

python / cpython Public

<> Code Issues 5k+ Pull requests 2.1k 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](#)

- ✓ 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 ``'data'`` filter.
1053	1062	In addition to what ``tar_filter`` does:
1054	1063	
1064	+	- Normalize link targets (:attr:`TarInfo.linkname`) using
1065	+	:func:`os.path.normpath`.
1066	+	Note that this removes internal ``..`` components, which may change the
1067	+	meaning of the link if the path in :attr:`!TarInfo.linkname` traverses
1068	+	symbolic links.
1069	+	
1055	1070	- :ref:`Refuse <tarfile-extraction-refuse>` 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 ``TarInfo`` 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 :func:`new temporary directory <tempfile.mkdtemp>`
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
2849 + to
2850 + avoid path traversal attacks. Add commentMore actions
2851 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2025-4138`.)
2852 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
2853 + attributes
2854 + when a directory was removed or replaced by another kind of file.
2855 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2024-12718`.)
2856 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2857 + now (re-)apply the extraction filter when substituting a link (hard or
2858 + symbolic) with a copy of another archive member, and when fixing up
2859 + directory attributes.
2860 + The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
2861 + (Contributed by Petr Viktorin for :cve:`2025-4330` and :cve:`2024-12718`.)
2862 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2863 + no longer extract rejected members when
2864 + :func:`~tarfile.TarFile.errorlevel` is zero.
2865 + (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
699 700 @@ -721,24 +723,32 @@ def realpath(path, *, strict=False):
700 701
701 702             if normcase(path) == devnull:
702 703                 return '\\\\.\\NUL'
703 704
703 705             had_prefix = path.startswith(prefix)
706 +
707 +             if strict is ALLOW_MISSING:
708 +                 ignored_error = FileNotFoundError
709 +                 strict = True
710 +             elif strict:
711 +                 ignored_error = ()
712 +             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)
469 493         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
```



Comments 0