SECURITY ADVISORY

GNU tar Stream Desynchronization

Listing vs Extraction Mismatch / Hidden File Injection

Reporter	Guillermo de Angel Garcia
Date	2026-03-18
Affected Software	GNU tar 1.35 (likely earlier versions)
Not Affected	bsdtar 3.7.2 (libarchive 3.7.2)
Severity	Medium, CVSS 3.1: 5.3 to 6.5
Status	Coordinated Disclosure Pending

1. Executive Summary

GNU tar does not reject archives where non-data-bearing typeflags (symbolic link '2', character device '3', block device '4', FIFO '6') carry a non-zero *size* field. Instead, it processes them inconsistently between listing mode (**tar -t**) and extraction mode (**tar -x**). This creates a stream desynchronization that enables **hidden file injection**: files embedded in the data region of these entries are invisible to listing but are materialized during extraction.

An attacker can craft a tar archive where **tar -t** reports N entries but **tar -x** creates N+M files. The additional files never appear in any listing output, allowing attackers to bypass pre-extraction inspection mechanisms that rely on **tar -t** or equivalent listing APIs. The result is **incomplete archive inspection**.

CVSS 3.1 Score	Severity	Confidentiality	Integrity	Availability
5.3 to 6.5	MEDIUM	Low	Medium	Low

2. Root Cause

According to POSIX, typeflags 2, 3, 4, and 6 represent non-data-bearing entries and are expected to carry a *size* field of zero. GNU tar accepts archives where these typeflags carry *size > 0* without warning or error, but processes them inconsistently across modes.

tar -t (listing mode)

Respects the *size* field. Skips forward by *size* bytes, rounded up to 512-byte blocks, to locate the next header. The skipped blocks are treated as opaque data and are never exposed to the user.

tar -x (extraction mode)

Ignores the *size* field for these typeflags. After processing the carrier header, it immediately reads the next 512-byte block as the next header. If that block contains a valid tar header with a correct checksum, it is parsed and extracted as a new entry.

The core issue is that GNU tar **accepts malformed input without validation** and then **processes it via divergent code paths** depending on the operation mode. Data blocks that are skipped during listing are parsed as headers during extraction.

3. Attack Scenario

Archive Layout

The following minimal archive structure demonstrates the attack:

```
Block 0:   [carrier_header]   typeflag='3' (chardev), size=1024
Block 1:   [injected_header]  typeflag='0' (regular), name='backdoor.sh'
Block 2:   [injected_data]    content of backdoor.sh
Block 3:   [marker_header]    typeflag='0' (regular), name='README.txt'
Block 4:   [marker_data]      content of README.txt
Block 5-6: [end-of-archive]   two zero blocks
```

Listing vs Extraction Discrepancy

Mode	Entries Reported
`tar -t`	carrier_entry, README.txt

`tar -x`	carrier_entry, **backdoor.sh** (hidden, not in listing), README.txt

backdoor.sh is never reported by **tar -t** but is created on disk by **tar -x**.

Exploit Flow

#	Action
1	Attacker crafts an archive with a non-data-bearing carrier entry (device, FIFO, or symlink) whose size field covers embedded malicious content.
2	Security scanner uses tar -t (or equivalent listing API) to enumerate archive contents. The listing is incomplete, and the injected file is not reported.
3	Archive passes inspection based on the incomplete inventory.
4	System extracts with tar -x. The injected file is created on disk alongside legitimate entries.
5	Post-extraction verification, if any, cannot detect the discrepancy unless it independently walks the filesystem, because the scanner's content inventory was incomplete from the start.

High-Risk Deployment Contexts

• CI/CD pipelines that scan tarballs before deployment

• Container image inspection tools

• Package repository scanners

• Automated software distribution systems

• Any workflow where tar -t output is treated as authoritative

4. Affected Typeflags

Typeflag	Name	GNU tar Desync	bsdtar
`'2'`	Symbolic link	YES	Consistent
`'3'`	Character device	YES	Consistent
`'4'`	Block device	YES	Consistent
`'5'`	Directory	No (both modes consistent)	Consistent
`'6'`	FIFO	YES	Consistent

5. Proof of Concept

A standalone Python script generates four PoC archives, one per vulnerable typeflag. No dependencies beyond Python 3 are required.

```
python3 scripts/cve_desync_poc.py --verify
```

Reproduction Steps

```
# 1. Generate PoC archives
python3 scripts/cve_desync_poc.py
# 2. List archive contents
```

```
tar -tf payloads/cve_desync/desync_chardev.tar
#  -> carrier_entry  marker.txt
# 3. Extract archive
mkdir /tmp/desync_test
tar -xf payloads/cve_desync/desync_chardev.tar -C /tmp/desync_test
# 4. Observe the discrepancy
ls -la /tmp/desync_test/
#  -> injected.txt  (NOT in listing output)  marker.txt
# 5. Compare with bsdtar (consistent behavior)
bsdtar -tf payloads/cve_desync/desync_chardev.tar
#  -> carrier_entry  injected.txt  marker.txt
```

6. Impact Assessment

Dimension	Rating	Notes
Confidentiality	Low	No information disclosure.
Integrity	Medium	Hidden file injection. Files can be created on the target system that were not visible during pre-extraction inspection. Contents are fully attacker-controlled.
Availability	Low	No denial of service.
Attack Complexity	Low	Crafted archive is less than 3 KB, structurally simple, requires no special permissions.
Privileges Required	None	Any user who can provide a tarball for extraction can exploit this.

Key differentiator: This is a single-implementation inconsistency within GNU tar itself. It does not require a cross-tool pipeline. The same binary produces different content inventories depending on the operation (-t vs -x). Any workflow that uses GNU tar listing output as a security gate is vulnerable.

7. Suggested Remediation

Option 1: Consistent skip (minimal change)

GNU tar's extraction mode should respect the size field for non-data-bearing typeflags, consistent with listing mode behavior. After processing a header with typeflag in {2, 3, 4, 6}, the parser should skip ceil(size / 512) * 512 bytes before reading the next header, regardless of operating mode.

Option 2: Reject malformed input (recommended)

GNU tar should reject archives where non-data-bearing typeflags specify a non-zero size, as these violate expected format constraints. A warning or error at parse time would prevent the inconsistency entirely and alert users to potentially crafted archives.

Option 3: Warn and skip

Emit a warning when encountering size > 0 on non-data typeflags, then skip the data blocks consistently in both modes. This preserves backward compatibility for non-conforming but legitimate archives while closing the desync.

8. Disclosure Timeline

Date	Action
2026-03-16	Vulnerability discovered during differential testing
2026-03-18	Report sent to bug-tar@gnu.org
2026-03-18	Report sent to secalert@redhat.com (Red Hat CNA)

9. References

• POSIX.1-2017 pax interchange format: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/pax.html

• GNU tar source: https://git.savannah.gnu.org/cgit/tar.git

• Related pattern: CVE-2018-1000001 (listing/extraction divergences in zip utilities)

10. Attachments

File	Description
`scripts/cve_desync_poc.py`	Standalone PoC generator. Produces all four payload archives (chardev, symlink, blockdev, FIFO variants). No dependencies beyond Python 3.

Coordinated Disclosure // 2026-03-18