

h3llh0und

home x github linkedin

SEP 24, 2026 · 13 MIN READ

CVE-2026-93353: a volume guard enforced one call site at a time

copyparty's xvol option keeps the file server from following symlinks out of a volume. Its SFTP front end runs that check inside one helper — which four handlers call and three do not — so mkdir, rmdir and chattr write outside the volume. The maintainer closed the report without a word and shipped no fix; the identifier came from VulnCheck. Present through 1.20.24.

> ***Disclaimer.*** *This is personal security research, carried out on my own equipment against a public open-source project. It has no connection to any current or former employer, and no client, employer or production environment was involved at any point. Every operation in it ran against a copyparty process on my own machine, serving a throwaway directory created for the test. The issue was reported privately through the project's own advisory process; the maintainer closed it without comment and shipped no fix. This write-up is published as the public reference the identifier*

> *requires, and unlike the others on this blog it describes a defect that is present in the current release.*

copyparty is a portable file server: one Python file you point at a directory, and it serves that directory over HTTP, WebDAV, FTP, SFTP and TFTP at once. When you expose a directory, you can also constrain it. The `xvol` volflag is one such constraint, and its help text states the whole intent:

```
--xvol   never follow symlinks leaving the volume root, unless the link i
         into another volume where the user also has access
```

A volume is a directory you have shared. A symlink inside it can point anywhere on disk. `xvol` is the promise that the server will not walk such a link out of the volume and hand back whatever it finds — the operator's home directory, another tenant's volume, `/etc`.

That promise is kept by a single function, `chk_ap` in `authsrv.py:954`. It takes a resolved absolute path and returns the volume it legitimately belongs to, or `None` if the path escaped. The interesting question is not what `chk_ap` does. It is where it is called from — because in copyparty it is called *per operation*, at each place a virtual path is turned into a real one, rather than once at a gateway every request must pass.

```
$ grep -rn 'chk_ap' copyparty/
authsrv.py:954   def chk_ap(...)           <- the check itself
ftpd.py:198          vfs.chk_ap(ap)
ftpd.py:331          vfs.chk_ap(ap, st)
httpcli.py:891       vn.chk_ap(ap)         <- HTTP, one chokepoint
sftpd.py:348         vn.chk_ap(ap)         <- SFTP, inside one helper
tftpd.py             (none)
```

Read that list as what it is: an inventory of every place someone remembered to enforce the boundary. HTTP calls it once, at a point every request funnels

through. FTP calls it on each operation. SFTP calls it in exactly one helper. TFTP does not call it at all.

A boundary maintained as a list of call sites is a boundary you can forget to extend, and the rest of this is about the three SFTP handlers where it was.

The helper that does the whole job

Inside the SFTP front end the check lives in `v2a` (`sftpd.py:315`). Given a virtual path and a set of requested rights, it resolves the path, and when the volume carries `xvol` or `xdev` it runs `chk_ap` and then re-reads the destination volume's own permission bits for the user:

```
def v2a(self, vpath, r=False, w=False, m=False, d=False):
    ...
    vn, rem = self.hub.asrv.vfs.get(vpath, self.uname, r, w, m, d)
    ...
    if "xdev" in vn.flags or "xvol" in vn.flags:
        ap = vn.canonical(rem)
        avn = vn.chk_ap(ap)
        if not avn:
            raise OSError(errno.EPERM, "permission denied in [/%s]" % (vp
        cr, cw, cm, cd, _, _, _, _, _ = avn.uaxs[self.uname]
        if r and not cr or w and not cw or m and not cm or d and not cd:
            raise OSError(errno.EPERM, "permission denied in [/%s]" % (vp
    else:
        ap = vn.canonical(rem, False)
    ...
    return ap, vn, rem
```

This is the correct, complete implementation. It canonicalises the path — resolving the symlink — and only then asks `chk_ap` whether the resolved location is still inside a volume the user may reach. Any SFTP handler that routes through `v2a` is safe.

Four of them do. `_open` (read and write of files), `_list_folder`, `_stat` and `_remove` all begin by calling `self.v2a(...)` with the rights they need, and inherit the guard for free.

Three that resolve the path themselves

The other three build the absolute path by hand and never reach `v2a`:

handler	line	path resolution	`chk_ap`?
`_mkdir`	`:667`	`vfs.get(...)` then `vn.canonical(rem, False)`	**no**
`_rmdir`	`:691`	`vfs.get(...)` then `os.path.join(vn.realpath, rem)`	**no**
`_chattr`	`:713`	`vfs.get(...)` then `os.path.join(vn.realpath, rem)`	**no**

`_mkdir` is representative:

```
def _mkdir(self, vp, attr):
    vn, rem = self.asrv.vfs.get(vp, self.uname, False, True)
    ap = vn.canonical(rem, False)
    bos.makedirs(ap, vf=vn.flags)
    if attr is not None:
        paramiko.SFTPServer.set_file_attr(ap, attr)
    return SFTP_OK
```

`vfs.get` answers a permission question about the *virtual* path — may this user write here — and returns the volume plus the remainder. Then `vn.canonical(rem, False)` builds the absolute path with the second argument `False`, which is copyparty's own flag for *do not resolve*. So the symlink in the path is never followed at check time and never checked after it is followed at write time. `chk_ap` is not called with `False` or with `True`; it is not

called. `_rmdir` and `_chattr` are the same shape with `os.path.join` in place of `canonical`.

The consequence is direct. Place a symlink inside a volume that points outside it, and these three handlers will act on the target:

```
lab/vol/                    <- the served volume, xvol on
lab/vol/esc -> lab/outside  <- a symlink that leaves it
lab/outside/secret.txt      <- 31 bytes, mode 644
```

```
mkdir   /vol/esc/pwned        -> OK   (directory created in lab/outside)
rmdir   /vol/esc/pwned        -> OK
chattr  /vol/esc/secret.txt   -> OK   (mode and size, below)
```

The precondition is the one copyparty's own advisories name for this class: such a symlink has to already exist, and copyparty does not create it. It is the ordinary situation of a volume that contains a link an operator put there, or that an account with upload rights was able to place by other means.

`chattr` is not a metadata write

`_chattr` deserves its own paragraph, because "change attributes on a path outside the volume" sounds like the mildest of the three and is the sharpest.

It ends in paramiko's `SFTPServer.set_file_attr`, and that function's handling of a size attribute is not a resize of an existing file:

```
if attr._flags & attr.FLAG_SIZE:
    with open(filename, "w+") as f:
        f.truncate(attr.st_size)
```

`open(filename, "w+")` creates the file if it does not exist and opens it for writing regardless. An `SSH_FXP_SETSTAT` carrying `st_size = 0` therefore

creates or truncates an arbitrary path outside the volume. Against the lab above, one setstat turns `secret.txt` from 31 bytes into 0, and a setstat naming a path that does not yet exist brings it into being. `set_file_attr` also applies mode and ownership when those flags are present, so the same handler chmods and chowns outside the volume as a lesser included effect.

So the integrity reach of the three is: create directories, remove directories, and create-or-truncate files, anywhere a symlink leaves the volume.

The fix that proved the pattern

This class already cost copyparty an advisory six days before I looked. GHSA-3fhv-rhjw-7hrg — *"sftp did not fully enforce volflags xvol/xdev"* — was the read side of the same gap, and its fix is instructive precisely because of how small it is. Commit `b80a210e`, released in 1.20.22, is `+1/−2` in one file. It rewrote `_open` to call `v2a` — the helper that was already sitting 150 lines above it, already correct.

That fix is the right change made once. It adds no new authorizer; it points one handler at the enforcement that already existed. And in doing so it says, in code, what the enforcement model is: *each handler is responsible for calling the guard.* It closed `_open` and left `_mkdir`, `_rmdir` and `_chattr` — three more handlers in the same file, resolving paths the same unguarded way, each one line from correct in exactly the manner `_open` had just been corrected.

The suggested fix in my report is therefore not a design: it is `b80a210e`, three more times. Resolve `_mkdir`, `_rmdir` and `_chattr` through `v2a` with the appropriate rights, as the other four handlers already do.

The deeper suggestion, which I raised rather than prescribed, is that a boundary enforced as a per-call-site list will keep shedding call sites — a new handler, a new protocol front end — until the check moves to the one place every path

must pass to become real. HTTP already has that shape. SFTP has a helper that four of seven handlers happen to call.

Measured per tag, still open at the current release

Whether a version is affected is not inferred from the commit date; it is the reproduction run at each released tag.

```
version   read control   mkdir     chattr    rmdir     secret.txt after
1.20.0    read=OK        ALLOWED   ALLOWED   ALLOWED   0 B  (was 31 B)
1.20.19   read=OK        ALLOWED   ALLOWED   ALLOWED   0 B
1.20.21   read=OK        ALLOWED   ALLOWED   ALLOWED   0 B
1.20.22   read=OK->DENY  ALLOWED   ALLOWED   ALLOWED   0 B
1.20.23   read=OK->DENY  ALLOWED   ALLOWED   ALLOWED   0 B
1.20.24   read=OK->DENY  ALLOWED   ALLOWED   ALLOWED   0 B
```

Each row is a fresh install, a fresh lab tree and a fresh server process. The read-control column is the point of the table: at 1.20.22 the earlier fix lands and the read through the symlink starts being denied — the harness rediscovering copyparty's own `patched_versions: 1.20.22` on its own. The other three columns never change. The SFTP front end first appears in `sftpd.py` at 1.20.0; every 1.19 tag returns *fatal: invalid object name* for that path. **The three handlers have been unguarded for the entire life of the SFTP server, and remain so in 1.20.24, the current release.** That is the range in the record: `<= 1.20.24`.

One sibling is out of scope of this identifier and worth naming. TFTP calls `chk_ap` nowhere in its front end, which makes the same escape available without an account — and a directory read request returns a generated listing, so there is nothing to guess. I reported it in the same finding; the assigned record covers the SFTP handlers, which need an authenticated account with write rights, and I score and describe those here to match it. The TFTP half is a separate conversation.

An identifier without a fix

I reported all of this through GitHub's private vulnerability reporting on 13 September. The maintainer closed the advisory the next day, 14 September, without a single comment. Nothing was disputed and nothing was confirmed; no fix was committed and no release shipped. I am stating that as the bare sequence rather than reading intent into it — the advisory was accepted for credit, then closed, and the tree has not moved on these handlers since.

That left a defect present in the current release with no vendor identifier and no vendor record. So the identifier came from elsewhere: I asked VulnCheck, a CVE Numbering Authority that assigns for open-source findings the vendor has not, and they assigned **CVE-2026-93353**, crediting me as finder, at `CVSS:4.0/AV:N/AC:H/AT:N/PR:L/UI:N/VC:N/VI:H/VA:N/SC:N/SI:N/SA:N` — 6.0, integrity-only, scoped to the SFTP handlers. A CNA that publishes a record requires a public reference first: something that names the product, the vulnerability type and the affected code path. This post is that reference; the VulnCheck advisory and the CVE record went public alongside it.

It is the pattern I have written about before, in a sharper form. Of the identifiers I have been credited on, the ones that exist do so because a foundation or a third-party CNA read the code and assigned — not because the repository host did. Here there was no fixed release to point a record at and no vendor advisory to carry an ID, and the record exists anyway, describing code you can still run.

What I did not claim

The strongest-looking lead in this finding died on verification, and the report says so rather than shipping it. `_rmdir` and `_chattr` carry the exact `os.path.join(vn.realpath, rem)` shape that two earlier copyparty advisories (CVE-2025-58753, CVE-2026-32108) fixed for single-file shares, which would have made this a third instalment of that story. But both handlers pass `will_del=True` to `vfs.get`, and copyparty's share logic hardcodes move and

delete to `False` for a share, so the share path is never reached. It looks like the same bug and it is not; stopping at the grep would have produced a confident and wrong report. I also did not test whether an anonymous SFTP configuration drops the SFTP handlers from *account required* to *no account*, and did not assert it — the record's `PR:L` reflects what I ran.

The shape to look for

The generalisable part is not "audit your SFTP handlers." It is what a security boundary looks like when it is maintained as a list of call sites instead of a gate.

Grep for the *check*, and read the result as a roster of everyone who remembered. Then find the operations that are not on it. A helper that does the enforcement correctly is not protection; it is protection for the handlers that call it, and the ones that resolve their own paths are a parallel set that has to be enumerated separately. When a fix for exactly this class lands and it is a one-line change that points one handler at the existing helper, the fix has told you the model — and the question it leaves is arithmetic: how many other handlers resolve a path without that call. Here it was three, plus a protocol.

Reported 13 September through the project's private advisory flow; closed without comment 14 September, unfixed. Identifier assigned by VulnCheck and published 24 September, crediting me — CVE-2026-93353, VulnCheck advisory. Still unfixed in copyparty at the time of writing.

Eurico Nicacio — @h3llh0und

← all posts

Eurico Nicacio · h3llh0und