



author Chao Yu <chao@kernel.org> 2025-03-03 11:47:38 +0800
committer Greg Kroah-Hartman <gregkh@linuxfoundation.org> 2025-05-02 07:40:49 +0200
commit [a67e1bf03c609a751d1740a1789af25e599966fa](#) (patch)
tree [139d9310347dcc1526a02d7ca8339de19416a748](#)
parent [12e4dc754333ec6e92b09ce09023c6fc270beb5f](#) (diff)
download [linux-a67e1bf03c609a751d1740a1789af25e599966fa.tar.gz](#)

diff options

context:
space:
mode:

f2fs: fix to avoid out-of-bounds access in f2fs_truncate_inode_blocks()

[Upstream commit e6494977bd4a83862118a05f57a8df40256951c0]

syzbot reports an UBSAN issue as below:

-----[cut here]-----

UBSAN: array-index-out-of-bounds in fs/f2fs/node.h:381:10
index 18446744073709550692 is out of range for type '__le32[5]' (aka 'unsigned int[5]')
CPU: 0 UID: 0 PID: 5318 Comm: syz.0.0 Not tainted 6.14.0-rc3-syzkaller-00060-g6537cfb395f3 #0
Call Trace:

```
<TASK>  
__dump_stack lib/dump_stack.c:94 [inline]  
dump_stack_lvl+0x241/0x360 lib/dump_stack.c:120  
ubsan_epilogue lib/ubsan.c:231 [inline]  
__ubsan_handle_out_of_bounds+0x121/0x150 lib/ubsan.c:429  
get_nid fs/f2fs/node.h:381 [inline]  
f2fs_truncate_inode_blocks+0xa5e/0xf60 fs/f2fs/node.c:1181  
f2fs_do_truncate_blocks+0x782/0x1030 fs/f2fs/file.c:808  
f2fs_truncate_blocks+0x10d/0x300 fs/f2fs/file.c:836  
f2fs_truncate+0x417/0x720 fs/f2fs/file.c:886  
f2fs_file_write_iter+0x1bdb/0x2550 fs/f2fs/file.c:5093  
aio_write+0x56b/0x7c0 fs/aio.c:1633  
io_submit_one+0x8a7/0x18a0 fs/aio.c:2052  
__do_sys_io_submit fs/aio.c:2111 [inline]  
__se_sys_io_submit+0x171/0x2e0 fs/aio.c:2081  
do_syscall_x64 arch/x86/entry/common.c:52 [inline]  
do_syscall_64+0xf3/0x230 arch/x86/entry/common.c:83  
entry_SYSCALL_64_after_hwframe+0x77/0x7f  
RIP: 0033:0x7f238798cde9
```

index 18446744073709550692 (decimal, unsigned long long)
= 0xfffffffffffffc64 (hexadecimal, unsigned long long)
= -924 (decimal, long long)

In f2fs_truncate_inode_blocks(), UBSAN detects that get_nid() tries to access .i_nid[-924], it means both offset[0] and level should zero.

The possible case should be in f2fs_do_truncate_blocks(), we try to truncate inode size to zero, however, dn ofs_in_node is zero and dn.node_page is not an inode page, so it fails to truncate inode page, and then pass zeroed free_from to f2fs_truncate_inode_blocks(), result in this issue.

```
if (dn.ofs_in_node || IS_INODE(dn.node_page)) {  
    f2fs_truncate_data_blocks_range(&dn, count);  
    free_from += count;  
}
```

```
}
```

I guess the reason why `dn.node_page` is not an inode page could be: there are multiple nat entries share the same node block address, once the node block address was reused, `f2fs_get_node_page()` may load a non-inode block.

Let's add a sanity check for such condition to avoid out-of-bounds access issue.

Reported-by: syzbot+6653f10281a1badc749e@syzkaller.appspotmail.com

Closes: <https://lore.kernel.org/all/66fdcdf3.050a0220.40bef.0025.GAE@google.com>

Signed-off-by: Chao Yu <chao@kernel.org>

Signed-off-by: Jaegeuk Kim <jaegeuk@kernel.org>

Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

```
-rw-r--r-- fs/f2fs/node.c 9
```

1 files changed, 8 insertions, 1 deletions

diff --git a/fs/f2fs/node.c b/fs/f2fs/node.c

index 348ad1d6199ff9..57baaba17174d0 100644

--- a/fs/f2fs/node.c

+++ b/fs/f2fs/node.c

@@ -1047,7 +1047,14 @@ int f2fs_truncate_inode_blocks(struct inode *inode, pgoff_t from)

trace_f2fs_truncate_inode_blocks_enter(inode, from);

level = get_node_path(inode, from, offset, noffset);

- if (level < 0) {

+ if (level <= 0) {

+ if (!level) {

+ level = -EFSCORRUPTED;

+ f2fs_err(sbi, "%s: inode ino=%lx has corrupted node block, from:%lu addr:%u",

+ __func__, inode->i_ino,

+ from, ADDRS_PER_INODE(inode));

+ set_sbi_flag(sbi, SBI_NEED_FSCK);

+ }

trace_f2fs_truncate_inode_blocks_exit(inode, level);

return level;

}