0-DAY NO CVE HIGH PUBLIC

ProFTPD SITE COPY dest follows symlink

ProFTPD 1.3.9d SITE COPY checks dest Limit on an unresolved path, then pr_fs_copy_file2 follows a dest symlink. Authenticated FTP can overwrite a WRITE-denied file. Labbed. Distinct from Issue 2256 RNTO EXDEV.

NAME **ProFTPD SITE COPY dest follows symlink**	TYPE **0-day analysis**
CVE **n/a**	CVE RISK **high**
DISCLOSURE STATUS **public**	VENDOR **ProFTPD Project**
AFFECTED **ProFTPD 1.3.9d (f59f248) contrib mod_copy. SITE SYMLINK plant uses mod_site_misc. CopyEngine default on.**	PUBLISHED **12 Oct 2026**
UPDATED **12 Oct 2026**	TAGS **0-day, proftpd, site-copy, privilege-leftover, file-write, cwe-59, cwe-863**

TL;DR

Lab files and proof-of-concept here.

01 Limit saw /upload/link. open(2) saw /secret/victim

I am @abraxas_null. The proof of concept is on GitHub: **abraxas/proftpd-site-copy-dest-follow** (loopback client). The lab stack is `lab/`: Dockerfile, docker-compose.yml, run.sh, poc.py, proftpd.conf, entrypoint.sh. Authorized lab only. It talks to loopback.

This is **ProFTPD 1.3.9d** (`f59f248e26bdfa514443e42bf72a145641df635d`) contrib `mod_copy`. No CVE yet. CWE-59 / CWE-863. 7.1 High (`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N`). Remote. Authenticated FTP. Privilege leftover / file write.

`SITE COPY` asks `<Limit WRITE>` whether `/upload/link` is allowed. That string is a symlink sitting in a WRITE-allowed directory. The Limit says yes. Then `pr_fs_copy_file2` comments that `stat` / `open` follow the dest symlink, and opens `O_WRONLY|O_CREAT|O_NONBLOCK` with no `O_NOFOLLOW`. `/secret/victim` is `<Limit WRITE> DenyAll`. Direct `STOR` of that file is `550`. Direct `SITE COPY` to that file is `550`. `STOR` of the link is `550`. Copy through the link is `200`, and the victim goes from `ORIGINAL` to the witness.

STOR Limit-checks a resolved path. SITE COPY dest does not. That is the leftover.

I did not overwrite host `/etc`. The oracle is a witness string in a WRITE-denied file the session UID can already Unix-write. FTP policy said no. `open(2)` said yes.

02 Why ProFTPD

I wanted a C FTP daemon with a real `<Directory>` / `<Limit>` model, a long SITE history, and a tag that had just finished a noisy CVE year. ProFTPD is honest about what it is: prefork, `RootRevoke` default on since 1.3.7rc1 so the worker is the login UID, sample `basic.conf` leaving `DefaultRoot` commented out, contrib modules off until you load them, Debian-style shared builds loading them anyway.

I only wanted Critical or High. Remote and Local. Medium stays on the floor unless it chains. DoS stays on the floor unless it is a write primitive. Anonymous-as-designed is out of bar. Unauthenticated banner / FEAT / SYST is recon. Distro packaging of 1.3.8.c is a different tree.

Default core FTP on this pin came back empty. The leftovers that survived are contrib and config-gated. This one is `mod_copy`. The plant uses `mod_site_misc` unless the dest link already exists. `CopyEngine` defaults on once the module is loaded.

03 Pinning v1.3.9d

Tag `v1.3.9d`, 14 Aug 2026, HEAD `f59f248e26bdfa514443e42bf72a145641df635d`. `include/version.h` says `PROFTPD_VERSION_TEXT "1.3.9d"` and `PROFTPD_VERSION_NUMBER 0x0001030908`. The lab image greps those strings before `make install`. `proftpd -v` in the run is `ProFTPD Version 1.3.9d`.

1.3.10rc4 shipped the same calendar day. That is not this pin. `mod_procfs` is not in 1.3.9d.

Intake is security@proftpd.org. Unpaid. No bounty.

04 The skip list is half the hunt

Before you look for leftovers, you write down what is already public so you do not "discover" a CVE with a press release. 1.3.9d already closed or publicly described the High classes this hunt was looking for:

- CVE-2026-42167 `mod_sql` SQLi. 1.3.9a/b.
- CVE-2026-44331 `mod_wrap2_sql` rDNS SQLi. 1.3.9b.

- CVE-2026-35025 RNFR `/proc/self/root` unresolved ACL bypass. Public. Skip the exact bug. Debian still marks 1.3.9d vulnerable. `DefaultRoot` is the documented mitigation.
- CVE-2026-53994 / 63090 SFTP heap overflow. 1.3.9c.
- CVE-2026-63091 SCP `off_t` overflow. 1.3.9c.
- 1.3.9a-d NEWS: MLSD facts overflow, RewriteMap unescape OOB, Controls ACLs, SSH ETM/ChaChaPoly, empty-password `mod_auth_file` / `mod_sql_passwd`, `.ftpaccess` policy, PASV `AllowForeignAddress`, STAT `-C` UAF.
- Public GitHub issues still sitting in this tarball: 2170, 2256, 2286, 2319, 2330, 2301, 2175, 2180, 2095, 831.
- Historical `mod_copy`: CVE-2015-3306 unauthenticated `SITE CPFR`/`CPTO`. CVE-2019-12815 default-allow copy. Auth gate is in tree. `copy_copy` returns `530 Please login with USER and PASS` if `authenticated` is false.

Two public tickets sit next to this leftover and are not it.

Issue 2095 is SITE COPY skipping `<Limit READ>` on the **source**. `copy_copy` `dir_check`s `G_WRITE` on dest and never `G_READ`s src. Public. Master got a READ check. 1.3.9d still has the skip. I did not re-file that exact bug. This leftover is dest WRITE on an unresolved path. The lab even requires READ on the victim to plant the link with `SITE SYMLINK`.

Issue 2256 is RNTO EXDEV fallback. Cross-filesystem `rename(2)` returns `EXDEV`, ProFTPD copies with `pr_fs_copy_file` / `pr_fs_copy_file2`, and that copy follows source and dest symlinks. Same-filesystem RNTO replaces the dest link entry, which is what the tests want. Public. Reporter: no auth bypass, no LPE. Milestone 1.3.10. This leftover is SITE COPY dest on **every** copy, not only EXDEV, and it is an ACL bypass of `<Limit WRITE>` because the Limit never sees the resolved dest.

Those stay on the skip list.

05 Fifteen surfaces, one FTP copy leftover

I split the tree. FTP parser, path canonicalization, SITE, MLSD/facts, chroot/privs, Controls, `mod_sftp`, `mod_tls`, `mod_sql`, rewrite / `.ftpaccess` / exec, auth, data/REST, wrap2/ldap/radius/snmp, digest/json/display, ifsession/DSO.

Most of that is ProFTPD doing what ProFTPD said it would do.

Parser leftovers 2019/2020/1683/2030 are closed on this pin. Controls 2210 deny-all. `RootRevoke` on. Auth 2319/2279/831 skip-public. REST/RANG 2286 skip-public. SQL 42167 closed. SITE CHMOD/CHGRP Limit-check a realpath. HASH/XCRC `dir_realpath` then `G_READ`. Sample `<Anonymous>` `<Limit WRITE> DenyAll` covers the `G_WRITE` group these SITE handlers pass.

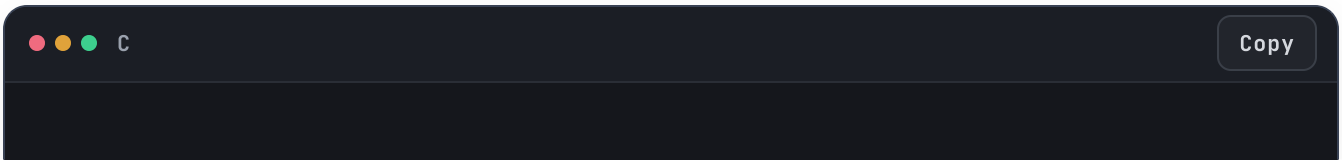
Default core FTP unpublished Crit/High: none.

Contrib is where the Highs sat. Same pin, two other leftovers I am not mixing into this post: SFTP `copy-file` skipping WRITE on a dest that does not exist yet, and `RewriteMap int:utf8trans` overflowing two 1024-slot BSS buffers. This write-up is FTP `SITE COPY` dest.

Then I opened `copy_copy` next to `xfer_pre_stor`.

06 SITE COPY dest is not STOR dest

`copy_copy` is the `SITE COPY from to` handler. `CopyEngine` defaults true. Auth required. Paths with spaces are a comment (`/* XXX What about paths which contain spaces? */`) and a functional bug, not this leftover.



```
from = dir_canonical_vpath(cmd->tmp_pool, decoded_path);
/* ... decode dest ... */
to = dir_canonical_vpath(cmd->tmp_pool, decoded_path);

cmd_name = cmd->argv[0];
pr_cmd_set_name(cmd, "SITE_COPY");
if (!dir_check(cmd->tmp_pool, cmd, G_WRITE, to, NULL)) {
  /* 550, EPERM */
}
pr_cmd_set_name(cmd, cmd_name);

if (copy_paths(cmd->tmp_pool, from, to) < 0) {
  /* 550 */
}
```

`dir_canonical_vpath` joins with the virtual cwd and runs `pr_fs_clean_path`. Dots, extra slashes, that is it. No symlink walk. `dir_check` matches `<Directory>` blocks against that lexical string. `/upload/link` lives under a WRITE-allowed tree. `/secret/victim` lives under `<Limit WRITE> DenyAll`. Direct dest `/secret/victim` dies here. Dest `/upload/link` does not.

`copy_paths` then `lstat`s the source (regular file in the lab), `stat`s the dest (follows), honors `AllowOverwrite`, and calls `pr_fs_copy_file2`.

`pr_fs_copy_file2` is not shy about the follow:

C Copy

```
  /* We use stat() here, not lstat(), since open() would follow a symlink
   * to its target, and what we really want to know here is whether the
   * ultimate destination file exists or not.
   */
  pr_fs_clear_cache2(dst);
  if (pr_fsio_stat(dst, &dst_st) == 0) {
    /* ... reject if dest is a directory ... */
    dst_existed = TRUE;
  }

  dst_fh = pr_fsio_open(dst, O_WRONLY|O_CREAT|O_NONBLOCK);
```

No `O_NOFOLLOW`. The comment is the bug. Knowing the ultimate dest exists is useful for the unlink-on-failure path. Checking `<Limit WRITE>` against that ultimate dest is the part `copy_copy` skipped.

STOR does the other thing. `xfer_pre_stor`:

```c
    pr_fs_clear_cache2(decoded_path);
    path = dir_best_path(cmd->tmp_pool, decoded_path);

    if (path == NULL ||
        !dir_check(cmd->tmp_pool, cmd, cmd->group, path, NULL)) {
      /* 550 denied by <Limit> */
    }
```

`dir_best_path` is "the most fully canonicalized path possible." It walks existing prefixes through `pr_fs_resolve_path`. `/upload/link` becomes `/secret/victim` (inside the `DefaultRoot` jail, the real file). `<Directory /secret>` `<Limit WRITE>` hits. `550`. Same for a STOR of `/secret/victim` itself.

That is the split. Same UID. Same Unix mode. Same `<Limit>`. STOR resolves, then checks. SITE COPY checks, then follows.

07 SITE SYMLINK is the plant, and it wants READ

You need a dest symlink in a WRITE-allowed directory whose target is the WRITE-denied file. A pre-existing link is enough. The lab plants one with contrib `mod_site_misc` `SITE SYMLINK`. `SiteMiscEngine` defaults on once loaded.

`site_misc_symlink`:

```c
      src = dir_canonical_path(cmd->tmp_pool, decoded_path);
      cmd->argv[0] = "SITE_SYMLINK";
```

```
    if (!dir_check_canon(cmd->tmp_pool, cmd, G_READ, src, NULL)) {
      /* 550 */
    }
    dst = dir_canonical_path(cmd->tmp_pool, decoded_path);
    if (!dir_check_canon(cmd->tmp_pool, cmd, G_WRITE, dst, NULL)) {
      /* 550 */
    }
    /* pr_fsio_stat(src), then pr_fsio_symlink(src, dst) */
```

Src is `G_READ`. `dir_check_canon` is `dir_check(..., dir_best_path(path), ...)`. Planting a link **to** the victim needs READ on that file. The split this leftover needs is READ allowed, WRITE denied. A secret you cannot `RETR` is not this plant. A secret you can `RETR` but cannot `STOR` is.

`pr_fsio_symlink` stores the already-canonical src string. Lab: `/upload/link` → `/secret/victim`. `stat` from the container: `symbolic link '/home/labuser/upload/link' → '/secret/victim'`.

Cannot plant host-absolute targets from a `DefaultRoot` jail. Canonical src cannot point at host `/etc`. The documented NcFTPd-chroot test already asserts `SITE SYMLINK /etc/passwd` fails inside a jail. This leftover is not a chroot escape. Unix DAC still applies. The session UID has to be able to `open` the victim for write.

08 What I almost filed instead

SITE RMDIR recursive follow. The SITE hunt claimed Limit on the lexical symlink, then `stat` / `opendir` walking the target, shutil-rmtree through a planted link. SITE RMDIR actually `dir_check_canon`s. `dir_check_canon` is `dir_best_path`. The existing dest symlink resolves before Limit. `<Directory /secret>` `<Limit WRITE>` (and the lab also names `RMD`) denies it. I dropped it. The copy lab proved the drop: `SITE RMDIR /upload/link` → `550 RMDIR /upload/link: Operation not permitted`, victim still holding the witness. Core `RMD` of the same link is `550 Permission denied`. `rmdir(2)` would have been `ENOTDIR` on a file symlink anyway. The recursive-delete leftover is not sitting on this pin the way the hunt first sold it.

DELE `/proc/self/root`. `core_dele` is `dir_canonical_path` plus lexical `dir_check` plus `unlink`. Same unresolved procfs-root class as public CVE-2026-35025 / Issue 2170, which already asked about DELE. Skip-public. SFTP `REMOVE` uses `dir_check_canon` and is closed for that class. `mod_procfs` is not in 1.3.9d.

Issue 2095 source READ as this leftover. Source READ skip is still true on 1.3.9d. Public. The dest WRITE leftover is a different check on a different path.

Calling Issue 2256 this bug. EXDEV RNTO follows dest only when `rename(2)` cannot. SITE COPY dest follows on every copy, and the Limit never saw the resolved name. Public 2256 is "symlink directory-entry semantics of a rename." This is "FTP ACL on the wrong string."

Unauth copy. CVE-2015-3306. Closed. `530`.

DefaultRoot to host `/etc`. Closed. The jail aliases the jail.

Inflating file write to RCE. The session UID can truncate a Limit-denied file it can Unix-write. If that file is later a hook, a cron fragment, or an authorized_keys the UID owns, that chain is not in the lab. Class is privilege leftover / file write.

09 The lab

Pin 1.3.9d built in Docker with `--with-modules=mod_copy:mod_site_misc`. Loopback `127.0.0.1:18720`. Compose project `proftpd-site-copy-dest-follow`. Witness `PROFTPD-SITE-COPY-DEST-FOLLOW-WITNESS`. Image greps `PROFTPD_VERSION_TEXT` for `1.3.9d` before install.

`lab/Dockerfile` is the pin:

DOCKERFILE Copy

```
# Pin: GitHub proftpd/proftpd tag v1.3.9d
# commit f59f248e26bdfa514443e42bf72a145641df635d
COPY src /src/proftpd

```

```
WORKDIR /src/proftpd

RUN grep -F 'PROFTPD_VERSION_TEXT' include/version.h \
    && grep -F '"1.3.9d"' include/version.h \
    && grep -F '0x0001030908' include/version.h \
    && ./configure \
        --prefix=/usr/local \
        --sysconfdir=/etc/proftpd \
        --localstatedir=/var/proftpd \
        --disable-auth-pam \
        --disable-cap \
        --disable-ident \
        --disable-nls \
        --with-modules=mod_copy:mod_site_misc \
    && make -j"$(nproc)" \
    && make install \
    && /usr/local/sbin/proftpd -v
```

`lab/docker-compose.yml` binds FTP and passive ports to loopback:

YAML Copy

```
    ports:
      - "127.0.0.1:18720:21"
      - "127.0.0.1:18721-18730:18721-18730"
```

`lab/proftpd.conf`: `DefaultRoot ~`, `CopyEngine on`, `SiteMiscEngine on`, `AllowOverwrite on`, `AuthOrder mod_auth_file.c`. `RootRevoke` is the compiled default. Two `<Directory>` blocks, `/home/labuser/secret` and `/secret`, both `<Limit WRITE STOR APPE MKD RMD DELE RNFR RNTO> DenyAll` and `<Limit READ RETR> AllowAll`. Upload and readable have no WRITE deny.

`lab/entrypoint.sh` lays down `/home/labuser/readable/src.txt` with the witness, `/home/labuser/secret/victim` with `ORIGINAL`, mode `0644`, owner the session UID. Unix DAC would allow the write. FTP Limit is supposed to be the gate.

Control, then plant, then copy, then the closed neighbors:

01 Banner `220 ProFTPD Server (PROFTPD-SITE-COPY-DEST-FOLLOW)`. Login. `FEAT` advertises `SITE COPY`, `SITE MKDIR`, `SITE RMDIR`, `SITE SYMLINK`, `SITE UTIME`.

02 `STOR /secret/victim` → `550 /secret/victim: Operation not permitted`. Disk still `ORIGINAL`.

03 `SITE COPY /readable/src.txt /secret/victim` → `550 /secret/victim: Operation not permitted`. Disk still `ORIGINAL`.

04 `SITE SYMLINK /secret/victim /upload/link` → `200`. Link node is a symlink to `/secret/victim`.

05 `SITE COPY /readable/src.txt /upload/link` → `200 SITE COPY command successful`.

06 `RETR /secret/victim` body is the witness. `docker compose exec cat` of the host path is the witness. `ORIGINAL` is gone.

07 `STOR /upload/link` → `550`. Victim still the witness, not the STOR marker.

08 `SITE RMDIR /upload/link` → `550`. Victim intact.

09 `RMD /upload/link` → `550`. Victim intact.

10 `DELE /upload/link` → `250`. Link node gone. Victim still the witness.

Replay:

PLAIN TEXT Copy

```
git clone https://github.com/abraxas/proftpd-site-copy-dest-follow
cd proftpd-site-copy-dest-follow/lab
./run.sh
```

Bind it to loopback. The loopback client is `lab/run.sh` with a banner on it.

PLAIN TEXT Copy

```
SUCCESS PROFTPD-SITE-COPY-DEST-FOLLOW victim-has-witness=yes stor-secret=denied s
```

10 Wrong turns, in the order I made them

Treating `dir_canonical_vpath` as `dir_best_path`. Canonical is lexical clean. Best is resolve-as-far-as-the-tree-exists. SITE COPY dest uses the first. STOR uses the second. I had to put the two handlers next to each other before the leftover was obvious instead of "copy is just another STOR."

SITE RMDIR as the headline. Recursive follow through a dest link would have been a delete primitive. `dir_check_canon` already resolves. The lab is the argument, not a comment in the hunt file. `550`, victim intact.

DELE of the dest link as a victim-delete. `unlink` of a symlink removes the link. `core_dele` even comments that it should not delete the file a symlink points to. `250`, victim still holding the witness. Correct Unix, and a useful negative for anyone who wanted this to be Issue 2170 in a new hat.

Issue 2256 as already this. EXDEV is a rename fallback. SITE COPY is the copy command people enable on purpose so they do not round-trip bytes through the client. Follow on every dest, Limit on the unresolved name, ACL bypass. Different ticket.

Issue 2095 as already this. Source READ is a different `dir_check` that is not there. Dest WRITE is a `dir_check` that is there, on the wrong string.

Scoring it Medium because you already have READ. READ lets you `RETR` the secret. WRITE deny is why you cannot change it. Overwriting a policy-denied file is integrity High. The lab's `RETR` after copy is the oracle, not a second leftover.

A pasteable `SITE COPY` against someone else's `:21`. Theatre. The missing resolve-or-`O_NOFOLLOW` is the useful part. If you own the daemon, run the proof of concept against loopback.

11 What an attacker can do

Needs an authenticated FTP login, contrib `mod_copy` loaded, `CopyEngine` on (the default once loaded). Planting the dest link via FTP also needs contrib `mod_site_misc` unless the link already exists. Victim file must be Unix-writable by the session UID. For the `SITE SYMLINK` plant, victim must be READ-allowed. No unauth. No shell. No RootRevoke bypass. No host `/etc` from a `DefaultRoot` jail.

PRIVILEGE

Integrity: overwrite a `<Limit WRITE> DenyAll` file

PRIVILEGE
Integrity: overwrite shell rc, `authorized_keys`, a cron fragment, or another file the UID owns b
Recon: `RETR` of the overwritten victim
Negative: `STOR /upload/link`
Negative: `SITE COPY` dest `/secret/victim`
Negative: `SITE RMDIR /upload/link`
Negative: core `RMD /upload/link`
Negative: `DELE /upload/link`

PRIVILEGE
Negative: unauthenticated `SITE COPY`
Negative: DefaultRoot jail to host `/etc`
Negative: Issue 2095 source READ skip
Negative: Issue 2256 RNTO EXDEV follow
RCE inside `proftpd`

`CPFR` / `CPTO` is the two-step form of the same module. `CPTO` `dir_check`s dest WRITE the same way. This lab drives `SITE COPY`.

12 The fix

Limit-check the **resolved** dest, the way STOR already does: `dir_best_path` / `dir_check_canon` on `to` before `copy_paths`. Or open the dest with `O_NOFOLLOW` so a planted link cannot be the write target. Both is better. `pr_fs_copy_file2` following dest is how Issue 2256 happened on EXDEV RNTO. SITE COPY is the same `open` with a Limit that never asked the resolved name.

Until then: `CopyEngine off`, or do not load `mod_copy`, if you use `<Directory>` WRITE denies as a security boundary and you also allow dest symlinks in a WRITE-allowed tree. Turning off `mod_site_misc` only removes the FTP plant. A pre-existing link is enough.

I am not going to print a `SITE COPY` you can paste at someone else's `:21`. The unresolved dest plus the follow is the useful part. If you own the daemon, run the proof of concept against loopback.

13 References

- Proof of concept: abraxas/proftpd-site-copy-dest-follow · proftpd-site-copy-dest-follow-Abraxas-Labs.py
- Lab: `lab/` · Dockerfile · docker-compose.yml · run.sh · poc.py · proftpd.conf · entrypoint.sh
- @abraxas_null · github.com/abraxas · abraxaslabs.tech · abraxas.null@proton.me
- CWE-59 · CWE-863
- Tree: proftpd/proftpd v1.3.9d (`f59f248e26bdfa514443e42bf72a145641df635d`) · `contrib/mod_copy.c` · `src/fsio.c` · `modules/mod_xfer.c` · `contrib/mod_site_misc.c` · `src/dirtree.c` · `src/support.c` · `doc/contrib/mod_copy.html`
- Nearby public (not this leftover): Issue 2095 SITE COPY source READ · Issue 2256 RNTO EXDEV follow · Issue 2170 / CVE-2026-35025 RNFR `/proc/self/root` · Issue 2175 SITE RMDIR PathAllowFilter · CVE-2015-3306 · CVE-2019-12815

- Intake: security@proftpd.org

Abraxas Labs

Independent security research by abraxas.

Analysis of 0-days, n-days, malware, and the

leftovers that still matter.

GitHub **@abraxas_null** **abraxas.null@proton.me** **abraxaslabs.tech**

Independent research. Private analysis by arrangement. No ads. abraxas.null@proton.me · For defensive awareness and authorized testing only.

© 2026 Abraxas Labs