author	Filipe Manana <fdmanana@suse.com>	2022-11-01 16:15:38 +0000
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2022-11-10 18:14:19 +0100
commit	61e06128113711df0534c404fb6bb528eb7d2332 (patch)	
tree	3f8ffe4d046a2a75bb59243f5c49c1b5459ad3b4	
parent	a52e24c7fcc3c5ce3588a14e3663c00868d36623 (diff)	
download	linux-61e06128113711df0534c404fb6bb528eb7d2332.tar.gz	

btrfs: fix inode list leak during backref walking at find_parent_nodes()

```
[ Upstream commit 92876eec382a0f19f33d09d2c939e9ca49038ae5 ]

During backref walking, at find_parent_nodes(), if we are dealing with a
data extent and we get an error while resolving the indirect backrefs, at
resolve_indirect_refs(), or in the while loop that iterates over the refs
in the direct refs rbtree, we end up leaking the inode lists attached to
the direct refs we have in the direct refs rbtree that were not yet added
to the refs ulist passed as argument to find_parent_nodes(). Since they
were not yet added to the refs ulist and prelim_release() does not free
the lists, on error the caller can only free the lists attached to the
refs that were added to the refs ulist, all the remaining refs get their
inode lists never freed, therefore leaking their memory.

Fix this by having prelim_release() always free any attached inode list
to each ref found in the rbtree, and have find_parent_nodes() set the
ref's inode list to NULL once it transfers ownership of the inode list
to a ref added to the refs ulist passed to find_parent_nodes().

Fixes: 86d5f9944252 ("btrfs: convert prelimary reference tracking to use rbtrees")
Signed-off-by: Filipe Manana <fdmanana@suse.com>
Signed-off-by: David Sterba <dsterba@suse.com>
Signed-off-by: Sasha Levin <sashal@kernel.org>
```

Diffstat

-rw-r--r-- fs/btrfs/backref.c 18

1 files changed, 17 insertions, 1 deletions

```
diff --git a/fs/btrfs/backref.c b/fs/btrfs/backref.c
index 70c1c15266d67c..6942707f8b034b 100644
--- a/fs/btrfs/backref.c
+++ b/fs/btrfs/backref.c
@@ -288,8 +288,10 @@ static void prelim_release(struct preftree *preftree)
        struct prelim_ref *ref, *next_ref;

        rbtree_postorder_for_each_entry_safe(ref, next_ref,
-                                            &preftree->root.rb_root, rbnode)
+                                            &preftree->root.rb_root, rbnode) {
+               free_inode_elem_list(ref->inode_list);
                free_pref(ref);
+       }
```

```
        preftree->root = RB_ROOT_CACHED;
        preftree->count = 0;
@@ -1388,6 +1390,12 @@ again:
                                if (ret < 0)
                                        goto out;
                                ref->inode_list = eie;
+                               /*
+                                * We transferred the list ownership to the ref,
+                                * so set to NULL to avoid a double free in case
+                                * an error happens after this.
+                                */
+                               eie = NULL;
                        }
                        ret = ulist_add_merge_ptr(refs, ref->parent,
                                                  ref->inode_list,
@@ -1413,6 +1421,14 @@ again:
                                eie->next = ref->inode_list;
                        }
                        eie = NULL;
+                       /*
+                        * We have transferred the inode list ownership from
+                        * this ref to the ref we added to the 'refs' ulist.
+                        * So set this ref's inode list to NULL to avoid
+                        * use-after-free when our caller uses it or double
+                        * frees in case an error happens before we return.
+                        */
+                       ref->inode_list = NULL;
                }
                cond_resched();
        }
```