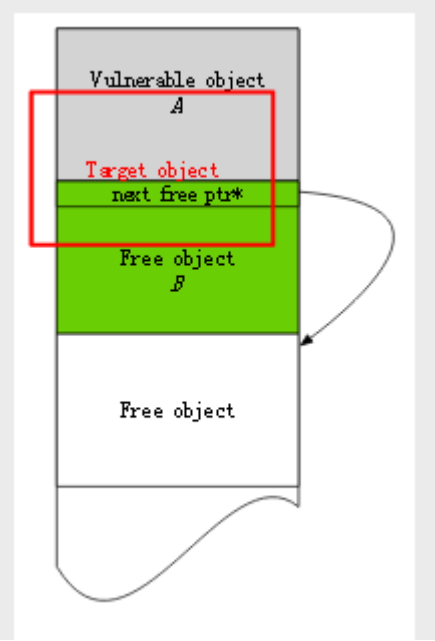
# CVE-2016-6187 Exploiting Linux kernel heap off-by-one 利用堆大小差一错误爆破 Linux 内核（中）

## Exploitation（漏洞利用）

利用该漏洞的一个优势是我们能控制目标对象的大小（args 对象大小由用户决定）。为了利用该漏洞，对象大小应该被设置为缓存大小中的一个（如 8，16，32，64，96 等）。本文不会详细介绍 SLUB 分配器如何工作（linux 默认内核内存分配器），我们所需要知道的是为了提高效率分配器会预先分配大量相同尺寸的对象。slab 是在缓存上包含同样大小对象主要的页。自由对象在偏移地址 0（默认）处有一个“next free”的指针指向 slab 的下一个自由对象。

我们的方案是在同样的一个 slab 放置我们脆弱的对象（A）邻近一个自由对象（B）,然后清除对象 B 的“next free”指针的 least-significant 字节。当两个新对象在同样的 slab 上被分配，最后的对象将会被分配在靠着“next free”指针的对象 A 和/或者对象 B。




上文情境（重叠 A 和 B 对象）是仅仅可能结局之一。目标对象“变化”的值是一个字节（0-255）并且最终目标对象的位置会依靠在原始的“next free”指针的值和对象大小。

假设目标对象会与对象 A 和 B 重叠，我们想要控制这些对象的内容。

在一个高等级，漏洞利用程序如下：

1. 在同样的 slab 上在邻近对象 B 处放置脆弱对象 A
2. 重写 B 中的“next free”指针的 least-significant 字节
3. 在同样的 slab 分配两个新对象：第一个对象将会被放在 B 处，并且第二个对象将会替代我的目标对象 C
4. 如果我们控制对象 A 和 B 的内容，我们可以强制对象 C 被分配在用户空间
5. 假设对象 C 有一个能从其他地方触发函数指针，在用户空间或者可能的一个 ROP 链（绕过 SMEP）中设置这个指针为我们的权利提升的 payload。

为了执行步骤 1-3，连续的对象分配能够取得使用一个标准的堆耗尽技术。

接下来，我们需要选择正确的对象的大小。对象比 128 字节更大（例如，申请缓存 256，512，1024 个字节）的话就不会在这里起作用。（原因在于我们只能重写一个字节，想清楚这点很关键）让我们假设起始的 slab 地址是 0x1000（标记 slab 的起始地址是与页大小一致的并且连续对象分配是相连的）。接下来的 C 程序列出了被给定对象大小的一个单页的分配：

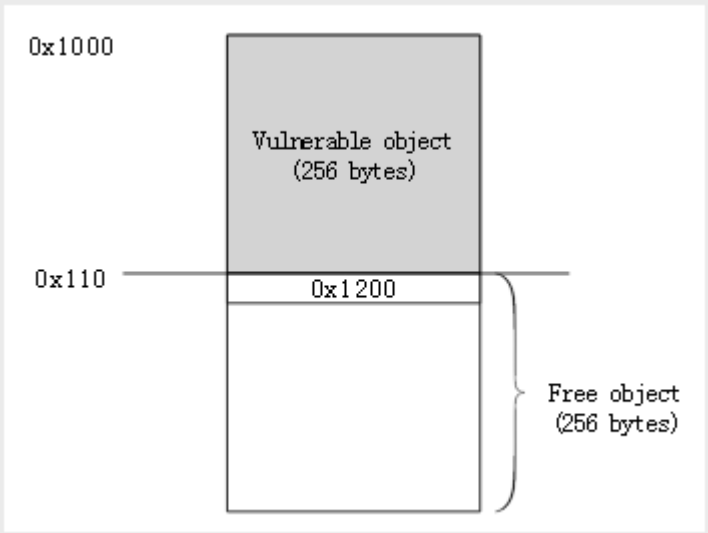
```
// page_align.c
#include

int main(int argc, char **argv) {
        int i;
        void *page_begin = 0x1000;

        for (i = 0; i < 0x1000; i += atoi(argv[1]))
                printf("%p\n", page_begin + i);

}
```

因为这些对象是 256 个字节（或>128 并<=256 字节），我们接下来匹配模式：

```
vnik@ubuntu:~$ ./align 256
0x1000
0x1100
0x1200
```

```
0x1300

0x1400

0x1500

0x1600

0x1700

0x1800

...
```

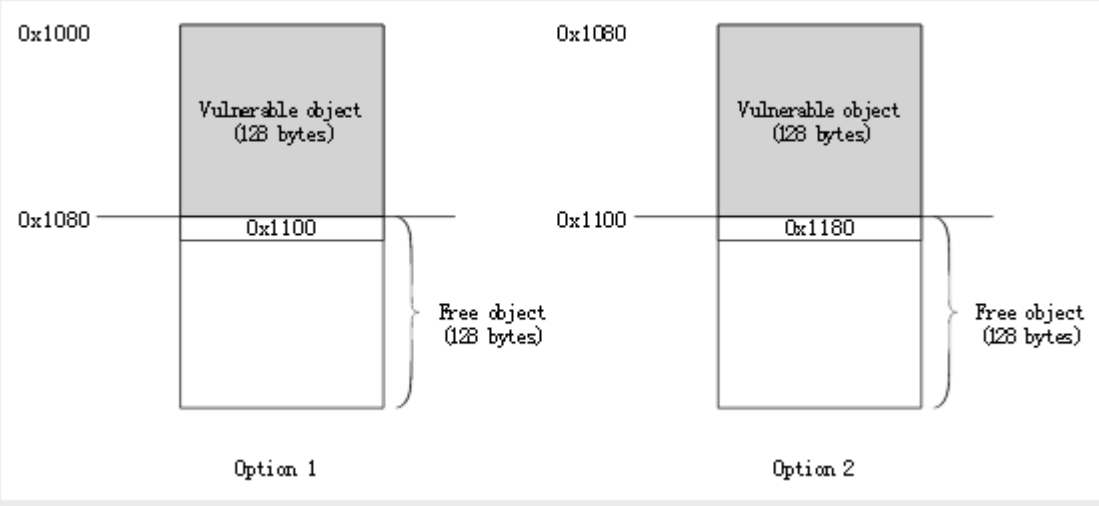
在 slab 所有配置的最低有效位为 0 并且重写邻近的自由对象的“next free”指针为 null 会没有效果：




因为 128 字节缓存，这里有两种可能选项：

```
vnik@ubuntu:~$ ./align 128

0x1000

0x1080

0x1100

0x1180

0x1200

0x1280

0x1300

0x1380

0x1400
```

...




第一个选项相似于之上的 256 字节例子（“next free”指针的最低有效位字节已经为 0）。第二个选项很有趣，因为重写“next free”指针的最低有效位字节会指向自由对象本身。分配一些 8 个字节大小的对象（A）到一些（确定的）用户空间内存地址，其次是目标对象（B）的分配会在用户空间用户控制的内存地址放置对象 B。这可能是在可靠性和易于利用两者间中的最好的选项。

1. 这有一个 50/50 成功机会。如果它是第一选项，没有崩溃,我们可以再试一次
2. 找到一个有一些用户空间地址的对象（首 8 个字节）将被放置在 `kmalloc-128` 缓存并不难。

尽管这是最好的方法，我决定将所有 96 字节对象和用 `msgsnd()` 堆耗尽/喷涂粘合起来。主要（仅有）的理由是因为已经发现了一个我想要使用的 96 字节的目标对象了。感谢 Thomas Pollet 帮助找到合适的堆对象并且在运行时用 gdb/python 自动化处理这个乏味的过程！

然而，显然使用 96 字节对象有下降趋势；一个主要的原因是利用的可靠性。一个耗尽 slab（如填充满 slab 部分）的主意就是 48 字节对象的标准 `msgget()` 技术（其他 48 字节被用来作为消息头）。这也将用作一个堆喷涂因为我们控制了 msg 对象的一半（48 字节）。我们也控制脆弱对象的内容（数据从用户空间被写到/proc/self/attr/current）。如果目标对象分配以便首 8 个字节被我们的数据所覆盖，然后漏洞利用将会成功。在另一方面，如果这 8 个字节用 msg 头（我们没有控制的）来覆盖，这会导致一个页面错误但是内核可能会被它

自身恢复。基于我的分析，这里有两个例子，在这“next free”指针会用先前分配的随机 msg 头覆盖。

这里是有一些技巧来提高漏洞利用的可靠性。

未完待续……

译者：rodster

校对：song

原文链接：**https://cyseclabs.com/blog/cve-2016-6187-heap-off-by-one-exploit**

原文作者：Vitaly Nikolenko