

# 从一字节溢出到任意代码执行-Linux下堆漏洞利用

阅读量 **35463** | 评论 **3** 稿费 **500**

发布时间：2016-10-19 10:40:20

分享到：



作者：**Ox9A82**

稿费：**500RMB**（不服你也来投稿啊！）

投稿方式：发送邮件至[linwei#360.cn](mailto:linwei#360.cn)，或登陆网页版在线投稿

一个字节溢出被称为off-by-one，曾经的一段时间里，off-by-one被认为是不可以利用的，但是后来研究发现在堆上哪怕只有一个字节的溢出也会导致任意代码的执行。同时堆的off-by-one利用也出现在国内外的各类CTF竞赛中，但是在网络上还不能找到一篇系统的介绍堆off-by-one利用的教程。在这篇文章中我列出了5种常见的堆上的off-by-one攻击方式，并且给出了测试DEMO，测试的环境均为x86。

## 0x01 背景知识

网络上关于Linux下堆管理机制的文章已经有很多了，这里不再详细的描述堆管理机制的细节，仅简单的列出一些要理解文章内容必须要掌握的基础知识点。

首先，目前的Linux使用的是基于ptmalloc的堆管理器。在ptmalloc中堆块被分为以下四种类型

### 1.fastbin

fastbin的范围处于16~64byte，使用单向链表来维护。每次从fastbin中分配堆块时，都会从尾部取出。fastbin块的inuse位永远是置于1的，并且享有最高的优先权，在分配和释放时总会最先考虑fastbin。

### 2.unsort bin

unsort bin在bins[]中仅占有一个位置，除了fastbin外的其他块被释放后都会进入到这里来作为一个缓冲，每当进行malloc时会把堆块从unsort bin中取出并放到对应的bins[]中。

### 3.small bin

small bin是指大于16byte且小于512byte的堆块，使用双向链表链接，不会有两个相邻的空的small bin块，因为一旦出现这种情况，相邻的块就会被合并成一个块。通常是在调用free函数时触发这一过程。需要注意的是在相邻空块合并时会调用unlink()宏来进行取下操作，但是调用malloc()时的取下操作却没有使用unlink宏。

4.large bin

超出large bin范围的即为large bin，large bin相比其他块而言具有一条额外的由fd\_nextsize和bk\_nextsize域组成的链表结构。



如图所示，其中size域低三位作为标志位，我们最需要记住的就是inuse位，这个位确定了前一个块是否处于使用状态。

是的，在ptmalloc中一个块是否使用是由下一个块进行记录的。

0x02 off-by-one的分类

off-by-one总共可以分为两种利用方式

chunk overlapping

off-by-one overwrite allocated

off-by-one overwrite freed

off-by-one null byte

unlink

off-by-one small bin

off-by-one large bin

这种划分的依据是基于利用的思路不同。

第一种利用的核心思路主要是为了进行chunk overlapping,而第二种的利用思路则是想要触发unlink。

0x03 达成漏洞利用的条件

off-by-one并不是全都可以达到利用的目的的。首先就要求堆必须以要求的size+0x4字节（x86）的大小进行分配。如果不满足这个条件那么就无法覆盖到inuse位了。这个是由于堆的字节对齐机制造成的，简单的说堆块是以8字节进行对齐的（x64为16字节）。如果malloc(1024)，那么实际会分配1024+8=1032字节，这一点很好理解。但是如果是malloc(1020)呢，1020+8=1028字节，而1028不满足8字节对齐，那么实际只会分配1020+4=1024字节，多出的4个字节由下一块的prev\_size提供空间。

而对于触发unlink的操作来说，还需要一个额外的附加条件。因为现在的unlink是有检验的，所以需要有一个指向堆上的指针才可以。

0x04 漏洞利用的效果

off-by-one能达到什么利用效果呢？这个是很关键的问题。根据分类来看可以实现两种效果

1.chunk overlapping

所谓的chunk overlapping是指，针对一个目标堆块。我们可以通过一些操作，使这个目标堆块被我们重新分配到某个我们控制的新的堆块中，这样就可以对目标堆块进行任意的读写了。

2.unlink

这种off-by-one造成的unlink的利用效果其实和溢出造成的unlink的利用效果是一致的。对于small bin可以使指向堆的指针ptr的值变为&ptr-0xc，这样再结合一系列的操作就可以达成几乎无限次的write-anything-anywhere了。

而large bin的unlink则可以实现一次任意地址写（write-anything-anywhere）。

0x05 漏洞利用的原理

chunk overlapping的原理在于ptmalloc的堆块验证机制的不完善，通过一些ptmalloc定义的宏就可以看出这一点。

inuse(): 仅通过下一块的inuse位来判定当前块是否使用.

prev\_chunk(): 如果前一个块为空，那么进行空块合并时，仅使用本块的prev\_size来寻找前块的头。

next\_chunk(): 仅通过本块头+本块大小的方式来寻找下一块的头

chunksize(): 仅通过本块的size确定本块的大小。

unlink的原理在于unlink宏在处理时会互写数据造成任意地址写。经过改进后的unlink宏增加了check，但是可以通过一个指向堆上的指针导致绕过情况。

0x06 达成漏洞利用的具体操作

off-by-one overwrite allocated

在这种情况下堆块布局是这样的



A是发生有off-by-one的堆块，其中B和C是allocated状态的块。而且C是我们的攻击目标块。

我们的目标是能够读写块C，那么就应该去构造出这样的内存布局。然后通过off-by-one去改写块B的size域（注意要保证inuse域的值1，否则会触发unlink导致crash）以实现把C块给整个包含进来。通过把B给free掉，然后再allocated一个大于B+C的块就可以返回B的地址，并且可以读写块C了。

具体的操作是：

- 1. 构成图示的内存布局
- 2. off-by-one改写B块的size域(增加大小以包含C，inuse位保持1)

3. free掉B块
4. malloc一个B+C大小的块
5. 通过返回的地址即可对C任意读写

注意，必须要把C块整个包含进来，否则free时会触发check，导致抛出错误。因为ptmalloc实现时的验证逻辑是当前块的下一块的inuse必须为1，否则在free时会触发异常，这一点本来是为了防止块被double free而做的限制，却给我们伪造堆块造成了障碍。

off-by-one overwrite freed

在这种情况下堆块布局依然是这样的



A是发生有off-by-one的堆块，其中B是free状态的块,C是allocated块。而且C是我们的攻击目标块。

我们的目标是能够读写块C，那么就应该去构造出这样的内存布局。然后通过off-by-one去改写块B的size域（注意要保证inuse域的值1） 以实现把C块给整个包含进来。但是这种情况下的B是free状态的,通过增大B块包含C块，然后再allocated一个B+C尺寸的堆块就可以返回B的地址，并且可以读写块C了。

具体的操作是：

1. 构成图示的内存布局
2. off-by-one改写B块的size域(增加大小以包含C，inuse位保持1)
3. malloc一个B+C大小的块
4. 通过返回的地址即可对C任意读写

off-by-one null byte

这种情况就与上面两种有所不同了，在这种情况下溢出的这个字节是一个'x00'字节。这种off-by-one可能是最为常见的，因为诸如：

```
buf=malloc(124);
if(strlen(str)==124)
{
    strcpy(buf,str);
}
```

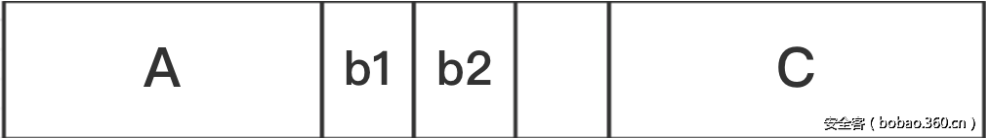
就会产生这种null byte off-by-one，即拷贝一个字符串到一个同样长的缓冲区时，并未考虑到NULL字节。

相比于前两种，这种利用方式就显得更复杂，而且对内存布局的要求也更高了。

首先内存布局需要三个块



其中A,B,C都是allocated块， A块发生了null byte off-by-one,覆盖了B块的inuse位，使B块伪造为空。然后在分配两个稍小的块b1、b2，根据ptmalloc的实现，这两个较小块（不能是fastbin）会分配在B块中。然后只要释放掉b1，再释放掉C，就会引发从原B块到C的合并。那么只要重新分配原B大小的chunk，就会重新得到b2。在这个例子中， b2是我们要进行读写的目标堆块。最后的堆块布局如下所示：



布局堆块结构如ABC所示

1. off-by-one覆盖B，目的是覆盖掉B的inuse位
2. free B
3. malloc b1,malloc b2
4. free C
5. free b1
6. malloc B
7. overlapping b2

这种利用方式成功的原因有两点:

通过prev\_chunk()宏查找前块时没有对size域进行验证

当B块的size域被伪造后，下一块的pre\_size域无法得到更新。

**off-by-one small bin**



这种方法是要触发unlink宏，因此需要一个指向堆上的指针来绕过fd和bk链表的check。

需要在A块上构造一个伪堆结构，然后覆盖B的pre\_size域和inuse域。这样当我们free B时，就会触发unlink宏导致指向堆上的指针ptr的值被改成&ptr-0xC(x64下为&ptr-0x18)。通过这个特点，我们可以覆写ptr指针，如果条件允许的话，几乎可以造成无限次的write-anything-anywhere。

1. 在A块中构造伪small bin结构，并且修改B块的prev\_size域和inuse域。
2. free B块
3. ptr指针被改为&ptr-0xC

**off-by-one large bin**

large bin通过unlink造成write-anything-anywhere的利用方法最早出现于Google的Project Zero项目的一篇文章中，具体链接是

<https://googleprojectzero.blogspot.fr/2014/08/the-poisoned-nul-byte-2014-edition.html>

在这篇文章中，提出了large bin检验仅仅是通过assert断言的形式来进行的，并不能真正的对漏洞进行有效的防护。但是经过我的测试发现，目前版本的ubuntu和CentOS已经均具备有检测large unlink的能力，如果发现存在指针被篡改的情况，则会抛出“corrupted double-linked list(not small)”的错误，之后翻阅了一下glibc中ptmalloc部分的实现代码却并没有发现有检测这部分的代码，猜测大概是后续版本中加入的。因为这种利用方式的意义已经不是很大，这里就不在详细列出步骤也不提供测试DEMO了。

**0x07 测试DEMO**

**1.off-by-one overwrite allocated**

```
int main(void)
{
    char buf[253]="";
    void *A,*B,*C;
    void *Overlapped;

    A=malloc(252);
    B=malloc(252);
    C=malloc(128);
    memset(buf,'a',252);
    buf[252]='x89'; // 把C块包含进来
    memcpy(A,buf,253); // A存在off-by-one漏洞

    free(B);
    Overlapped=malloc(500);
}
```

这段代码演示了通过off-by-one对C块实施了overlapping。通过返回的变量Overlapped就可以对C块进行任意的读写了。

## 2.off-by-one overwrite freed

```
int main(void)
{
    char buf[253]="";
    void *A,*B,*C;
    void *Overlapped;

    A=malloc(252);
    B=malloc(252);
    C=malloc(128);
    free(B);
    memset(buf,'a',252);
    buf[252]='x89';
    memcpy(A,buf,253); // A存在off-by-one漏洞

    Overlapped=malloc(380);
}
```

这个DEMO与上面的类似，同样可以overlapping后面的块C，导致可以对C进行任意读写。

## 3.off-by-one null byte

```

int main(void)
{
    void *A,*B,*C;
    void *B1,*B2;
    void *Overlapping;
    A=malloc(0x100);
    B=malloc(0x208);
    C=malloc(0x100);
    free(B);
    ((char *)A)[0x104]='x00';
    B1=malloc(0x100);
    B2=malloc(0x80);
    free(B1);
    free(C);
    malloc(0x200);
}

```

可以成功的对B2进行任意读写。

#### 4.off-by-one small bin

```

void *ptr;
int main(void)
{
    int prev_size,size,fd,bk;
    void *p1,*p2;
    char buf[253]="";

    p1=malloc(252);
    p2=malloc(252);

    ptr=p1;
    prev_size=0;
    size=249;
    fd=(int)(&ptr)-0xC;
    bk=(int)(&ptr)-0x8;

    memset(buf,'c',253);
    memcpy(buf,&prev_size,4);
    memcpy(buf+4,&size,4);
    memcpy(buf+8,&fd,4);
    memcpy(buf+12,&bk,4);
    size=248;
    memcpy(&buf[248],&size,4);
    buf[252]='x00';

    memcpy(p1,buf,253);

    free(p2);
}

```

这个DEMO中使用了一个指向堆上的指针ptr， ptr是全局变量处于bss段上。通过重复写ptr值即可实现write-everything-anywhere。



0x08 后记

这是本人第一次投稿原创文章，之前只是写写博客。文笔不好，错误也在所难免，希望大家包容下。

参考文档：

[CTF中的内存漏洞利用技巧 -清华大学网络与信息安全实验室](#)

[Google Project Zero Blog](#)

[glibc\\_malloc from github](#)

[Glibc Adventures: The Forgotten Chunks](#)

本文由安全客原创发布  
转载，请参考[转载声明](#)，注明出处：<https://www.anquanke.com/post/id/84752>  
安全客 - 有思想的安全新媒体

安全知识

👍 赞

❤ 收藏

 Ox9A82

分享到：

推荐阅读



[门罗币挖矿的恶意家族 HiddenMiner导致手机设备管理](#)  
[2018-03-29 16:00:46](#)



[一道pwn题带来的新思路 — 从 unsorted bin attack 到 large](#)  
[2018-03-29 15:00:03](#)



[Exploiting Jolokia Agent with Java EE Servers](#)  
[2018-03-29 14:00:23](#)



[8291端口告警事件简报](#)  
[2018-03-29 12:00:50](#)

发表评论

发表你的评论吧

昵称 大表哥

🔄 换一个

发表评论

评论列表

- 


[buffree](#) · 2018-01-25 12:56:30

3.off-by-one null byte 目前测试了一下，在比较新的glibc 下已经添加了新的防护，不能成功了，ub12还能成功，反正libc-2.19.so已经不能成功了，corrupted size vs. prev\_size: 0x0804b10

👍 回复
- 

我不是黑客 · 2018-01-24 12:53:57

((char \*)A)[0x104]='x00'; 这里面 确认没有 ‘，请确认一下 谢谢

👍 回复
- 

[buffree](#) · 2018-01-25 15:55:44

这4demo 非常好 除了第三个在高版本libc上失效以外，其他三个目前都能利用

👍 1 回复





Ox9A82

www.weibo.com/u/1828621423

文章 18 粉丝 7

+ 关注

TA的文章

Edge Type Confusion利用：从内存读写到控制流程

2018-02-26 16:03:48

Edge Type Confusion利用：从type confused到内存读写

2018-02-24 14:18:02

【漏洞分析】对Edge浏览器的js解析引擎Chakra漏洞CVE-2017-8548的分析

2017-11-27 15:08:33

【技术分享】IE浏览器漏洞综合利用技术：UAF利用技术的发展

2017-03-28 14:38:58

【技术分享】IE浏览器漏洞综合利用技术：堆喷射技术

2017-03-24 11:03:54



输入关键字搜索内容

相关文章

3月23日安全热点 – 亚特兰大IT系统被SamSam Ransomware袭击

3月21日安全热点 – Windows远程协助可帮助黑客窃取敏感文件

域攻击之精准定位特权用户

中奖名单公布 | 安全圈活动大盘点（内附干货大补丸）

中奖名单公布 | 安全客年末充电必备的12本好书

年终盘点 | 最全面的2017物联网安全事件盘点

利用域委派获取域管理权限

热门推荐



安全客  
有思想的安全新媒体

知

安全客

- 关于我们
- 加入我们
- 联系我们
- 用户协议

商务合作

- 合作内容
- 联系方式
- 友情链接

内容须知

- 投稿须知
- 转载须知

合作单位

国家互联网应急中心

国家信息安全漏洞库  
China National Vulnerability Database of Information Security