



Products

Services

Publications

Resources

What's new

[Follow @Openwall on Twitter for new release announcements and other news](#)

[<prev] [next>] [thread-next>] [day] [month] [year] [list]

Message-ID: <69b791e8-91d6-47cd-ad81-827d13cc0b65@frasunek.com>
 Date: Thu, 16 Apr 2026 07:49:45 +0200
 From: Przemyslaw Frasunek <przemyslaw@...sunek.com>
 To: oss-security@...ts.openwall.com
 Subject: UAF in rsync 3.4.1 and below

VULNERABILITY REPORT

rsync: Use-After-Free via qsort Out-of-Bounds in receive_xattr()

Reporter: Przemyslaw Frasunek <przemyslaw@...sunek.com>
 Date: 2026-04-16
 Affected: rsync 3.0.1 through 3.4.1 (all versions with xattr abbreviation)
 Current development head (3.4.2dev, commit b905ab23) also affected.
 File: xattrs.c, function receive_xattr(), line 864
 Severity: Medium / High
 Attack vector: Network (malicious rsync sender)

1. SUMMARY

The receive_xattr() function in rsync uses a wire-supplied count value as the length argument to qsort() instead of the actual number of items stored in the array. When xattr name filtering causes some items to be discarded, the count of items passed to qsort exceeds the number of valid items. The excess array positions contain stale data from a previously processed file's xattr list. After qsort reorders these stale entries into the stored positions, they are persisted in the global xattr list (rsync_xal_l) with dangling pointers to heap allocations that are subsequently freed and reallocated.

This creates use-after-free conditions in the receiver process: stale datum pointers are dereferenced in rsync_xal_set() for checksum comparison and in rcv_xattr_request() for name copying, and freed again in both functions, leading to double-free or free-of-allocated-memory scenarios.

2. AFFECTED CONFIGURATIONS

The bug is triggered when qsort runs over the stale array elements. The conditions depend on the operating system:

Linux:

The need_sort flag defaults to 0 and is only set to 1 when a non-user namespace xattr (e.g. system.*, security.*) is received and prefixed with the rsync namespace prefix (RSYNC_PREFIX = "user.rsync."). This prefixing only occurs when am_root <= 0, specifically:

- rsync receiver running with --fake-super (am_root = -1): VULNERABLE. Non-user namespace xattrs from the sender are prefixed and need_sort is set. This is a common configuration for backup mirrors and rsyncd deployments that operate without root privileges.
- rsync receiver running as non-root (am_root = 0) with xattr filter (--filter='x ...') configured: VULNERABLE. Non-user xattrs that pass the filter are prefixed and need_sort is set.
- rsync receiver running as root without --fake-super: NOT VULNERABLE on Linux. Non-user xattrs are accepted without prefixing or sorting.
- rsync receiver running as non-root without xattr filter: NOT VULNERABLE. Non-user xattrs are silently discarded without setting need_sort.

FreeBSD, macOS, and other non-Linux platforms:

The need_sort flag defaults to 1 unconditionally (xattrs.c line 771).

The bug is triggerable whenever `count > 1`. All configurations running with `-X/--xattrs` are VULNERABLE.

3. ROOT CAUSE

In `receive_xattr()` (`xattrs.c`), the wire-supplied `xattr` count for each file is read at line 786:

```
if ((count = read_varint(f)) != 0) {
    (void)EXPAND_ITEM_LIST(&temp_xattr, rsync_xa, count);
    temp_xattr.count = 0;
}
```

The `EXPAND_ITEM_LIST` call ensures the `temp_xattr.items` array has room for count items. However, the loop that follows (lines 791-861) only adds items that pass namespace filtering and `xattr` filter checks. Each accepted item increments `temp_xattr.count` via `EXPAND_ITEM_LIST(..., 1)`. Items that are filtered out are freed and skipped via `continue`.

When some items are filtered, `temp_xattr.count < count`. The array positions from `temp_xattr.count` to `count-1` are not cleared. Because `temp_xattr` is a static variable reused across files, these positions contain stale `rsync_xa` structs from the most recent file whose `receive_xattr()` populated those indices.

At line 863-864:

```
if (need_sort && count > 1)
    qsort(temp_xattr.items, count, sizeof (rsync_xa), rsync_xal_compare_names);
```

`qsort` operates on `count` items, including the stale entries. Since `qsort` sorts by `xattr` name (`rsync_xal_compare_names`), and the stale entries contain valid name pointers at this stage, the stale entries can be sorted into the first `temp_xattr.count` positions of the array.

At line 866, `rsync_xal_store(&temp_xattr)` copies exactly `temp_xattr.count` items into the global `rsync_xal_l` list. If stale entries were sorted into positions `0..temp_xattr.count-1`, they are now permanently stored as the current file's `xattr` data, with `datum` and `name` pointers pointing into another file's `xattr` allocations.

4. EXPLOITATION

The stale `rsync_xa` structs stored in `rsync_xal_l` reference `datum` buffers allocated during a prior file's `receive_xattr()`. These buffers are freed when `recv_xattr_request()` processes that prior file's abbreviated `xattr` requests (line 753: `free(old_datum)`), creating dangling pointers in the current file's stored `xattr` list.

The dangling pointers are subsequently dereferenced in two locations:

a) `recv_xattr_request()` for the current file (if the generator requests the stale items):

```
old_datum = rxa->datum; // dangling
rxs->datum = new_array(char, rxs->datum_len + rxs->name_len);
memcpy(name, rxs->name, rxs->name_len); // read-after-free
free(old_datum); // double-free / UAF free
```

b) `rsync_xal_set()` for the current file:

```
if (XATTR_ABBREV(rxsas[i])) { // struct field check
    ptr = get_xattr_data(fnamecmp, name, &len, 1); // read-after-free
    if (memcmp(sum, rxsas[i].datum + 1, ...) != 0) // read-after-free
        ...
    free(rxsas[i].datum); // double-free / UAF free
```

4.1 Stock receiver exploitation constraints

Stock `rsync` does not write to freed memory after `free(rxsas[i].datum)` in `rsync_xal_set()`. Without a write-after-free primitive, redirecting `tcache` allocations to arbitrary addresses is not directly achievable.

The following primitives ARE available on stock rsync:

- Read-after-free: stale datum pointers are dereferenced for checksum comparison (memcmp against heap contents) and xattr name reads (memcpy from freed memory for name strings past tcache metadata offset).
- Free-of-allocated-memory: stale pointers that were re-allocated by another file's `recv_xattr_request` are freed again, bypassing glibc tcache key detection (the key field was overwritten by application data during re-allocation). This corrupts heap state by placing an in-use allocation on the tcache free list.
- Information disclosure: if `rsync_xal_set` processes a stale item via the non-abbreviated path (`lsetxattr`), tcache metadata (safe-linked pointers, tcache key) is written as xattr values on destination files. This leaks heap layout information to the filesystem.
- Denial of service: the double-free / heap corruption reliably crashes the receiver process.

5. TRIGGER CONDITIONS

For a malicious sender to trigger the vulnerability:

- a) The victim must run rsync with `-X (--xattrs)` to enable xattr transfer.
- b) On Linux, the victim must also use `--fake-super`, or have an xattr filter configured that passes non-user namespace xattrs. On FreeBSD/macOS, no additional flags are required.
- c) The sender must include at least one non-user namespace xattr in the triggering file's xattr list (Linux only, to set `need_sort=1`).
- d) The sender must set the wire count higher than the number of xattrs that will pass filtering, and arrange for a prior file in the transfer to populate the stale array positions with exploitable xattr structs.

The sender fully controls conditions (c) and (d) through the rsync protocol. Conditions (a) and (b) depend on the victim's rsync invocation.

6. SUGGESTED FIX

Replace `count` with `temp_xattr.count` in the `qsort` call at `xattrs.c` line 864:

Before:

```
qsort(temp_xattr.items, count, sizeof (rsync_xa), rsync_xal_compare_names);
```

After:

```
qsort(temp_xattr.items, temp_xattr.count, sizeof (rsync_xa),  
rsync_xal_compare_names);
```

This ensures `qsort` only operates on actually populated items, preventing stale data from being sorted into the stored xattr list.

7. TIMELINE

- 2008-03-07 Bug introduced in commit `d724dd186` (rsync 3.0.1pre1).
The commit added `qsort` to `receive_xattr()` for sorting xattrs after namespace prefix munging in `--fake-super` mode.
- 2026-04-16 This report.

8. REFERENCES

Source: <https://github.com/RsyncProject/rsync>

Buggy commit: `d724dd186` ("Fixed the interaction of `--fake-super` with `--link-dest` & `--xattrs`. Fixed the munging of non-user namespace xattrs w/`--fake-super`. Fixed the sorting of received xattrs when name-munging occurs.")

Powered by [blists](#) - [more mailing lists](#)

Please check out the [Open Source Software Security Wiki](#), which is counterpart to this [mailing list](#).

Confused about [mailing lists](#) and their use? [Read about mailing lists on Wikipedia](#) and check out these [guidelines on proper formatting of your messages](#).