

python / cpython Public

<> Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

# Commit 19de092



6 people authored on Jun 3, 2025 · ✖ 22 / 25 · Partially verified

```
[3.12] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (GH-135066)

Addresses CVEs 2024-12718, 2025-4138, 2025-4330, and 2025-4517.

(cherry picked from commit 3612d8f)

Co-authored-by: Łukasz Langa <lukasz@langa.pl>
Signed-off-by: Łukasz Langa <lukasz@langa.pl>
Co-authored-by: Petr Viktorin <encukou@gmail.com>
Co-authored-by: Seth Michael Larson <seth@python.org>
Co-authored-by: Adam Turner <9087854+AA-Turner@users.noreply.github.com>
Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>
```

🔗 3.12 (#135066) · 🔍 v3.12.13 ... v3.12.11

1 parent [f3272d8](#) commit 19de092 📄

**11 files changed** +1,064 -139 🟢🟢🟢🟢

📄 ↑ Top ⚙️

🔍 Filter files... ☰

- ✓ 📁 Doc
  - ✓ 📁 library
    - 📄 os.path.rst
    - 📄 tarfile.rst
  - ✓ 📁 whatsnew
    - 📄 3.12.rst
- ✓ 📁 Lib
  - 📄 genericpath.py

- ntpath.py
- posixpath.py
- tarfile.py
- test
  - test\_ntpath.py
  - test\_posixpath.py
  - test\_tarfile.py
- Misc/NEWS.d/next/Security
  - 2025-06-02-11-32-23.gh-issue-135034.RLGjbp.rst



Search within code



Doc/library/os.path.rst



```

@@ -377,10 +377,26 @@ the :mod:`glob` module.)
377 377     links encountered in the path (if they are supported by the operating
378 378     system).
379 379
380 -   If a path doesn't exist or a symlink loop is encountered, and strict is
381 -   ``True``, :exc:`OSError` is raised. If strict is ``False``, the path is
382 -   resolved as far as possible and any remainder is appended without checking
383 -   whether it exists.
380 +   By default, the path is evaluated up to the first component that does not
381 +   exist, is a symlink loop, or whose evaluation raises :exc:`OSError`.
382 +   All such components are appended unchanged to the existing part of the path.
383 +
384 +   Some errors that are handled this way include "access denied", "not a
385 +   directory", or "bad argument to internal function". Thus, the
386 +   resulting path may be missing or inaccessible, may still contain
387 +   links or loops, and may traverse non-directories.
388 +
389 +   This behavior can be modified by keyword arguments:
390 +
391 +   If strict is ``True``, the first error encountered when evaluating the
392 +   path is
393 +   re-raised.
393 +   In particular, :exc:`FileNotFoundError` is raised if path does not exist,

```

```

394 + or another :exc:`OSError` if it is otherwise inaccessible.
395 +
396 + If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
397 + :exc:`FileNotFoundError` are re-raised (as with `strict=True`).
398 + Thus, the returned path will not contain any symbolic links, but the named
399 + file and some of its parent directories may be missing.

```

```

384 400
385 401 .. note::
386 402     This function emulates the operating system's procedure for making a path
@@ -399,6 +415,15 @@ the :mod:`glob` module.)

```

```

399 415 .. versionchanged:: 3.10
400 416     The *strict* parameter was added.
401 417

```

```

418 + .. versionchanged:: next
419 +     The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
420 +     was added.
421 +
422 + .. data:: ALLOW_MISSING
423 +
424 +     Special value used for the *strict* argument in :func:`realpath`.
425 +
426 + .. versionadded:: next

```

```

402 427
403 428 .. function:: relpath(path, start=os.curdir)
404 429

```

Doc/library/tarfile.rst



```
@@ -249,6 +249,15 @@ The :mod:`tarfile` module defines the following
exceptions:
```

```

249 249     Raised to refuse extracting a symbolic link pointing outside the
destination

```

```

250 250     directory.

```

```

251 251

```

```

252 + .. exception:: LinkFallbackError

```

```

253 +

```

```

254 +     Raised to refuse emulating a link (hard or symbolic) by extracting another
255 +     archive member, when that member would be rejected by the filter location.

```

```

256 +     The exception that was raised to reject the replacement member is
available

```

257	+	as <code>:attr:~!BaseException.__context__`.</code>
258	+	
259	+	<code>.. versionadded:: next</code>
260	+	
252	261	
253	262	The following constants are available at the module level:
254	263	
		@@ -1039,6 +1048,12 @@ reused in custom filters:
1039	1048	Implements the <code>``'data'``</code> filter.
1040	1049	In addition to what <code>``tar_filter``</code> does:
1041	1050	
1051	+	- Normalize link targets ( <code>:attr:~TarInfo.linkname`)</code> using
1052	+	<code>:func:~os.path.normpath`.</code>
1053	+	Note that this removes internal <code>``..``</code> components, which may change the
1054	+	meaning of the link if the path in <code>:attr:~!TarInfo.linkname`</code> traverses
1055	+	symbolic links.
1056	+	
1042	1057	- <code>:ref:~Refuse &lt;tarfile-extraction-refuse&gt;`</code> to extract links (hard or soft)
1043	1058	that link to absolute paths, or ones that link outside the destination.
1044	1059	
		@@ -1067,6 +1082,10 @@ reused in custom filters:
1067	1082	
1068	1083	Return the modified <code>``TarInfo``</code> member.
1069	1084	
1085	+	<code>.. versionchanged:: next</code>
1086	+	
1087	+	Link targets are now normalized.
1088	+	
1070	1089	
1071	1090	<code>.. _tarfile-extraction-refuse:</code>
1072	1091	
		@@ -1093,6 +1112,7 @@ Here is an incomplete list of things to consider:
1093	1112	* Extract to a <code>:func:~new temporary directory &lt;tempfile.mkdtemp&gt;`</code>
1094	1113	to prevent e.g. exploiting pre-existing links, and to make it easier to
1095	1114	clean up after a failed extraction.
1115	+	* Disallow symbolic links if you do not need the functionality.
1096	1116	* When working with untrusted data, use external (e.g. OS-level) limits on
1097	1117	disk, memory and CPU usage.

1098 1118 \* Check filenames against an allow-list of characters



Doc/whatsnew/3.12.rst



@@ -2320,3 +2320,37 @@ sys

```

2320 2320 * The previously undocumented special function :func:`sys.getobjects`,
2321 2321 which only exists in specialized builds of Python, may now return objects
2322 2322 from other interpreters than the one it's called in.
2323 +
2324 +
2325 + Notable changes in 3.12.10
2326 + =====
2327 +
2328 + os.path
2329 + -----
2330 +
2331 + * The strict parameter to :func:`os.path.realpath` accepts a new value,
2332 + :data:`os.path.ALLOW_MISSING`.
2333 + If used, errors other than :exc:`FileNotFoundError` will be re-raised;
2334 + the resulting path can be missing but it will be free of symlinks.
2335 + (Contributed by Petr Viktorin for :cve:`2025-4517`.)
2336 +
2337 + tarfile
2338 + -----
2339 +
2340 + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
2341 + to
2342 + avoid path traversal attacks.
2343 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2025-4138`.)
2344 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
2345 + attributes
2346 + when a directory was removed or replaced by another kind of file.
2347 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2024-12718`.)
2348 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2349 + now (re-)apply the extraction filter when substituting a link (hard or
2350 + symbolic) with a copy of another archive member, and when fixing up
2351 + directory attributes.
2352 + The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
2353 + (Contributed by Petr Viktorin for :cve:`2025-4330` and :cve:`2024-12718`.)
2354 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`

```

```

2353 + no longer extract rejected members when
2354 + :func:~tarfile.TarFile.errorlevel` is zero.
2355 + (Contributed by Matt Prodan and Petr Viktorin in :gh:`112887`
2356 + and :cve:`2025-4435`.)

```

Lib/genericpath.py

```

↑... @@ -8,7 +8,7 @@
8 8
9 9     __all__ = ['commonprefix', 'exists', 'getatime', 'getctime', 'getmtime',
10 10             'getsize', 'isdir', 'isfile', 'islink', 'samefile', 'sameopenfile',
11 11             'samestat']
11 11 +         'samestat', 'ALLOW_MISSING']
12 12
13 13
14 14     # Does a path exist?
↓...
↑... @@ -165,3 +165,12 @@ def _check_arg_types(funcname, *args):
165 165             f'os.PathLike object, not
           {s.__class__.__name__!r}') from None
166 166         if hasstr and hasbytes:
167 167             raise TypeError("Can't mix strings and bytes in path components") from
           None
168 +
169 + # A singleton with a true boolean value.
170 + @object.__new__
171 + class ALLOW_MISSING:
172 +     """Special value for use in realpath()."""
173 +     def __repr__(self):
174 +         return 'os.path.ALLOW_MISSING'
175 +     def __reduce__(self):
176 +         return self.__class__.__name__

```

Lib/ntpath.py

```

↑... @@ -30,7 +30,8 @@
30 30             "ismount", "expanduser", "expandvars", "normpath", "abspath",
31 31             "curdir", "pardir", "sep", "pathsep", "defpath", "altsep",
32 32
           "extsep", "devnull", "realpath", "supports_unicode_filenames", "relpath",
33 33 -         "samefile", "sameopenfile", "samestat", "commonpath", "isjunction"]

```

```

33 +         "samefile", "sameopenfile", "samestat", "commonpath", "isjunction",
34 +         "ALLOW_MISSING"]
34 35
35 36     def _get_bothseps(path):
36 37         if isinstance(path, bytes):
37 38             return path
38 39
39 40     @@ -609,9 +610,10 @@ def abspath(path):
40 41
41 42         from nt import _getfinalpathname, readlink as _nt_readlink
42 43     except ImportError:
43 44         # realpath is a no-op on systems without _getfinalpathname support.
44 45
45 46     -     realpath = abspath
46 47
47 48     +     def realpath(path, *, strict=False):
48 49         +         return abspath(path)
49 50
50 51     else:
51 52
52 53     -     def _readlink_deep(path):
53 54
54 55     +     def _readlink_deep(path, ignored_error=OSError):
55 56
56 57         # These error codes indicate that we should stop reading links and
57 58         # return the path we currently have.
58 59         # 1: ERROR_INVALID_FUNCTION
59 60
60 61     @@ -644,7 +646,7 @@ def _readlink_deep(path):
61 62
62 63         path = old_path
63 64
64 65         break
65 66
66 67         path = normpath(join(dirname(old_path), path))
67 68
68 69     -     except OSError as ex:
69 70
70 71     +     except ignored_error as ex:
71 72
72 73         if ex.winerror in allowed_winerror:
73 74
74 75             break
75 76
76 77             raise
77 78
78 79     @@ -653,7 +655,7 @@ def _readlink_deep(path):
79 80
80 81             break
81 82
82 83         return path
83 84
84 85
85 86     -     def _getfinalpathname_nonstrict(path):
86 87
87 88     +     def _getfinalpathname_nonstrict(path, ignored_error=OSError):
88 89
89 90         # These error codes indicate that we should stop resolving the path
90 91         # and return the value we currently have.
91 92         # 1: ERROR_INVALID_FUNCTION
92 93
93 94     @@ -680,17 +682,18 @@ def _getfinalpathname_nonstrict(path):
94 95
95 96         try:

```

```

681 683         path = _getfinalpathname(path)
682 684         return join(path, tail) if tail else path
683 -         except OSError as ex:
685 +         except ignored_error as ex:
684 686             if ex.winerror not in allowed_winerror:
685 687                 raise
686 688             try:
687 689                 # The OS could not resolve this path fully, so we attempt
688 690                 # to follow the link ourselves. If we succeed, join the
689 691                 tail
690                 # and return.
690 -         new_path = _readlink_deep(path)
692 +         new_path = _readlink_deep(path,
693 +                                 ignored_error=ignored_error)
691 694             if new_path != path:
692 695                 return join(new_path, tail) if tail else new_path
693 -         except OSError:
696 +         except ignored_error:
694 697             # If we fail to readlink(), let's keep traversing
695 698             pass
696 699             path, name = split(path)
@@ -721,24 +724,32 @@ def realpath(path, *, strict=False):
721 724         if normcase(path) == normcase(devnull):
722 725             return '\\\\.\NUL'
723 726         had_prefix = path.startswith(prefix)
727 +
728 +         if strict is ALLOW_MISSING:
729 +             ignored_error = FileNotFoundError
730 +             strict = True
731 +         elif strict:
732 +             ignored_error = ()
733 +         else:
734 +             ignored_error = OSError
735 +
724 736         if not had_prefix and not isabs(path):
725 737             path = join(cwd, path)
726 738         try:
727 739             path = _getfinalpathname(path)
728 740             initial_winerror = 0

```

```

729 741         except ValueError as ex:
730 742             # gh-106242: Raised for embedded null characters
731 -         # In strict mode, we convert into an OSError.
743 +         # In strict modes, we convert into an OSError.
732 744             # Non-strict mode returns the path as-is, since we've already
733 745             # made it absolute.
734 746             if strict:
735 747                 raise OSError(str(ex)) from None
736 748             path = normpath(path)
737 -         except OSError as ex:
738 -             if strict:
739 -                 raise
749 +         except ignored_error as ex:
740 750             initial_winerror = ex.winerror
741 -         path = _getfinalpathname_nonstrict(path)
751 +         path = _getfinalpathname_nonstrict(path,
752 +             ignored_error=ignored_error)
742 753             # The path returned by _getfinalpathname will always start with \\?\ -
743 754             # strip off that prefix unless it was already provided on the original
744 755             # path.

```

Lib/posixpath.py

```

@@ -35,7 +35,7 @@
35 35         "samefile", "sameopenfile", "samestat",
36 36         "curdir", "pardir", "sep", "pathsep", "defpath", "altsep", "extsep",
37 37         "devnull", "realpath", "supports_unicode_filenames", "relpath",
38 -         "commonpath", "isjunction"]
38 +         "commonpath", "isjunction", "ALLOW_MISSING"]
39 39
40 40
41 41     def _get_sep(path):
@@ -438,6 +438,15 @@ def _joinrealpath(path, rest, strict, seen):
438 438         sep = '/'
439 439         curdir = '.'
440 440         pardir = '..'
441 +         getcwd = os.getcwd
442 +         if strict is ALLOW_MISSING:
443 +             ignored_error = FileNotFoundError

```

```

444 + elif strict:
445 +     ignored_error = ()
446 + else:
447 +     ignored_error = OSError
448 +
449 + maxlinks = None

441 450
442 451     if isabs(rest):
443 452         rest = rest[1:]
@@ -460,9 +469,7 @@ def _joinrealpath(path, rest, strict, seen):
460 469         newpath = join(path, name)
461 470         try:
462 471             st = os.lstat(newpath)
463 -         except OSError:
464 -             if strict:
465 -                 raise
472 +         except ignored_error:
466 473             is_link = False
467 474         else:
468 475             is_link = stat.S_ISLNK(st.st_mode)

```

Lib/tarfile.py

```

@@ -752,10 +752,22 @@ def __init__(self, tarinfo, path):
752 752         super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
753 753             + 'which is outside the destination')
754 754
755 + class LinkFallbackError(FilterError):
756 +     def __init__(self, tarinfo, path):
757 +         self.tarinfo = tarinfo
758 +         self._path = path
759 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
760 +             + f'copy of {path!r}, which was rejected')
761 +
762 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
763 + # we consider to be issues with the argument, rather than a bug in the
764 + # filter function
765 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
766 +
755 767     def _get_filtered_attrs(member, dest_path, for_data=True):

```

```

756     768         new_attrs = {}
757     769         name = member.name
758     -   dest_path = os.path.realpath(dest_path)
770     +   dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
759     771         # Strip leading / (tar's directory separator) from filenames.
760     772         # Include os.sep (target OS directory separator) as well.
761     773         if name.startswith('/', os.sep):
@@ -765,7 +777,8 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
765     777             # For example, 'C:/foo' on Windows.
766     778             raise AbsolutePathError(member)
767     779             # Ensure we stay in the destination
768     -   target_path = os.path.realpath(os.path.join(dest_path, name))
780     +   target_path = os.path.realpath(os.path.join(dest_path, name),
781     +                                 strict=os.path.ALLOW_MISSING)
769     782         if os.path.commonpath([target_path, dest_path]) != dest_path:
770     783             raise OutsideDestinationError(member, target_path)
771     784         # Limit permissions (no high bits, and go-w)
@@ -803,14 +816,18 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
803     816         if member.islnk() or member.issym():
804     817             if os.path.isabs(member.linkname):
805     818                 raise AbsoluteLinkError(member)
819     +   normalized = os.path.normpath(member.linkname)
820     +   if normalized != member.linkname:
821     +       new_attrs['linkname'] = normalized
806     822         if member.issym():
807     823             target_path = os.path.join(dest_path,
808     824                                     os.path.dirname(name),
809     825                                     member.linkname)
810     826         else:
811     827             target_path = os.path.join(dest_path,
812     828                                     member.linkname)
813     -   target_path = os.path.realpath(target_path)
829     +   target_path = os.path.realpath(target_path,
830     +                                 strict=os.path.ALLOW_MISSING)
814     831         if os.path.commonpath([target_path, dest_path]) != dest_path:
815     832             raise LinkOutsideDestinationError(member, target_path)
816     833         return new_attrs

```

	↓		@@ -2291,30 +2308,58 @@
	↑		def extractall(self, path=".", members=None, *, numeric_owner=False,
2291	2308		members = self
2292	2309		
2293	2310		for member in members:
2294	-		tarinfo = self._get_extract_tarinfo(member, filter_function, path)
	2311	+	tarinfo, unfiltered = self._get_extract_tarinfo( 2312 + member, filter_function, path)
2295	2313		if tarinfo is None:
2296	2314		continue
2297	2315		if tarinfo.isdir():
2298	2316		# For directories, delay setting attributes until later,
2299	2317		# since permissions can interfere with extraction and
2300	2318		# extracting contents can reset mtime.
2301	-		directories.append(tarinfo)
	2319	+	directories.append(unfiltered)
2302	2320		self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(), 2303 - numeric_owner=numeric_owner)
	2321	+	numeric_owner=numeric_owner, 2322 + filter_function=filter_function)
2304	2323		
2305	2324		# Reverse sort directories.
2306	2325		directories.sort(key=lambda a: a.name, reverse=True)
2307	2326		
	2327	+	
2308	2328		# Set correct owner, mtime and filemode on directories.
2309	-		for tarinfo in directories:
2310	-		dirpath = os.path.join(path, tarinfo.name)
	2329	+	for unfiltered in directories:
2311	2330		try:
	2331	+	# Need to re-apply any filter, to take the *current* filesystem
	2332	+	# state into account.
	2333	+	try:
	2334	+	tarinfo = filter_function(unfiltered, path)
	2335	+	except _FILTER_ERRORS as exc:
	2336	+	self._log_no_directory_fixup(unfiltered, repr(exc))
	2337	+	continue
	2338	+	if tarinfo is None:

```

2339 +         self._log_no_directory_fixup(unfiltered,
2340 +                                     'excluded by filter')
2341 +         continue
2342 +         dirpath = os.path.join(path, tarinfo.name)
2343 +         try:
2344 +             lstat = os.lstat(dirpath)
2345 +         except FileNotFoundError:
2346 +             self._log_no_directory_fixup(tarinfo, 'missing')
2347 +             continue
2348 +         if not stat.S_ISDIR(lstat.st_mode):
2349 +             # This is no longer a directory; presumably a later
2350 +             # member overwrote the entry.
2351 +             self._log_no_directory_fixup(tarinfo, 'not a directory')
2352 +             continue
2312 2353         self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
2313 2354         self.utime(tarinfo, dirpath)
2314 2355         self.chmod(tarinfo, dirpath)
2315 2356         except ExtractError as e:
2316 2357             self._handle_nonfatal_error(e)
2317 2358
2359 +     def _log_no_directory_fixup(self, member, reason):
2360 +         self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
2361 +                   (member.name, reason))
2362 +
2318 2363     def extract(self, member, path="", set_attrs=True, *,
                numeric_owner=False,
2319 2364                 filter=None):
2320 2365         """Extract a member from the archive to the current working
                directory,
@@ -2330,41 +2375,56 @@ def extract(self, member, path="",
                set_attrs=True, *, numeric_owner=False,
2330 2375                 String names of common filters are accepted.
2331 2376         """
2332 2377         filter_function = self._get_filter_function(filter)
2333 -         tarinfo = self._get_extract_tarinfo(member, filter_function, path)
2378 +         tarinfo, unfiltered = self._get_extract_tarinfo(
2379 +             member, filter_function, path)
2334 2380         if tarinfo is not None:
2335 2381             self._extract_one(tarinfo, path, set_attrs, numeric_owner)
2336 2382

```

```

2337 2383         def _get_extract_tarinfo(self, member, filter_function, path):
2338 -         """Get filtered TarInfo (or None) from member, which might be a
          str"""
2384 +         """Get (filtered, unfiltered) TarInfos from *member*
2385 +
2386 +         *member* might be a string.
2387 +
2388 +         Return (None, None) if not found.
2389 +         """
2390 +
2391 2391         if isinstance(member, str):
2340 -             tarinfo = self.getmember(member)
2392 +             unfiltered = self.getmember(member)
2341 2393         else:
2342 -             tarinfo = member
2394 +             unfiltered = member
2343 2395
2344 -             unfiltered = tarinfo
2396 +             filtered = None
2345 2397         try:
2346 -             tarinfo = filter_function(tarinfo, path)
2398 +             filtered = filter_function(unfiltered, path)
2347 2399         except (OSError, FilterError) as e:
2348 2400             self._handle_fatal_error(e)
2349 2401         except ExtractError as e:
2350 2402             self._handle_nonfatal_error(e)
2351 -             if tarinfo is None:
2403 +             if filtered is None:
2352 2404                 self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2353 -             return None
2405 +             return None, None
2406 +
2354 2407         # Prepare the link target for makelink().
2355 -             if tarinfo.islnk():
2356 -                 tarinfo = copy.copy(tarinfo)
2357 -                 tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2358 -             return tarinfo
2408 +             if filtered.islnk():
2409 +                 filtered = copy.copy(filtered)
2410 +                 filtered._link_target = os.path.join(path, filtered.linkname)

```

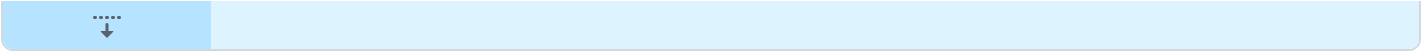
2411	+	<code>return filtered, unfiltered</code>
2359	2412	
2360	-	<code>def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):</code>
2361	-	<code>    """Extract from filtered tarinfo to disk"""</code>
2413	+	<code>def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,</code>
2414	+	<code>    filter_function=None):</code>
2415	+	<code>    """Extract from filtered tarinfo to disk.</code>
2416	+	
2417	+	<code>        filter_function is only used when extracting a *different*</code>
2418	+	<code>        member (e.g. as fallback to creating a symlink)</code>
2419	+	<code>    """</code>
2362	2420	<code>self._check("r")</code>
2363	2421	
2364	2422	<code>try:</code>
2365	2423	<code>    self._extract_member(tarinfo, os.path.join(path, tarinfo.name),</code>
2366	2424	<code>        set_attrs=set_attrs,</code>
2367	-	<code>        numeric_owner=numeric_owner)</code>
2425	+	<code>        numeric_owner=numeric_owner,</code>
2426	+	<code>        filter_function=filter_function,</code>
2427	+	<code>        extraction_root=path)</code>
2368	2428	<code>except OSError as e:</code>
2369	2429	<code>    self._handle_fatal_error(e)</code>
2370	2430	<code>except ExtractError as e:</code>
		<code>@@ -2422,9 +2482,13 @@ def extractfile(self, member):</code>
2422	2482	<code>    return None</code>
2423	2483	
2424	2484	<code>def _extract_member(self, tarinfo, targetpath, set_attrs=True,</code>
2425	-	<code>    numeric_owner=False):</code>
2426	-	<code>    """Extract the TarInfo object tarinfo to a physical</code>
2485	+	<code>        numeric_owner=False, *, filter_function=None,</code>
2486	+	<code>        extraction_root=None):</code>
2487	+	<code>    """Extract the filtered TarInfo object tarinfo to a physical</code>
2427	2488	<code>    file called targetpath.</code>
2489	+	
2490	+	<code>        filter_function is only used when extracting a *different*</code>
2491	+	<code>        member (e.g. as fallback to creating a symlink)</code>
2428	2492	<code>    """</code>
2429	2493	<code>    # Fetch the TarInfo object for the given name</code>
2430	2494	<code>    # and build the destination pathname, replacing</code>

		@@ -2453,7 +2517,10 @@ def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2453	2517	elif tarinfo.ischr() or tarinfo.isblk():
2454	2518	self.makedev(tarinfo, targetpath)
2455	2519	elif tarinfo.islnk() or tarinfo.issym():
2456	-	self.makelink(tarinfo, targetpath)
	2520	self.makelink_with_filter(
	2521	tarinfo, targetpath,
	2522	filter_function=filter_function,
	2523	extraction_root=extraction_root)
2457	2524	elif tarinfo.type not in SUPPORTED_TYPES:
2458	2525	self.makeunknown(tarinfo, targetpath)
2459	2526	else:
		@@ -2536,29 +2603,57 @@ def makedev(self, tarinfo, targetpath):
2536	2603	os.makedev(tarinfo.devmajor, tarinfo.devminor))
2537	2604	
2538	2605	def makelink(self, tarinfo, targetpath):
	2606	return self.makelink_with_filter(tarinfo, targetpath, None, None)
	2607	
	2608	def makelink_with_filter(self, tarinfo, targetpath,
	2609	filter_function, extraction_root):
2539	2610	"""Make a (symbolic) link called targetpath. If it cannot be created
2540	2611	(platform limitation), we try to make a copy of the referenced file
2541	2612	instead of a link.
	2613	
	2614	filter_function is only used when extracting a *different*
	2615	member (e.g. as fallback to creating a link).
2542	2616	"""
	2617	keyerror_to_extracterror = False
2543	2618	try:
2544	2619	# For systems that support symbolic and hard links.
2545	2620	if tarinfo.issym():
2546	2621	if os.path.lexists(targetpath):
2547	2622	# Avoid FileExistsError on following os.symlink.
2548	2623	os.unlink(targetpath)
2549	2624	os.symlink(tarinfo.linkname, targetpath)
	2625	return
2550	2626	else:
2551	2627	if os.path.exists(tarinfo._link_target):

```

2552 2628             os.link(tarinfo._link_target, targetpath)
2553 -             else:
2554 -                 self._extract_member(self._find_link_target(tarinfo),
2555 -                                     targetpath)
2629 +             return
2556 2630         except symlink_exception:
2631 +             keyerror_to_extracterror = True
2632 +
2633 +         try:
2634 +             unfiltered = self._find_link_target(tarinfo)
2635 +         except KeyError:
2636 +             if keyerror_to_extracterror:
2637 +                 raise ExtractError(
2638 +                     "unable to resolve link inside archive") from None
2639 +             else:
2640 +                 raise
2641 +
2642 +         if filter_function is None:
2643 +             filtered = unfiltered
2644 +         else:
2645 +             if extraction_root is None:
2646 +                 raise ExtractError(
2647 +                     "makelink_with_filter: if filter_function is not None, "
2648 +                     + "extraction_root must also not be None")
2557 2649         try:
2558 -             self._extract_member(self._find_link_target(tarinfo),
2559 -                                 targetpath)
2560 -         except KeyError:
2561 -             raise ExtractError("unable to resolve link inside archive")
                from None
2650 +             filtered = filter_function(unfiltered, extraction_root)
2651 +         except _FILTER_ERRORS as cause:
2652 +             raise LinkFallbackError(tarinfo, unfiltered.name) from cause
2653 +         if filtered is not None:
2654 +             self._extract_member(filtered, targetpath,
2655 +                                 filter_function=filter_function,
2656 +                                 extraction_root=extraction_root)
2562 2657
2563 2658         def chown(self, tarinfo, targetpath, numeric_owner):
2564 2659             """Set owner of targetpath according to tarinfo. If numeric_owner

```



Lib/test/test\_ntpath.py

Load Diff

Large diffs are not rendered by default.

Lib/test/test\_posixpath.py

Load Diff

Large diffs are not rendered by default.

Lib/test/test\_tarfile.py

Load Diff

Large diffs are not rendered by default.

...5-06-02-11-32-23.gh-issue-135034.RLGjbp.rst

```

@@ -0,0 +1,6 @@
1 + Fixes multiple issues that allowed ``tarfile`` extraction filters
2 + (``filter="data"`` and ``filter="tar"``) to be bypassed using crafted
3 + symlinks and hard links.
4 +
5 + Addresses :cve:`2024-12718`, :cve:`2025-4138`, :cve:`2025-4330`, and :cve:`2025-
6 + 4517`.

```

Comments 0