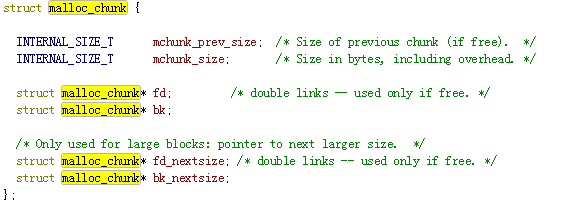
关于unlink的一些理解

发现利用unlink机制的漏洞利用可分为两步，首先利用某漏洞达到伪造条件，然后free某chunk指针触发相邻chunk(第一步伪造的)被unlink掉。

导致的结果是存放被unlink掉的”伪造chunk”的地址的地址上被放置为该地址-0x18，也就是伪造chunk所在chunk的可用区间地址改为放地址的地址-0x18了，这样就可以有一个任意地址修改了。

1. 前提知识:

Chunk结构:



第0个单位长度(64位下为8字节)表示前个chunk的大小；

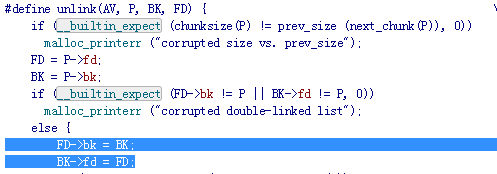
//前一个chunk使用中，则为0；64位下chunk大小16字节对齐(32位下8字节)；后3位用不大则用作flag，其中最后一位和prev\_size一块来判断前个chunk是否空闲以及其位置(0时表示空闲)(需要伪造的条件1)。

第1个单位长度表示本chunk的大小；

//chunk的大小包含其header大小。

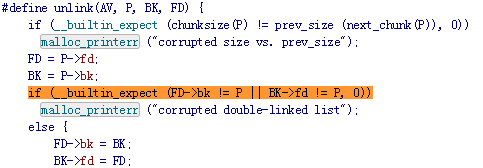
1. Unlink

Free一个chunk时会先检查前个和后个chunk是否为空闲，空闲则使用unlink机制将其从中链表删除。



经典的链表节点删除。

为了防止溢出恶意修改fd,bk信息，有一个Safe\_unlink判断如下，确保fd和bk指针的有效性，即检测fd指针指向的堆块的bk指针是指向自己。(需要伪造的条件2)



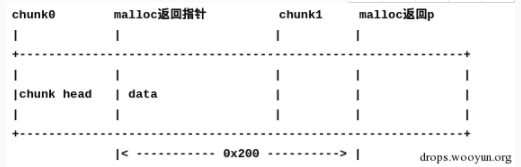
2漏洞利用实例

2.1 double free漏洞(32位下)

>malloc(504)

>malloc(512)

然后free这两块内存。这样就可以在距离第一个指针偏移量为0x200(504+8)的地方有了一个野指针。

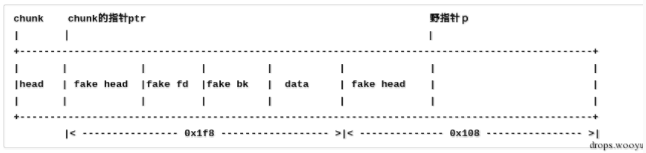


此时，我们有一个野指针chunk1\_p。需要做的是伪造条件，然后free野指针。

首先，申请一块大于0X200的chunk3，这样我们就可以构造伪造条件了。

Chunk3内容：

>0x0+0x1f9+0x0804bfc0 - 0xc + 0x0804bfc0 - 0x8 + 'a'\*(0x200-24) + 0x000001f8 + 0x108



0x0和0x1f9代表了前个chunk正在使用，当前chunk的大小是0x1f8；然后是伪造的fd和bk，0x0804bfc0是.bss段中存放malloc地址的地址(是chunk3的可用区间地址，也是伪造的chunk的地址)，这样就满足伪造chunk的FD->bk==BK->fd==P(伪造条件2)。0x000001f8 + 0x108指示野指针chunk的前个chunk(即伪造的chunk)处于空闲状态(伪造条件1)。

然后，free野指针，就会导致.bss段中存放malloc地址的地址上的内容变为 .bss段中存放malloc地址的地址-0xc 。

结果就是chunk3的可用区间地址变为存放chunk3地址的地址-0xc了。通过edit功能edit chunk3+0xc处的4字节，可继续改变chunk3的地址了，再edit时就可以edit任意地址了。

好像只能一次，任意地址的改写，因为此时chunk3的地址已经变为第一次的任意地址了，不能再更改chunk3的地址为下个任意地址了。

2.2溢出漏洞(64位下)

与上一个类似，不过是利用的溢出漏洞，

漏洞利用不同处：

利用任意地址修改，把free\_got改为chunk\_3的地址，所以只能更改free\_got的内容；

//可把free\_got改为不同函数的地址，这样想调用函数x，只需先edit free\_got为对应函数的地址，然后调用free函数。

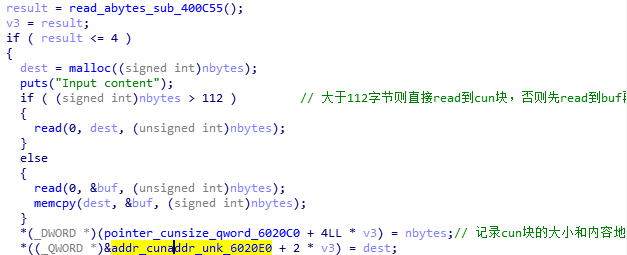
实例是：

1. 改为puts，参数为puts\_got得到puts函数地址；
2. 由偏移关系得到system函数地址
3. 改为system函数地址，参数为存放”bin/sh”的chunk的可用空间地址。

这样既可get\_shell了。

具体分析:

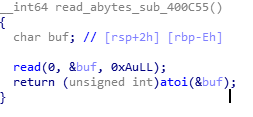
1. 存在溢出漏洞



Cre\_cun块，把malloc的地址放到地址6020e0(某偏移)上，malloc的大小放到指针变量6020c0(地址6020c0上的地址)(某偏移上)。

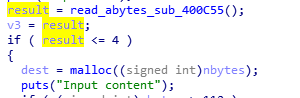
注意：地址+2\*v3受单位长度影响(qword);指针变量的值就是一个数值，+4\*v3没有单位长度的影响。简单理解，指针变量是数值，则直接相加；地址则是偏移了，考虑单位长度。

溢出点在：

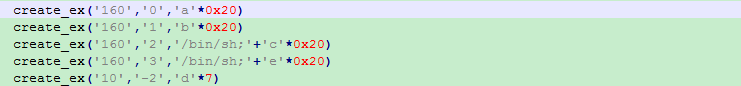


IDA这里分析为unsigned int强制转换，其实是没有的，该函数会直接转化为负数。

可以在后面的if中得以验证:



这样当输入-2的时候，就把malloc的地址放到 存放size的指针变量上了 ，结果malloc的索引为-2的地址是指针变量值了，即索引-2的chunk块存放size了，size就间接溢出为很大了。



1. Unlink使得chunk0的地址被修改为 存放该地址的地址-0x18

edit chunk0(此时其size已经很大)，伪造chunk0的fd和bk过掉safe\_unlink(伪造条件2)，伪造后一个chunk的header(伪造条件1)



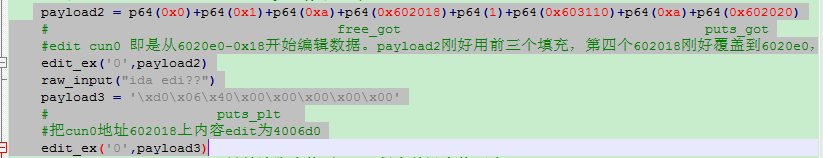
edit\_ex('0',payload)

delete\_ex('1')

unlink之后，效果是存放chunk0地址的地址被覆盖为 存放chunk0地址的地址-0x18 ，这样就可以覆盖chunk0的地址为任意地址了，然后就可以edit任意地址了。

注意:由于一般只能edit一个任意地址，最好高效利用。

1. 由2 edit chunk0地址+0x18 可edit任意地址



这里edit free\_got为puts\_plt

4>”free”

首先，free chunk2，此时 存放chunk2地址处 已经被放置为puts\_got，这样就会显示puts的函数地址

//由偏移关系可得到system函数地址

然后,free chunk3，chunk3地址上最开始已经被放置为” /bin/sh;”，可直接作为参数。

注意：

<1>共享库中库函数地址的rva可知：



<2>程序中库函数的PLT地址和GOT地址也是可知的:

Plt地址用objdump -d -j .plt ./victim\_nx | grep puts;

got地址用r；

//plt地址处：call func@got;所以如果只是想劫持流程使用函数而非具体知道函数地址可以直接用func@PLT地址覆盖。

参考链接:

<https://bbs.pediy.com/thread-218318.htm>

<https://bbs.pediy.com/thread-218313.htm>

<http://www.vuln.cn/6172>