

python / cpython Public

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

# Commit aa9eb5f



6 people authored on Jun 3, 2025 · ✖ 78 / 88 · Partially verified

```
[3.13] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (GH-135064)

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>
```

🔗 3.13 (AcreationOS-Linux/python#2, #135064) · 📦 v3.13.13 ... v3.13.4

1 parent 9f3d999 commit aa9eb5f 📄

**11 files changed** +965 -164

📄 ↑ Top ⚙️

🔍 Filter files... ☰

- ✓ 📁 Doc
  - ✓ 📁 library
    - 📄 os.path.rst
    - 📄 tarfile.rst
  - ✓ 📁 whatsnew
    - 📄 3.13.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



▼ Doc/library/os.path.rst



```
@@ -408,9 +408,26 @@ the :mod:`glob` module.)
408 408     system). On Windows, this function will also resolve MS-DOS (also called
      8.3)
409 409     style names such as ``C:\\PROGRA~1`` to ``C:\\Program Files``.
410 410
411 -   If a path doesn't exist or a symlink loop is encountered, and strict is
412 -   ``True``, :exc:`OSError` is raised. If strict is ``False`` these errors
413 -   are ignored, and so the result might be missing or otherwise inaccessible.
411 +   By default, the path is evaluated up to the first component that does not
412 +   exist, is a symlink loop, or whose evaluation raises :exc:`OSError`.
413 +   All such components are appended unchanged to the existing part of the path.
414 +
415 +   Some errors that are handled this way include "access denied", "not a
416 +   directory", or "bad argument to internal function". Thus, the
417 +   resulting path may be missing or inaccessible, may still contain
418 +   links or loops, and may traverse non-directories.
419 +
420 +   This behavior can be modified by keyword arguments:
421 +
422 +   If strict is ``True``, the first error encountered when evaluating the
      path is
423 +   re-raised.
424 +   In particular, :exc:`FileNotFoundError` is raised if path does not exist,
425 +   or another :exc:`OSError` if it is otherwise inaccessible.
426 +
```

```

427 + If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
428 + :exc:`FileNotFoundError` are re-raised (as with ``strict=True``).
429 + Thus, the returned path will not contain any symbolic links, but the named
430 + file and some of its parent directories may be missing.

414 431
415 432     .. note::
416 433         This function emulates the operating system's procedure for making a path
@@ -429,6 +446,15 @@ the :mod:`glob` module.)
429 446     .. versionchanged:: 3.10
430 447         The *strict* parameter was added.
431 448

449 +     .. versionchanged:: next
450 +         The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
451 +         was added.
452 +
453 + .. data:: ALLOW_MISSING
454 +
455 +     Special value used for the *strict* argument in :func:`realpath`.
456 +
457 +     .. versionadded:: next

432 458
433 459     .. function:: relpath(path, start=os.curdir)
434 460

```

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 :attr:`!BaseException.__context__`.
258 +

```

259	+	.. <code>versionadded:: next</code>
260	+	
252	261	
253	262	The following constants are available at the module level:
254	263	
⋮ ↓ ↑ ⋮		@@ -1052,6 +1061,12 @@ reused in custom filters:
1052	1061	Implements the <code>``'data'``</code> filter.
1053	1062	In addition to what <code>``tar_filter``</code> does:
1054	1063	
1064	+	- Normalize link targets ( <code>:attr:`TarInfo.linkname`</code> ) using
1065	+	<code>:func:`os.path.normpath`</code> .
1066	+	Note that this removes internal <code>``..``</code> components, which may change the
1067	+	meaning of the link if the path in <code>:attr:`!TarInfo.linkname`</code> traverses
1068	+	symbolic links.
1069	+	
1055	1070	- <code>:ref:`Refuse &lt;tarfile-extraction-refuse&gt;`</code> to extract links (hard or soft)
1056	1071	that link to absolute paths, or ones that link outside the destination.
1057	1072	
⋮ ↓ ↑ ⋮		@@ -1080,6 +1095,10 @@ reused in custom filters:
1080	1095	
1081	1096	Return the modified <code>``TarInfo``</code> member.
1082	1097	
1098	+	.. <code>versionchanged:: next</code>
1099	+	
1100	+	Link targets are now normalized.
1101	+	
1083	1102	
1084	1103	.. <code>_tarfile-extraction-refuse:</code>
1085	1104	
⋮ ↕ ⋮		@@ -1106,6 +1125,7 @@ Here is an incomplete list of things to consider:
1106	1125	* Extract to a <code>:func:`new temporary directory &lt;tempfile.mkdtemp&gt;`</code>
1107	1126	to prevent e.g. exploiting pre-existing links, and to make it easier to
1108	1127	clean up after a failed extraction.
1128	+	* Disallow symbolic links if you do not need the functionality.
1109	1129	* When working with untrusted data, use external (e.g. OS-level) limits on
1110	1130	disk, memory and CPU usage.
1111	1131	* Check filenames against an allow-list of characters
⋮ ↓ ⋮		

Doc/whatsnew/3.13.rst



↑

@@ -2829,3 +2829,36 @@ sys

2829 2829 \* The previously undocumented special function `:func:`sys.getobjects``,  
 2830 2830 which only exists in specialized builds of Python, may now return objects  
 2831 2831 from other interpreters than the one it's called in.

2832 +

2833 + Notable changes in 3.13.4

2834 + =====

2835 +

2836 + `os.path`

2837 + -----

2838 +

2839 + \* The `*strict*` parameter to `:func:`os.path.realpath`` accepts a new value,  
 2840 + `:data:`os.path.ALLOW_MISSING``.

2841 + If used, errors other than `:exc:`FileNotFoundError`` will be re-raised;  
 2842 + the resulting path can be missing but it will be free of symlinks.

2843 + (Contributed by Petr Viktorin for `:cve:`2025-4517``.)

2844 +

2845 + `tarfile`

2846 + -----

2847 +

2848 + \* `:func:`~tarfile.data_filter`` now normalizes symbolic link targets in order  
 to

2849 + avoid path traversal attacks. Add comment More actions

2850 + (Contributed by Petr Viktorin in `:gh:`127987`` and `:cve:`2025-4138``.)

2851 + \* `:func:`~tarfile.TarFile.extractall`` now skips fixing up directory  
 attributes

2852 + when a directory was removed or replaced by another kind of file.

2853 + (Contributed by Petr Viktorin in `:gh:`127987`` and `:cve:`2024-12718``.)

2854 + \* `:func:`~tarfile.TarFile.extract`` and `:func:`~tarfile.TarFile.extractall``

2855 + now (re-)apply the extraction filter when substituting a link (hard or  
 2856 + symbolic) with a copy of another archive member, and when fixing up  
 2857 + directory attributes.

2858 + The former raises a new exception, `:exc:`~tarfile.LinkFallbackError``.

2859 + (Contributed by Petr Viktorin for `:cve:`2025-4330`` and `:cve:`2024-12718``.)

2860 + \* `:func:`~tarfile.TarFile.extract`` and `:func:`~tarfile.TarFile.extractall``

2861 + no longer extract rejected members when

2862 + `:func:`~tarfile.TarFile.errorlevel`` is zero.

2863 + (Contributed by Matt Prodan and Petr Viktorin in `:gh:`112887``)

2864 + and :cve:`2025-4435`.)

## Lib/genericpath.py

...

↑

@@ -8,7 +8,7 @@

```

8      8
9      9     __all__ = ['commonprefix', 'exists', 'getatime', 'getctime', 'getmtime',
10     10         'getsize', 'isdevdrive', 'isdir', 'isfile', 'isjunction', 'islink',
11     -     'lexists', 'samefile', 'sameopenfile', 'samestat']
11     +     'lexists', 'samefile', 'sameopenfile', 'samestat', 'ALLOW_MISSING']

```

12 12

13 13

14 14 # Does a path exist?

↓

↑

@@ -189,3 +189,12 @@ def \_check\_arg\_types(funcname, \*args):

```

189 189         f'os.PathLike object, not
        {s.__class__.__name__!r}') from None
190 190         if hasstr and hasbytes:
191 191             raise TypeError("Can't mix strings and bytes in path components") from
        None
192 +
193 + # A singleton with a true boolean value.
194 + @object.__new__
195 + class ALLOW_MISSING:
196 +     """Special value for use in realpath()."""
197 +     def __repr__(self):
198 +         return 'os.path.ALLOW_MISSING'
199 +     def __reduce__(self):
200 +         return self.__class__.__name__

```

## Lib/ntpath.py

...

↑

@@ -29,7 +29,7 @@

```

29 29         "abspath", "curdir", "pardir", "sep", "pathsep", "defpath", "altsep",
30 30
        "extsep", "devnull", "realpath", "supports_unicode_filenames", "relpath",
31 31         "samefile", "sameopenfile", "samestat", "commonpath", "isjunction",
32  -     "isdevdrive"]
32  +     "isdevdrive", "ALLOW_MISSING"]

```

33 33

34 34 def \_get\_bothseps(path):

```

35     35         if isinstance(path, bytes):
36         ↓
37         ↑
38         @@ -601,9 +601,10 @@ def abspath(path):
601     601             from nt import _findfirstfile, _getfinalpathname, readlink as _nt_readlink
602     602         except ImportError:
603     603             # realpath is a no-op on systems without _getfinalpathname support.
604     -     604         realpath = abspath
605     +     604         def realpath(path, *, strict=False):
606     +     605             return abspath(path)
607     606         else:
608     -     606         def _readlink_deep(path):
609     +     607         def _readlink_deep(path, ignored_error=OSError):
610     608             # These error codes indicate that we should stop reading links and
611     609             # return the path we currently have.
612     610             # 1: ERROR_INVALID_FUNCTION
613         ↓
614         ↑
615         @@ -636,7 +637,7 @@ def _readlink_deep(path):
636     637                 path = old_path
637     638                 break
638     639                 path = normpath(join(dirname(old_path), path))
639     -     639         except OSError as ex:
640     +     640         except ignored_error as ex:
641     641             if ex.winerror in allowed_winerror:
642     642                 break
643     643                 raise
644         ↕
645         @@ -645,7 +646,7 @@ def _readlink_deep(path):
645     646                 break
646     647                 return path
647     648
648     -     648         def _getfinalpathname_nonstrict(path):
649     +     649         def _getfinalpathname_nonstrict(path, ignored_error=OSError):
650     650             # These error codes indicate that we should stop resolving the path
651     651             # and return the value we currently have.
652     652             # 1: ERROR_INVALID_FUNCTION
653         ↓
654         ↑
655         @@ -673,25 +674,26 @@ def _getfinalpathname_nonstrict(path):
673     674                 try:
674     675                     path = _getfinalpathname(path)
675     676                     return join(path, tail) if tail else path
676     -     676         except OSError as ex:

```

```

677 +         except ignored_error as ex:
677 678             if ex.winerror not in allowed_winerror:
678 679                 raise
679 680             try:
680 681                 # The OS could not resolve this path fully, so we attempt
681 682                 # to follow the link ourselves. If we succeed, join the
682 683                 tail
682 683                 # and return.
683 -         new_path = _readlink_deep(path)
684 +         new_path = _readlink_deep(path,
685 +                                 ignored_error=ignored_error)
684 686             if new_path != path:
685 687                 return join(new_path, tail) if tail else new_path
686 -         except OSError:
688 +         except ignored_error:
687 689             # If we fail to readlink(), let's keep traversing
688 690             pass
689 691             # If we get these errors, try to get the real name of the file
690 692             without accessing it.
690 692             if ex.winerror in (1, 5, 32, 50, 87, 1920, 1921):
691 693                 try:
692 694                     name = _findfirstfile(path)
693 695                     path, _ = split(path)
694 -         except OSError:
696 +         except ignored_error:
695 697             path, name = split(path)
696 698             else:
697 699             path, name = split(path)
698 699
@@ -721,24 +723,32 @@ def realpath(path, *, strict=False):
721 723             if normcase(path) == devnull:
722 724                 return '\\\\.\\NUL'
723 725             had_prefix = path.startswith(prefix)
726 +
727 +             if strict is ALLOW_MISSING:
728 +                 ignored_error = FileNotFoundError
729 +                 strict = True
730 +             elif strict:
731 +                 ignored_error = ()
732 +             else:

```

```

733 +         ignored_error = OSError
734 +
724 735         if not had_prefix and not isabs(path):
725 736             path = join(cwd, path)
726 737         try:
727 738             path = _getfinalpathname(path)
728 739             initial_winerror = 0
729 740         except ValueError as ex:
730 741             # gh-106242: Raised for embedded null characters
731 -             # In strict mode, we convert into an OSError.
742 +             # In strict modes, we convert into an OSError.
732 743             # Non-strict mode returns the path as-is, since we've already
733 744             # made it absolute.
734 745             if strict:
735 746                 raise OSError(str(ex)) from None
736 747             path = normpath(path)
737 -             except OSError as ex:
738 -                 if strict:
739 -                     raise
748 +             except ignored_error as ex:
740 749                 initial_winerror = ex.winerror
741 -                 path = _getfinalpathname_nonstrict(path)
750 +                 path = _getfinalpathname_nonstrict(path,
751 +                 ignored_error=ignored_error)
742 752             # The path returned by _getfinalpathname will always start with \\?\ -
743 753             # strip off that prefix unless it was already provided on the original
744 754             # path.

```

Lib/posixpath.py

```

@@ -36,7 +36,7 @@
36 36         "samefile", "sameopenfile", "samestat",
37 37         "curdir", "pardir", "sep", "pathsep", "defpath", "altsep", "extsep",
38 38         "devnull", "realpath", "supports_unicode_filenames", "relpath",
39 -         "commonpath", "isjunction", "isdevdrive"]
39 +         "commonpath", "isjunction", "isdevdrive", "ALLOW_MISSING"]
40 40
41 41
42 42     def _get_sep(path):

```

```

↑
@@ -402,6 +402,15 @@ def realpath(filename, *, strict=False):
402 402         curdir = '.'
403 403         pardir = '..'
404 404         getcwd = os.getcwd
405 +         if strict is ALLOW_MISSING:
406 +             ignored_error = FileNotFoundError
407 +             strict = True
408 +         elif strict:
409 +             ignored_error = ()
410 +         else:
411 +             ignored_error = OSError
412 +
413 +         maxlinks = None
405 414
406 415         # The stack of unresolved path parts. When popped, a special value of None
407 416         # indicates that a symlink target has been resolved, and that the original
↓
@@ -462,25 +471,28 @@ def realpath(filename, *, strict=False):
↑
462 471             path = newpath
463 472             continue
464 473             target = os.readlink(newpath)
465 -         except OSError:
466 -             if strict:
467 -                 raise
468 -             path = newpath
474 +         except ignored_error:
475 +             pass
476 +         else:
477 +             # Resolve the symbolic link
478 +             if target.startswith(sep):
479 +                 # Symlink target is absolute; reset resolved path.
480 +                 path = sep
481 +             if maxlinks is None:
482 +                 # Mark this symlink as seen but not fully resolved.
483 +                 seen[newpath] = None
484 +                 # Push the symlink path onto the stack, and signal its
specialness
485 +                 # by also pushing None. When these entries are popped, we'll
486 +                 # record the fully-resolved symlink target in the 'seen'
mapping.

```

```

487 +         rest.append(newpath)
488 +         rest.append(None)
489 +         # Push the unresolved symlink target parts onto the stack.
490 +         target_parts = target.split(sep)[::-1]
491 +         rest.extend(target_parts)
492 +         part_count += len(target_parts)
493 +         continue
470 -         # Resolve the symbolic link
471 -         seen[newpath] = None # not resolved symlink
472 -         if target.startswith(sep):
473 -             # Symlink target is absolute; reset resolved path.
474 -             path = sep
475 -             # Push the symlink path onto the stack, and signal its specialness by
476 -             # also pushing None. When these entries are popped, we'll record the
477 -             # fully-resolved symlink target in the 'seen' mapping.
478 -             rest.append(newpath)
479 -             rest.append(None)
480 -             # Push the unresolved symlink target parts onto the stack.
481 -             target_parts = target.split(sep)[::-1]
482 -             rest.extend(target_parts)
483 -             part_count += len(target_parts)
494 +             # An error occurred and was ignored.
495 +             path = newpath
484 496
485 497         return path
486 498

```

Lib/tarfile.py

```

@@ -68,7 +68,7 @@
68 68         "DEFAULT_FORMAT", "open", "fully_trusted_filter", "data_filter",
69 69         "tar_filter", "FilterError", "AbsoluteLinkError",
70 70         "OutsideDestinationError", "SpecialFileError",
        "AbsolutePathError",
71 -         "LinkOutsideDestinationError"]
71 +         "LinkOutsideDestinationError", "LinkFallbackError"]
72 72
73 73
74 74     #-----

```

	⬆️	@@ -755,10 +755,22 @@ def __init__(self, tarinfo, path):
755	755	super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
756	756	+ 'which is outside the destination')
757	757	
758		+ class LinkFallbackError(FilterError):
759		+     def __init__(self, tarinfo, path):
760		+         self.tarinfo = tarinfo
761		+         self._path = path
762		+         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
763		+             + f'copy of {path!r}, which was rejected')
764		+
765		+ # Errors caused by filters -- both "fatal" and "non-fatal" -- that
766		+ # we consider to be issues with the argument, rather than a bug in the
767		+ # filter function
768		+ _FILTER_ERRORS = (FilterError, OSError, ExtractError)
769		+
758	770	def _get_filtered_attrs(member, dest_path, for_data=True):
759	771	new_attrs = {}
760	772	name = member.name
761		-    dest_path = os.path.realpath(dest_path)
773		+    dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
762	774	# Strip leading / (tar's directory separator) from filenames.
763	775	# Include os.sep (target OS directory separator) as well.
764	776	if name.startswith('/'): os.sep):
		@@ -768,7 +780,8 @@ def _get_filtered_attrs(member, dest_path,
		for_data=True):
768	780	# For example, 'C:/foo' on Windows.
769	781	raise AbsolutePathError(member)
770	782	# Ensure we stay in the destination
771		-    target_path = os.path.realpath(os.path.join(dest_path, name))
783		+    target_path = os.path.realpath(os.path.join(dest_path, name),
784		+        strict=os.path.ALLOW_MISSING)
772	785	if os.path.commonpath([target_path, dest_path]) != dest_path:
773	786	raise OutsideDestinationError(member, target_path)
774	787	# Limit permissions (no high bits, and go-w)
		@@ -806,14 +819,18 @@ def _get_filtered_attrs(member, dest_path,
		for_data=True):
806	819	if member.islnk() or member.issym():
807	820	if os.path.isabs(member.linkname):
808	821	raise AbsoluteLinkError(member)

822	+	normalized = os.path.normpath(member.linkname)
823	+	if normalized != member.linkname:
824	+	new_attrs['linkname'] = normalized
809	825	if member.issym():
810	826	target_path = os.path.join(dest_path,
811	827	os.path.dirname(name),
812	828	member.linkname)
813	829	else:
814	830	target_path = os.path.join(dest_path,
815	831	member.linkname)
816	-	target_path = os.path.realpath(target_path)
832	+	target_path = os.path.realpath(target_path,
833	+	strict=os.path.ALLOW_MISSING)
817	834	if os.path.commonpath([target_path, dest_path]) != dest_path:
818	835	raise LinkOutsideDestinationError(member, target_path)
819	836	return new_attrs
⋮		@@ -2323,30 +2340,58 @@ def extractall(self, path=".", members=None, *,
⋮		numeric_owner=False,
2323	2340	members = self
2324	2341	
2325	2342	for member in members:
2326	-	tarinfo = self._get_extract_tarinfo(member, filter_function,
		path)
2343	+	tarinfo, unfiltered = self._get_extract_tarinfo(
2344	+	member, filter_function, path)
2327	2345	if tarinfo is None:
2328	2346	continue
2329	2347	if tarinfo.isdir():
2330	2348	# For directories, delay setting attributes until later,
2331	2349	# since permissions can interfere with extraction and
2332	2350	# extracting contents can reset mtime.
2333	-	directories.append(tarinfo)
2351	+	directories.append(unfiltered)
2334	2352	self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
2335	-	numeric_owner=numeric_owner)
2353	+	numeric_owner=numeric_owner,
2354	+	filter_function=filter_function)
2336	2355	
2337	2356	# Reverse sort directories.
2338	2357	directories.sort(key=lambda a: a.name, reverse=True)

```

2339 2358
2359 +
2340 2360 # Set correct owner, mtime and filemode on directories.
2341 - for tarinfo in directories:
2342 -     dirpath = os.path.join(path, tarinfo.name)
2361 + for unfiltered in directories:
2343 2362     try:
2363 +         # Need to re-apply any filter, to take the *current*
filesystem
2364 +         # state into account.
2365 +         try:
2366 +             tarinfo = filter_function(unfiltered, path)
2367 +         except _FILTER_ERRORS as exc:
2368 +             self._log_no_directory_fixup(unfiltered, repr(exc))
2369 +             continue
2370 +         if tarinfo is None:
2371 +             self._log_no_directory_fixup(unfiltered,
2372 +                                         'excluded by filter')
2373 +             continue
2374 +         dirpath = os.path.join(path, tarinfo.name)
2375 +         try:
2376 +             lstat = os.lstat(dirpath)
2377 +         except FileNotFoundError:
2378 +             self._log_no_directory_fixup(tarinfo, 'missing')
2379 +             continue
2380 +         if not stat.S_ISDIR(lstat.st_mode):
2381 +             # This is no longer a directory; presumably a later
2382 +             # member overwrote the entry.
2383 +             self._log_no_directory_fixup(tarinfo, 'not a directory')
2384 +             continue
2344 2385 self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
2345 2386 self.utime(tarinfo, dirpath)
2346 2387 self.chmod(tarinfo, dirpath)
2347 2388 except ExtractError as e:
2348 2389     self._handle_nonfatal_error(e)
2349 2390
2391 + def _log_no_directory_fixup(self, member, reason):
2392 +     self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
2393 +               (member.name, reason))
2394 +

```

```

2350 2395     def extract(self, member, path="", set_attrs=True, *,
numeric_owner=False,
2351 2396         filter=None):
2352 2397         """Extract a member from the archive to the current working
directory,
@@ -2362,41 +2407,56 @@ def extract(self, member, path="",
set_attrs=True, *, numeric_owner=False,
2362 2407             String names of common filters are accepted.
2363 2408         """
2364 2409         filter_function = self._get_filter_function(filter)
2365 2410         - tarinfo = self._get_extract_tarinfo(member, filter_function, path)
2410 2411         + tarinfo, unfiltered = self._get_extract_tarinfo(
2411 2412             member, filter_function, path)
2366 2412         if tarinfo is not None:
2367 2413             self._extract_one(tarinfo, path, set_attrs, numeric_owner)
2368 2414
2369 2415     def _get_extract_tarinfo(self, member, filter_function, path):
2370 2416         - """Get filtered TarInfo (or None) from member, which might be a
str"""
2416 2417         + """Get (filtered, unfiltered) TarInfos from *member*
2417 2418         +
2418 2419         + *member* might be a string.
2419 2420         +
2420 2421         + Return (None, None) if not found.
2421 2422         +
2422 2423
2371 2423         if isinstance(member, str):
2372 2424         - tarinfo = self.getmember(member)
2424 2425         + unfiltered = self.getmember(member)
2373 2425         else:
2374 2426         - tarinfo = member
2426 2427         + unfiltered = member
2375 2427
2376 2428         - unfiltered = tarinfo
2428 2429         + filtered = None
2377 2429         try:
2378 2430         - tarinfo = filter_function(tarinfo, path)
2430 2431         + filtered = filter_function(unfiltered, path)
2379 2431         except (OSError, UnicodeEncodeError, FilterError) as e:
2380 2432             self._handle_fatal_error(e)

```

```

2381 2433         except ExtractError as e:
2382 2434             self._handle_nonfatal_error(e)
2383 -         if tarinfo is None:
2435 +         if filtered is None:
2384 2436             self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2385 -         return None
2437 +         return None, None
2438 +
2386 2439         # Prepare the link target for makelink().
2387 -         if tarinfo.islnk():
2388 -             tarinfo = copy.copy(tarinfo)
2389 -             tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2390 -         return tarinfo
2440 +         if filtered.islnk():
2441 +             filtered = copy.copy(filtered)
2442 +             filtered._link_target = os.path.join(path, filtered.linkname)
2443 +         return filtered, unfiltered
2391 2444
2392 -         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2393 -             """Extract from filtered tarinfo to disk"""
2445 +         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
2446 +             filter_function=None):
2447 +             """Extract from filtered tarinfo to disk.
2448 +
2449 +             filter_function is only used when extracting a *different*
2450 +             member (e.g. as fallback to creating a symlink)
2451 +             """
2394 2452         self._check("r")
2395 2453
2396 2454         try:
2397 2455             self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2398 2456                 set_attrs=set_attrs,
2399 -                 numeric_owner=numeric_owner)
2457 +                 numeric_owner=numeric_owner,
2458 +                 filter_function=filter_function,
2459 +                 extraction_root=path)
2400 2460         except (OSError, UnicodeEncodeError) as e:
2401 2461             self._handle_fatal_error(e)
2402 2462         except ExtractError as e:

```



```
@@ -2454,9 +2514,13 @@ def extractfile(self, member):
```

		↑	
2454	2514		return None
2455	2515		
2456	2516		def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2457	-	-	numeric_owner=False):
2458	-	-	"""Extract the TarInfo object tarinfo to a physical
	2517	+	numeric_owner=False, *, filter_function=None,
	2518	+	extraction_root=None):
	2519	+	"""Extract the filtered TarInfo object tarinfo to a physical
2459	2520		file called targetpath.
	2521	+	
	2522	+	filter_function is only used when extracting a *different*
	2523	+	member (e.g. as fallback to creating a symlink)
2460	2524		"""
2461	2525		# Fetch the TarInfo object for the given name
2462	2526		# and build the destination pathname, replacing
		↓	@@ -2485,7 +2549,10 @@ def _extract_member(self, tarinfo, targetpath,
		↑	set_attrs=True,
2485	2549		elif tarinfo.ischr() or tarinfo.isblk():
2486	2550		self.makedev(tarinfo, targetpath)
2487	2551		elif tarinfo.islnk() or tarinfo.issym():
2488	-	-	self.makelink(tarinfo, targetpath)
	2552	+	self.makelink_with_filter(
	2553	+	tarinfo, targetpath,
	2554	+	filter_function=filter_function,
	2555	+	extraction_root=extraction_root)
2489	2556		elif tarinfo.type not in SUPPORTED_TYPES:
2490	2557		self.makeunknown(tarinfo, targetpath)
2491	2558		else:
		↓	@@ -2568,29 +2635,57 @@ def makedev(self, tarinfo, targetpath):
		↑	
2568	2635		os.makedev(tarinfo.devmajor, tarinfo.devminor))
2569	2636		
2570	2637		def makelink(self, tarinfo, targetpath):
	2638	+	return self.makelink_with_filter(tarinfo, targetpath, None, None)
	2639	+	
	2640	+	def makelink_with_filter(self, tarinfo, targetpath,
	2641	+	filter_function, extraction_root):
2571	2642		"""Make a (symbolic) link called targetpath. If it cannot be created
2572	2643		(platform limitation), we try to make a copy of the referenced file

```
2573 2644         instead of a link.
2645 +
2646 +         filter_function is only used when extracting a *different*
2647 +         member (e.g. as fallback to creating a link).
2574 2648         """
2649 +         keyerror_to_extracterror = False
2575 2650     try:
2576 2651         # For systems that support symbolic and hard links.
2577 2652         if tarinfo.issym():
2578 2653             if os.path.lexists(targetpath):
2579 2654                 # Avoid FileExistsError on following os.symlink.
2580 2655                 os.unlink(targetpath)
2581 2656                 os.symlink(tarinfo.linkname, targetpath)
2657 +                 return
2582 2658     else:
2583 2659         if os.path.exists(tarinfo._link_target):
2584 2660             os.link(tarinfo._link_target, targetpath)
2585 -         else:
2586 -             self._extract_member(self._find_link_target(tarinfo),
2587 -                                 targetpath)
2661 +         return
2588 2662     except symlink_exception:
2663 +         keyerror_to_extracterror = True
2664 +
2665 +     try:
2666 +         unfiltered = self._find_link_target(tarinfo)
2667 +     except KeyError:
2668 +         if keyerror_to_extracterror:
2669 +             raise ExtractError(
2670 +                 "unable to resolve link inside archive") from None
2671 +         else:
2672 +             raise
2673 +
2674 +     if filter_function is None:
2675 +         filtered = unfiltered
2676 +     else:
2677 +         if extraction_root is None:
2678 +             raise ExtractError(
2679 +                 "makelink_with_filter: if filter_function is not None, "
2680 +                 + "extraction_root must also not be None")
```

```

2589 2681         try:
2590     -             self._extract_member(self._find_link_target(tarinfo),
2591     -                                 targetpath)
2592     -         except KeyError:
2593     -             raise ExtractError("unable to resolve link inside archive")
                from None
2682 +             filtered = filter_function(unfiltered, extraction_root)
2683 +         except _FILTER_ERRORS as cause:
2684 +             raise LinkFallbackError(tarinfo, unfiltered.name) from cause
2685 +         if filtered is not None:
2686 +             self._extract_member(filtered, targetpath,
2687 +                                 filter_function=filter_function,
2688 +                                 extraction_root=extraction_root)
2594 2689
2595 2690     def chown(self, tarinfo, targetpath, numeric_owner):
2596 2691         """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