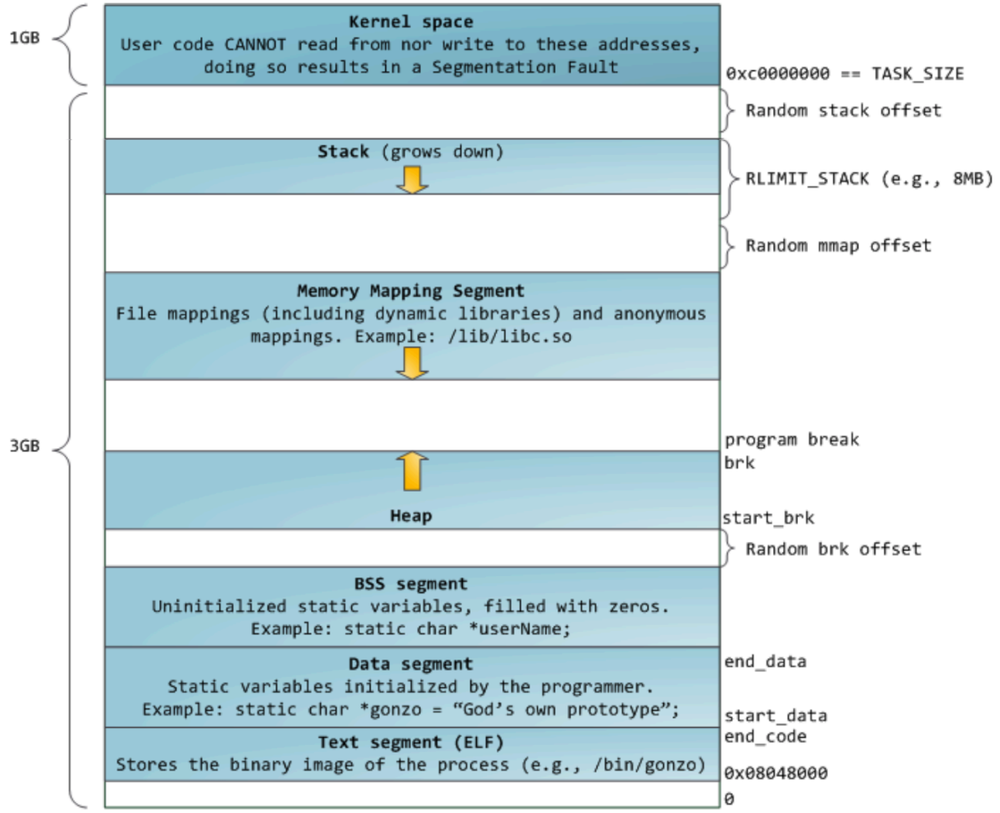
02 堆溢出的利用思想与防护策略

03 堆溢出的利用技术与技巧

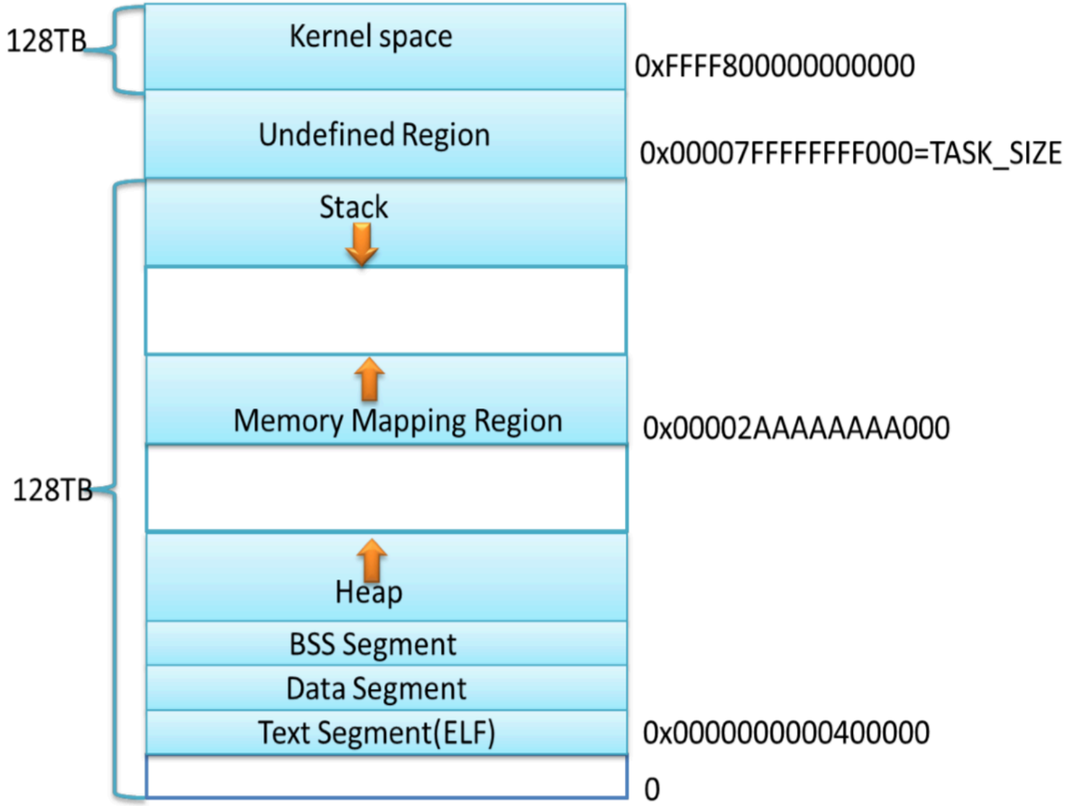
基础知识

操作系统中的内存布局(Linux):

内核空间&用户空间，堆、栈等; cat /proc/pid/maps



32位内存布局



64位内存布局

基础知识

什么是堆？

- 数据结构：父节点总大于/小于子节点的特殊的完全二叉树
- 操作系统：程序在运行时动态申请和释放的内存空间(malloc, realloc, free, new, del等)

不同操作系统对堆内存有不同的管理策略，某些软件(如浏览器)会自己实现堆内存管理

- 多数Linux发行版(with glibc): **ptmalloc/dlmalloc**
- Android/firefox:jemalloc
- Windows: 微软自己实现了一套内存管理机制
- Linux内核: slab，slub, slob 分配器
- ....

了解内存管理的策略是玩转堆漏洞的关键

基础知识

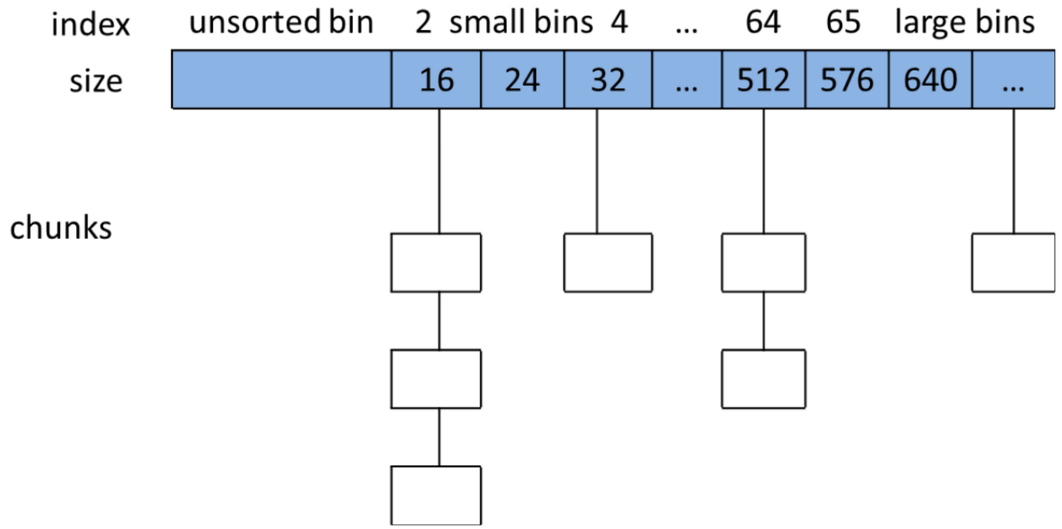
CTF题中堆管理机制：大多数是ptmalloc/dlmalloc, 少数题中自己实现

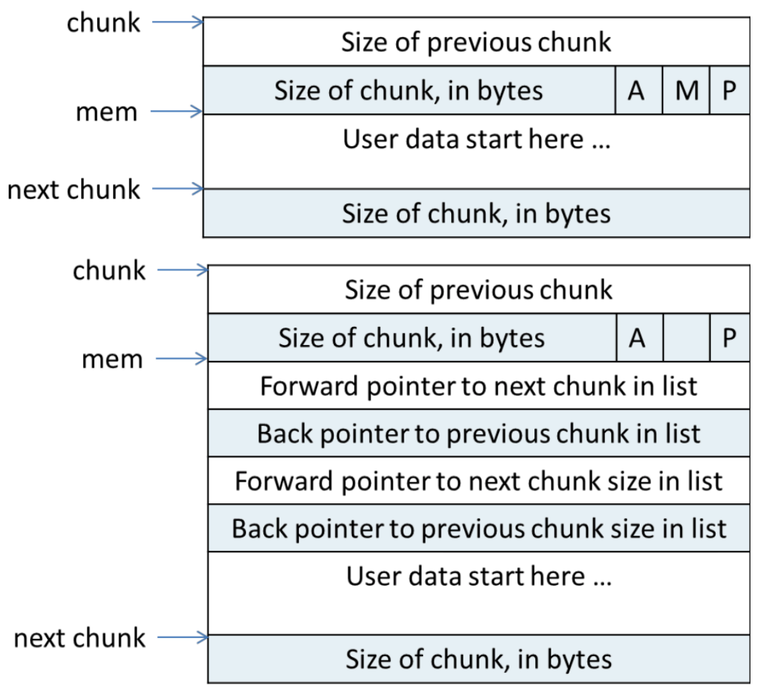
ptmalloc/dlmalloc是glibc的默认内存管理机制，了解它对PWN堆题来说至关重要！

强烈推荐： glibc内存管理ptmalloc源代码分析.pdf　建议**先通读，再用作工具书**

作业： 精读“glibc内存管理ptmalloc源代码分析.pdf”的1-27页，粗读28-130页(end)

arena, bin, chunk. Know it and pwn it!





堆漏洞的利用思想与防护策略

堆漏洞的利用思想：

- 破坏堆内存管理的相关数据结构：如arena、bin、chunk
- 破坏堆内存中的用户数据：覆盖变量指针、函数指针、数据等
- 一般情况下都是为了构造任意内存读写以及控制流劫持

堆漏洞的防护方法：

- 保护堆内存管理相关的数据结构：
  - Heap Canary、对数据结构进行加密、**在堆管理代码中加入大量安全检查**
- 通用防护：**ASLR，DEP**

```
bck = unsorted_chunks(av);
fwd = bck->fd;
if (__builtin_expect (fwd->bk != bck, 0))
  {
    errstr = "free(): corrupted unsorted chunks";
    goto errout;
  }
p->fd = fwd;
p->bk = bck;
if (!in_smallbin_range(size))
  {
    p->fd_nextsize = NULL;
    p->bk_nextsize = NULL;
  }
bck->fd = p;
fwd->bk = p;
```

以上保护机制在Win10中基本已经全部开启，但是在CTF环境中（Linux with glibc），除了通用防护以及堆管理中的安全检查，剩下的都均默认关闭或未实现

堆漏洞的利用技术与技巧

Use After Free & Double Free

Heap Overflow

- Overflow directly
- Fast bin attack
- Unsorted bin attack
- Overwrite Topchunk
- Classical&Modern Unlink Attack
- Off by one & Off by null
- Other techniques

General exploit techniques

- Heap fengshui
- Heap spray
- Exploit mmap chunk

Use After Free & Double Free

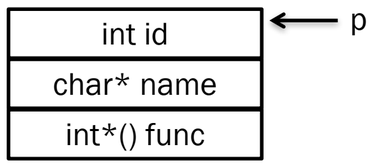
Dangling pointer: 指向被释放的内存的指针，通常是由于释放内存后未将指针置null

Use After Free: 对Dangling pointer所指向内存进行use。如指针解引用等。

Use After Free漏洞根据use的方式会产生不同的危害。

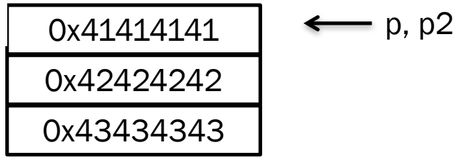
UAF的利用思路：

- 想办法将Dangling pointer指向的内存重新分配回来，且尽可能的使该内存中的内容可控（如重新分配为字符串）。如果use的方式是打印*(Dangling pointer+8)，那就会产生任意地址读，如果use方式是将*(Dangling pointer+12)作为函数指针进行调用，那就可以劫持控制流！



free(p) without set p =null

P is dangling now



char *p2=(char*)malloc(12)

strcpy(p2,” AAAABBBBCCCC”）

puts(p->name) => puts((char*)(0x42424242)) 任意地址读！

p->func() => call 0x43434343 控制流劫持！

strcpy(p->name,str) => strcpy((char*)(0x42424242,str)) 任意地址写！

free(p) => p2 is Dangling too; Double free is special UAF!

Use After Free & Double Free

Double Free: UAF中的use为再次free, 是一种特殊的UAF，且可转换为普通的UAF

转换：free(p), p2=malloc(), p2与p指向同一个内存free(p), p2为Dangling pointer=>UAF

习题：

- UAF：DEFCON CTF Qualifier 2014: shitsco、BCTF 2016: router、HCTF2016 5-days(较难)
- Double Free：0CTF2016: freenote、HCTF2016 fheap 、HCTF2016 5-days(较难)

作业：完成习题shitsco (writeup很多)

参考阅读：

- https://blog.skullsecurity.org/2014/defcon-quals-writeup-for-shitsco-use-after-free-vuln
- http://www.tuicool.com/articles/yquU732
- http://blog.csdn.net/sdulibh/article/details/47375969

Heap Overflow-Overflow Directly

直接覆盖相邻堆块的内存的内容。

关键：如何让想被覆盖的堆块正好在具有溢出漏洞的堆块之后。

- 堆风水/堆排布：通过操纵内存的分配与释放，来控制堆块装内存中的相对位置。
- 堆排布几乎是所有堆漏洞利用所必需的技能，需要对glibc内存管理策略非常熟悉。知道什么时候分析什么内存。再次强调：一定要阅读且经常翻阅“glibc内存管理ptmalloc源代码分析.pdf” ！！！

例题：XMAN2016 fengshui(zijinghua pwn)，SSC安全大会百度展厅 heapcanary

作业：完成heapcananry

其实真实环境中大多数漏洞都是通过这种方式进行利用的。但是CTF中不算特别常见。（因为太简单了）

Heap Overflow-Fast bin attack

改写fastbin单向链表中的fd，那再次分配就会分配到被改写的fd指向的地址

改写目标必须有一个正确的size对应，否则会挂

CTF中的套路：如果bss上有指针，通常会改到bss的指针附近，再次分配可以分配到bss地址，修改新分配的内容便可以修改bss上的指针。

另外还有：House of Spirit

例题：

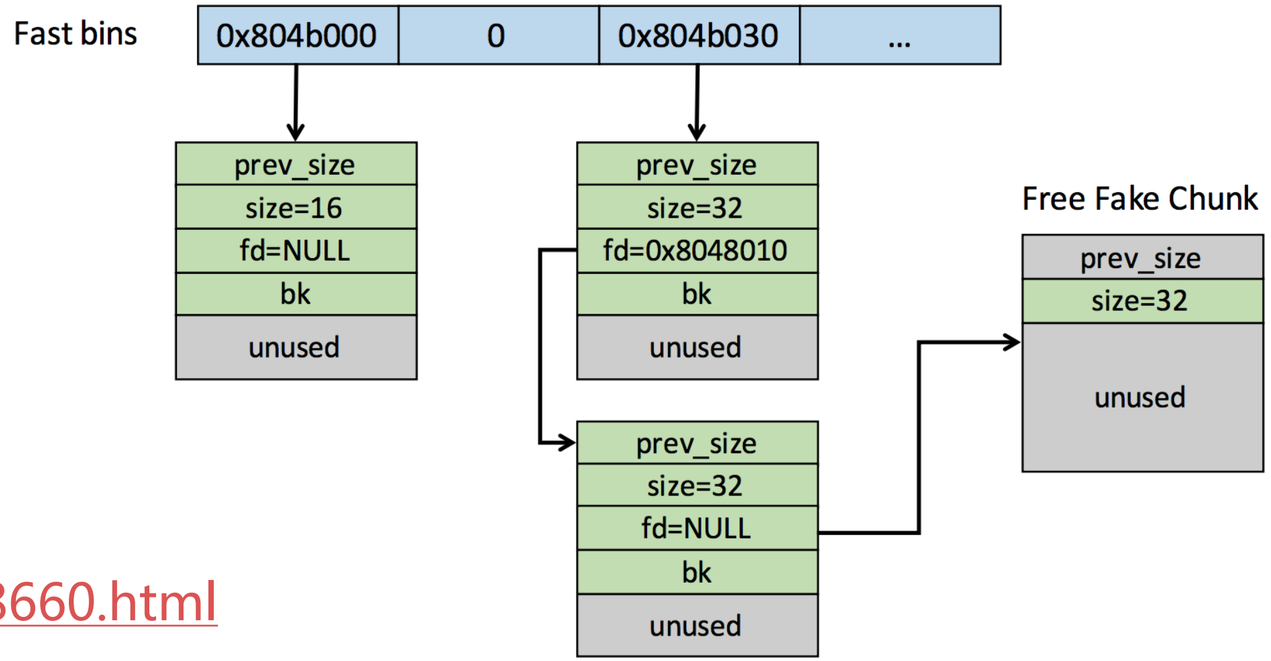
alictf 2016 fb (作业，推荐完成）

alictf 2016 starcraft

0ctf 2016 zerostorage （比较难）

推荐阅读：

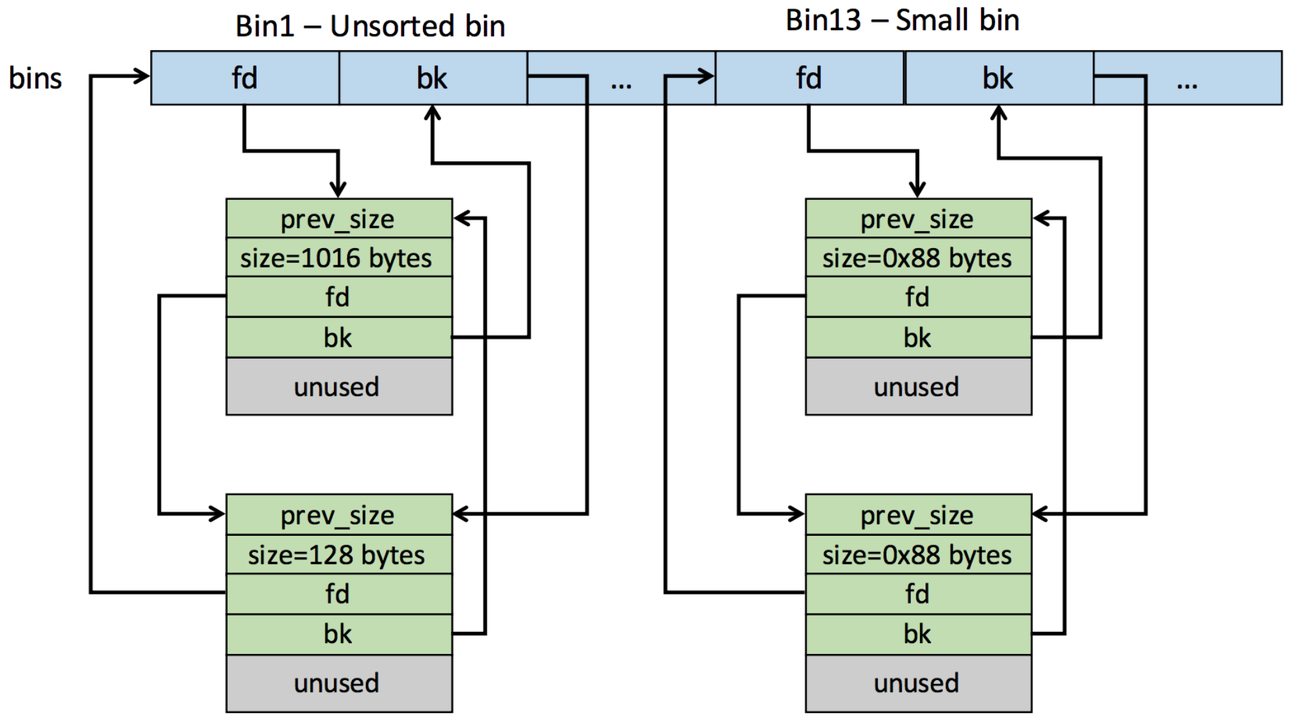
- http://www.freebuf.com/news/88660.html
- http://angelboy.logdown.com/posts/291983-heap-exploitation



Heap Overflow-unsorted bin attack

当需要分配的内存无法在fastbin或者smallbin找到时，glibc会从unsort bins的链表头的bk开始遍历，遍历过程中会把unsortbin中的块加入合适的smallbin/largebin中，如果找到合适大小内存块则会返回。

```
bck = victim->bk;
...
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```



Heap Overflow-unsorted bin attack

利用思路：

- 通过堆溢出覆盖victim->bk为要写入的地址-4，再次分配时bck->fd = unsorted_chunks (av)会触发一个任意地址写。写入的内容是libc中的一个地址。只不过此时unsortbin被破坏，再次分配代码会崩掉，所以要谨慎考虑写入的地址，通常可以改写global_max_fast, 从而导致接下来所有分配都是在fastbin进行。
- 通过堆溢出覆盖victim->bk为一个size为x的fake chunk，再次分配unsorted_chunks (av)->bk = bck会改写unsortbin链表头的bk, 此时再分配x-4大小的内存即可返回fakechunk。

习题：

- 0ctf2016 Zerostorage(http://brieflyx.me/2016/ctf-writeups/0ctf-2016-zerostorage/)
  - 第一步unsortedbin attack改写global max fast，第二步fastbin attack

Heap Overflow-Overwrite Topchunk

House of Force：

- Bin中没有任何合适的内存时会从Topchunk分配内存：
  - **If(topchunk->size > alloc_size) { victim = topchunk; topchunk = topchunk - alloc_size; return victim; }**
- 改写Topchunk的size为一个很大的数，如0xffffffff, 分配alloc_size-4大小的内存。由于alloc_size可控，所以此时topchunk位置可控，再次分配即可分配到想分配的位置
- 需要预先泄漏topchunk 的地址

例题：BCTF 2016 bcloud(推荐完成) BCTF 2016 ruin(arm结构的程序，选做)

推荐阅读：

- https://gbmaster.wordpress.com/2015/06/28/x86-exploitation-101-house-of-force-jedi-overflow/
- http://angelboy.logdown.com/posts/291983-heap-exploitation

Heap Overflow-Classical&Modern Unlink Attack

Unlink:当free(mem)调用时，如果与mem相邻的块是空闲的，则会将其从空闲链表中拿（unlink）下来并与mem合并。

```
#define  unlink(  P, BK, FD ) {
  BK = P->bk;
  FD =  P->fd;
  FD->bk = BK;
  BK->fd = FD;
}
```

Classical Unlink Attack:

如果通过heapoverflow将P->bk 以及P->fd覆盖为攻击者可控制的地址，那FD->bk = BK; BK->fd = FD; => P->fd->bk=P->bk; P->bk->fd=p->fd; 造成任意写。不过要求 (要写的内容+4) or (要写的内容+8) 必须可写，否则会崩。

已不可用，现代glibc已有此检查：**P->fd->bk == P&&P->bk->fd == P**

Heap Overflow-Classical&Modern Unlink Attack

Modern Unlink Attack:

- 找一个Pointer X, *X=P, Overflow P->bk=X-4; P->fd=X-8
- P->bk->fd == X-4->fd ==P, P->fd->bk == X-8->bk ==P
- Unlink 可得到 *P=X，此时可通过P修改X，如果X是数据指针则可能造成任意地址读写。

例题：

- Hitcon 2014 qualifier stkof (Modern Unlink Attack) (作业 推荐完成)
- MMA CTF 2016 Dairy (Off by one + Classic Unlink Attack + sandbox bypass)
- PlaidCTF 2014 200 ezhp (Classic Unlink Attack) (作业 推荐完成)

推荐阅读：

- https://gbmaster.wordpress.com/2014/08/11/x86-exploitation-101-heap-overflows-unlink-me-would-you-please/
- http://acez.re/ctf-writeup-hitcon-ctf-2014-stkof-or-modern-heap-overflow/

Off by one & Off by null

Off by one: 溢出位数为1的溢出漏洞

Off by null: 溢出位数为1且溢出内容为null的溢出漏洞

在glibc中，如果攻击者可以控制malloc的大小和malloc与free的时机，堆中的Off by one和Off by null是可用的，通常可以构造出UAF，进而构造任意地址读写&控制流劫持。

主要利用思路：改写下一个chunk的chunk size(including inuse bit)

作业：阅读论文Glibc_Adventures-The_Forgotten_Chunks.pdf

习题：

- Off By one: MMA CTF 2016 Dairy (Off by one + Classic Unlink Attack + sandbox bypass)
- Off By null: plaid CTF 2015 datastore，XMAN 2016 Final love_letter

其他推荐阅读

- http://angelboy.logdown.com/posts/567673-advanced-heap-exploitation

Other techniques

改写morecore

- 在HCTF 2016 5-days首次出现
- 5-day wp.pdf

House of Orange：改写_IO_list_all ，在hitcon 2016首次出现

- Hitcon 2016 House of orange
- http://angelboy.tw

作业（选做）：

- 完成题目5-day House of Orage

General exploit techniques-Heap fengshui

高级堆排布技术：Heap fengshui

动机：真实漏洞在利用的时候，堆是混乱的，因为存在漏洞的服务可能已经服务过很多用户，在触发漏洞时无法预计堆已经做了多少次malloc多少次free。

Heap fengshui可以让堆从混乱状态转换为确定状态

不同的内存管理策略对应的heap fengshui 的方法不同，Example：

- For glibc fastbin：把每种可能的大小都分配好多次。

CTF题目一般不需要利用这种技术，因为大多数CTF题目都是程序启动后立刻被攻击者利用，堆处于确定的状态。

例题：XMAN 2016 fengshui, 33c3 CTF babyfengshui

作业：完成以上题目

General exploit techniques-Heap spray

Heap spray, 堆喷: 不断分配内存，并填充 (大量0x0c)+shellcode, 直到0x0c0c0c0c内存地址被分配，多用于脚本语言漏洞的利用。

大多数内存地址的值都是0x0c0c0c0c，0x0c0c0c0c地址也是 0x0c slide+shellcode可以用其绕过ASLR，控制流劫持(jmp addr/jmp *addr)时，只要addr是喷过地址都可以执行shellcode, 注意*addr=0x0c0c0c0c **addr=0x0c0c0c0c ***addr=0x0c0c0c。

必须在NX关闭时才能直接利用heap spray劫持控制流

例题： pwnhub.cn calc

推荐阅读：

- http://www.tuicool.com/articles/3uI
- https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/BJv

General exploit techniques-Exploit mmap chunk

mmaped chunk:当malloc的块的大小大于128KB时，glibc会直接mmap内存。

如果mmap的内存将整个binary 的地址空间全部覆盖，我们可以轻松拿到与任意地址相邻的堆内存，ASLR就失去了意义。

适用于没有限制分配内存大小的题目

例题： Hitcon 2014 qualifier stkof (unintended solution)

CTF用的不多, 所以不细讲

推荐阅读：0ops培训资料 Linux heap internal.pdf

The End