[介绍](#header-n2)  
 [基本示例](#header-n17)  
 [小总结](#header-n25)  
 [寻找堆分配函数](#header-n26)  
 [寻找危险函数](#header-n29)  
 [确定填充长度](#header-n32)

## 介绍

堆溢出是指程序向某个堆块中写入的字节数超过了堆块本身可用的字节数，因而导致了数据的溢出，并覆盖到了**物理相邻**的高地址的下一个堆块。

so，堆溢出漏洞发生的前提：

* 写入的数据写到了堆上
* 写入数据的大小没有被控制

堆溢出是一种特定的缓冲区溢出（还有栈溢出，bss段溢出等）。但是堆溢出并不会像栈溢出那样覆盖ret地址进而控制程序执行流。一般来说，利用堆溢出的策略：

1. 覆盖**物理相邻**的下一个chunk的内容
2. 利用堆中的机制（如unlink等）来实现任意地址写入（write-anything-anwhere），或控制堆块中的内容等效果，从而控制程序的执行流。

## 基本示例

例子：

#include <stdio.h>  
int main()   
{  
 char \*chunk1,\*chunk2;  
 chunk1 = malloc(24);  
 chunk2 = malloc(24);  
 puts("Get input:");  
 gets(chunk);  
 return 0;  
}

写入：AAAAAAAABBBBBBBBAAAAAAAA

pwndbg> x/32gx 0x56558150  
0x56558150: 0x0000000000000000 0x0000002100000000 <= chunk1  
0x56558160: 0x4141414141414141 0x4242424242424242  
0x56558170: 0x4141414141414141 0x0000002100000000 <= chunk2  
0x56558180: 0x0000000000000000 0x0000000000000000  
0x56558190: 0x0000000000000000 0x0000041100000000 <=other

如果输入字符串过长，会覆盖下一个chunk，比如：

pwndbg> x/32gx 0x56558150  
0x56558150: 0x0000000000000000 0x0000002100000000 <= chunk1  
0x56558160: 0x4141414141414141 0x4242424242424242  
0x56558170: 0x4141414141414141 0x4242424242424242 <= chunk2  
0x56558180: 0x0000000000000000 0x0000000000000000  
0x56558190: 0x0000000000000000 0x0000041100000000 <=other

## 小总结

比较重要的几个步骤

### 寻找堆分配函数

通常堆是通过调用glibc的malloc函数进行分配的，某些情况会使用calloc来分配。

malloc与calloc区别：calloc在分配后回自动清空。

calloc(0x20);
  
//等价于
  
ptr = malloc(0x20);
  
memset(ptr,0,0x20);

此外，还有一种是由realloc进行分配的，realloc函数可以同时具备malloc和free两个函数的功能。

#include<stdio.h>
  
int main()
  
{
  
 char \*chunk,\*chunk1;
  
 chunk = malloc(16);
  
 chunk1 = realloc(chunk,32);
  
 return 0;
  
}

realloc内部会根据不同的情况进行不同的操作：

* 当realloc(ptr,size)的size不等于ptr的size时：
  + 如果申请的size>原来的size：
    - 如果chunk与top chunk相邻，直接扩展这个chunk到新的size大小
    - 如果chunk与top chunk不相邻，相当于free(ptr)，malloc(new\_size)
  + 如果申请的size<原来的size：
    - 如果相差不足以容下一个最小的chunk（64位下为32字节，32位下为16字节），则保持不变
    - 如果相差可以容下一个最小chunk，则切割原chunk为两部分，free掉后一部分。
* 当realloc(otr,size)的size==0时，相当于free(ptr)
* 当realloc(otr,size)的size==ptr的size时，不进行任何操作

### 寻找危险函数

常见危险函数：

* INPUT：
  + gets：直接读取一行，忽略\x00
  + scanf
  + vscanf
* OUTPUT：
  + sprintf
* String:
  + strcpy：字符串复制，遇到\x00停止
  + strcat：字符串拼接，遇到\x00停止
  + bcopy

### 确定填充长度

这一部分主要是计算开始写入的地址与要覆盖的地址之间的距离。

一个常见的误区是malloc的参数等于实际分配堆块的大小，但实际上ptmalloc分配的大小是对其的，这个长度一般是机器字长的2倍。对于不大于2倍机器字长的请求，malloc会直接返回2倍的机器字长，也就是最小chunk。

比如64位系统malloc(0)是会返回16字节的块。