堆漏洞总结

总结

前面都是在介绍堆漏洞的类型和各种漏洞技巧用法，这些技巧其实还不够全面，如果你能详细了解 glibc 内存管理机制，你会发现很多情况下都可以构造漏洞利用技巧（不得不说 glibc 内存管理设计存在诸多缺陷），除此之外，现在我来介绍点你可能不知道的东西。

这里纯属我自己脑洞大开，有没有实际意义不知道，准不准确不知道；如果后续我发现新的东西会及时补充，发现有纰漏的地方会及时修改。

内存在没有分配之前是否可用

在第一个 malloc 调用之后（glibc 内存管理初始化后），即使它后面的内存没有被程序分配，你仍然是可以进行使用的，当然这有空间上的限制，如 arena 为 132k（理论上比这小很多），heap 看情况。

```
char* str= "1111111122222222333333334444444455555555566666666";

char* p1 = (char*) malloc(8);
strcpy(p1, str);

printf("p1 = %p, p1 = %s\n", p1, p1);
char* p2 = p1 + 8 ; // first 8 is p1 mem
printf("p2 = %p, p2 = %s\n", p2, p2);
return 0;
```

运行结果：

```
[sp00f@localhost heap]$ ./test0
p1 = 0x87f8008, p1 = 1111111122222222333333334444444455555555566666666
p2 = 0x87f8010, p2 = 2222222233333333444444445555555566666666
```

内存在没有分配之前是否可以被赋值

和上面情形一样，在第一个 malloc 调用之后（glibc 内存管理初始化后），即使它后面的内存没有被分配，它仍然可以被赋值。

```
char* str= "1111111122222222333333334444444455555555566666666";

char* p1 = (char*) malloc(8);
strcpy(p1, str);

printf("p1 = %s\n", p1);
char* p2 = malloc(8);
printf("p2 = %s\n", p2);
return 0;
```

运行结果

```
[sp00f@localhost heap]$ ./test1
p1 = 0x9095008, p1 = 1111111122222222333333334444444455555555566666666
p2 = 0x9095018, p2 = 333333334444!2225555555566666666
```

通过堆漏洞我们还可以干点别的什么事

只要存在堆溢出漏洞，我们能做的事情远不止前面介绍的（漏洞类型和漏洞技巧）那些，它可以让你覆盖任意分配的或未分配的内存；这样你就可以任意的构造伪 chunk 欺骗系统执行如合并，回收等（伪 chunk 可以在分配的内存中构建，也可以在没有分配的内存构建，glibc 内存管理根本察觉不了）；那堆地址随机化还有意义嘛？它确实值得思考，或许某些场景下它确实没有意义；只要我们通过如堆溢出控制了一片内存也许就可以为所欲为了（当然这是在不破坏程序正常运行的情况下；精心设计后程序不一定会被破坏，同时不影响我们的为所欲为，否则程序可能会出现各种各样的问题，大多情况下是崩溃）。