Hole in file analysis

宝德技术研究院 李磊

# 概述

本文通过一个例子验证文件洞是否占用磁盘的空间

# 文件洞的生成

以下是生成文件洞的测试代码hole\_creator.c：

1

2 #include <unistd.h>

3 #include <stdio.h>

4 #include <fcntl.h>

5

6 char buf1[] = "abcdefghij";

7 char buf2[] = "ABCDEFGHIJ";

8

9 int main(void)

10 {

11 int fd;

12 if((fd = open("file.hole",O\_CREAT|O\_RDWR, 0755)) < 0){

13 perror("creat error\n");

14 return -1;

15 }

16

17 if(write(fd,buf1,10) != 10){

18 perror("buf1 write error\n");

19 goto close\_fd;

20 }

21

22 if(lseek(fd,0x8000,SEEK\_SET) == -1){

23 perror("lseek error\n");

24 goto close\_fd;

25 }

26

27 if(write(fd,buf2,10) != 10){

28 perror("write buf2 error\n");

29 goto close\_fd;

30 }

31

32 close(fd);

33 return 0;

34

35 close\_fd:

36 close(fd);

37 return -1;

38 }

执行hole\_creator后就会生成一个file.hole的文件

[root@localhost ~]# ll -i

total 108

1509508 -rwxr-xr-x 1 root root 4827 Jul 16 19:44 cap.sh

1573000 drwxr-xr-x 2 root root 4096 Jun 19 23:36 Desktop

1509494 -rwxr-xr-x 1 root root 32778 Sep 29 20:53 file.hole

1507350 -rwxr-xr-x 1 root root 7533 Sep 29 20:51 hole\_creator

1509507 -rw-r--r-- 1 root root 698 Sep 29 20:51 hole\_creator.c

[root@localhost ~]# hexdump -C file.hole

00000000 61 62 63 64 65 66 67 68 69 6a 00 00 00 00 00 00 |abcdefghij......|

00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

\*

00001000 41 42 43 44 45 46 47 48 49 4a |ABCDEFGHIJ|

0000100a

看出空洞文件生成成功。

# 分析文件inode

接下来，看看下file.hole所在磁盘的设备名：

[root@localhost ~]# df

Filesystem 1K-blocks Used Available Use% Mounted on

/dev/sda2 18012336 3656936 13425640 22% /

/dev/sda1 295561 48095 232206 18% /boot

tmpfs 512936 0 512936 0% /dev/shm

我的根目录设备是/dev/sda2

继续查看/dev/sda2的文件系统信息

[root@localhost ~]# dumpe2fs /dev/sda2 | less

Filesystem volume name: /

Last mounted on: <not available>

Filesystem UUID: 3d9e0a81-e3d8-49f5-9b98-926428bb8e06

Filesystem magic number: 0xEF53

Filesystem revision #: 1 (dynamic)

Filesystem features: has\_journal ext\_attr resize\_inode dir\_index filetype needs\_recovery sparse\_super large\_file

Default mount options: user\_xattr acl

Filesystem state: clean

Errors behavior: Continue

Filesystem OS type: Linux

Inode count: 4653056

Block count: 4648809

Reserved block count: 232440

Free blocks: 3573371

Free inodes: 4531451

First block: 0

Block size: 4096

Fragment size: 4096

Reserved GDT blocks: 1022

Blocks per group: 32768

Fragments per group: 32768

Inodes per group: 32768

Inode blocks per group: 1024

Filesystem created: Thu Jun 19 22:21:36 2014

Last mount time: Mon Sep 29 18:22:28 2014

Last write time: Mon Sep 29 18:22:28 2014

Mount count: 367

Maximum mount count: -1

Last checked: Thu Jun 19 22:21:36 2014

Check interval: 0 (<none>)

Reserved blocks uid: 0 (user root)

Reserved blocks gid: 0 (group root)

First inode: 11

Inode size: 128

Journal inode: 8

Default directory hash: tea

Directory Hash Seed: 0780e00b-024f-491e-9f97-5e7bd01fadc1

Journal backup: inode blocks

Journal size: 128M

...

Group 46: (Blocks 1507328-1540095)

Block bitmap at 1507328 (+0), Inode bitmap at 1507329 (+1)

Inode table at 1507330-1508353 (+2)

6842 free blocks, 30570 free inodes, 923 directories

...

红色部分为最重要的信息 。

file.hole的inode为1509494。可以计算出它在（1509494/32768）第46个块组中，块组中inode下标2166。inode table的地址是1507330 \* 4096 + (2166 - 1 ) \* 128= 0x170045A80。

我们来看一下file.hole inode的信息。

[root@localhost ~]# hexdump -C -s 0x170045A80 /dev/sda2 | less

170045a80 ed 81 00 00 0a 80 00 00 21 2c 29 54 fe 2b 29 54 |........!,)T.+)T|

170045a90 fe 2b 29 54 00 00 00 00 00 00 01 00 10 00 00 00 |.+)T............|

170045aa0 00 00 00 00 00 00 00 00 00 60 17 00 00 00 00 00 |.........`......|

170045ab0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

下面是内核中ext3文件系统inode的部分成员，磁盘上inode的信息就是这些成员的排列。

struct ext3\_inode {

\_\_le16 i\_mode; /\* File mode \*/

\_\_le16 i\_uid; /\* Low 16 bits of Owner Uid \*/

\_\_le32 i\_size; /\* Size in bytes \*/

\_\_le32 i\_atime; /\* Access time \*/

\_\_le32 i\_ctime; /\* Creation time \*/

\_\_le32 i\_mtime; /\* Modification time \*/

\_\_le32 i\_dtime; /\* Deletion Time \*/

\_\_le16 i\_gid; /\* Low 16 bits of Group Id \*/

\_\_le16 i\_links\_count; /\* Links count \*/

\_\_le32 i\_blocks; /\* Blocks count \*/

\_\_le32 i\_flags; /\* File flags \*/

union {

struct {

\_\_u32 l\_i\_reserved1;

} linux1;

struct {

\_\_u32 h\_i\_translator;

} hurd1;

struct {

\_\_u32 m\_i\_reserved1;

} masix1;

} osd1; /\* OS dependent 1 \*/

\_\_le32 i\_block[EXT3\_N\_BLOCKS];/\* Pointers to blocks \*/

可以看出（这里需要大小端转换）：

●i\_mode = 0x81ed，说明这个文件是一个普通文件，权限是755。具体的i\_mode

表示可以参考ext2 filesystem layout analysis.docx。

●i\_size=0x800a

●i\_blocks=0x10, 即16\*512 = 8K,正好2个块

●i\_mtime=0x54292bfe =1411984382。

●i\_block[0] = 0x176000

●i\_block[8] = 0x176001

查看时间戳：

[root@localhost ~]# date -d "@1411984382"

Mon Sep 29 20:53:02 VLAST 2014

确定了这个文件是hole.file。i\_size=0x800a，理论上占用9个 block的大小（本例中，一个block 4096byte）。而i\_blocks =16，表示占用磁盘2个block（i\_blocks以512byte为单位）。

我们接着看看i\_block[0]和i\_block[8]中的内容：

i\_block[0]的地址：0x176000 \* 0x1000=176000000

i\_block[8]的地址：0x176001 \* 0x1000=176001000

176000000 61 62 63 64 65 66 67 68 69 6a 00 00 00 00 00 00 |abcdefghij......|

176000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

\*

176001000 41 42 43 44 45 46 47 48 49 4a 00 00 00 00 00 00 |ABCDEFGHIJ......|

176001010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|

# 结论

从上面是分析可以得出

1. 文件空洞并不在磁盘上面占用具体的块，仅仅是在内存中将空洞部分以0x0填充。但是在inode中记录文件大小时却包含了空洞部分的大小。
2. 空洞前后的数据分别保存在不同的数据块。一定程度上也造成了磁盘空间的浪费。本例中，本来只有20字节的数据占用了2个block,也就是8096byte。
3. 虽然空洞不占用磁盘空间，但是在inode中还是预留了block给空洞。在本例中file.hole的inode预留了block[1]-block[7]给空洞。如果有进程在空洞区域内写数据。空洞后的数据也不需要移动，这恰恰是文件系统设计者的睿智所在。