

python / cpython Public

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

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

# Commit dd8f187



6 people authored on Jun 3, 2025 · ❌ 33 / 36 · Partially verified



[3.9] gh-135034: Normalize link targets in tarfile, add os.path.realpath(strict='allow\_missing') (GH-135037) (GH-135084)

Addresses CVEs 2024-12718, 2025-4138, 2025-4330, and 2025-4517. (cherry picked from commit 3612d8f)

Co-authored-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>

· 📦 v3.9.25 ... 3.9

1 parent 24eaf53 commit dd8f187 📄

📁 11 files changed +946 -134 lines changed

↑ Top ⚙

🔍 Filter files...

- ✓ 📁 Doc
- ✓ 📁 library
  - 📄 os.path.rst
  - 📄 tarfile.rst
- ✓ 📁 whatsnew
  - 📄 3.9.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

📄 **11 files changed** +946 -134 lines changed

🔍 Search within code



▼ Doc/library/os.path.rst <> 📄 ⋮

```

... @@ -351,10 +351,26 @@ the :mod:`glob` module.)
351 351     links encountered in the path (if they are supported by the operating
352 352     system).
353 353
354 -   If a path doesn't exist or a symlink loop is encountered, and strict is
355 -   ``True``, :exc:OSError is raised. If strict is ``False``, the path is
356 -   resolved as far as possible and any remainder is appended without checking
357 -   whether it exists.
354 +   By default, the path is evaluated up to the first component that does not
355 +   exist, is a symlink loop, or whose evaluation raises :exc:OSError.
356 +   All such components are appended unchanged to the existing part of the path.
357 +
358 +   Some errors that are handled this way include "access denied", "not a
359 +   directory", or "bad argument to internal function". Thus, the
360 +   resulting path may be missing or inaccessible, may still contain
361 +   links or loops, and may traverse non-directories.
362 +
363 +   This behavior can be modified by keyword arguments:
364 +
365 +   If strict is ``True``, the first error encountered when evaluating the
    path is
366 +   re-raised.

```

```

367 + In particular, :exc:`FileNotFoundError` is raised if *path* does not exist,
368 + or another :exc:`OSError` if it is otherwise inaccessible.
369 +
370 + If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
371 + :exc:`FileNotFoundError` are re-raised (as with `strict=True`).
372 + Thus, the returned path will not contain any symbolic links, but the named
373 + file and some of its parent directories may be missing.

```

358 374

359 375 .. `note::`

360 376 This function emulates the operating system's procedure for making a path

@@ -373,6 +389,15 @@ the `:mod:`glob`` module.)373 389 .. `versionchanged:: 3.9.23`374 390 The `*strict*` parameter was added.

375 391

392 + .. `versionchanged:: next`393 + The `:py:data:`~os.path.ALLOW_MISSING`` value for the `*strict*` parameter  
394 + was added.

395 +

396 + .. `data:: ALLOW_MISSING`

397 +

398 + Special value used for the `*strict*` argument in `:func:`realpath``.

399 +

400 + .. `versionadded:: next`

376 401

377 402 .. `function:: relpath(path, start=os.curdir)`

378 403



Doc/library/tarfile.rst



@@ -237,6 +237,15 @@ The `:mod:`tarfile`` module defines the following  
exceptions:

237 237 Raised to refuse extracting a symbolic link pointing outside the  
destination

238 238 directory.

239 239

240 + .. `exception:: LinkFallbackError`

241 +

242 + Raised to refuse emulating a link (hard or symbolic) by extracting another

243 + archive member, when that member would be rejected by the filter location.

244	+	The exception that was raised to reject the replacement member is available
245	+	as <code>:attr:`!BaseException.__context__`.</code>
246	+	
247	+	<code>.. versionadded:: next</code>
248	+	
240	249	
241	250	The following constants are available at the module level:
242	251	
		@@ -954,6 +963,12 @@ reused in custom filters:
954	963	Implements the <code>``'data'``</code> filter.
955	964	In addition to what <code>``tar_filter``</code> does:
956	965	
966	+	- Normalize link targets ( <code>:attr:`TarInfo.linkname`</code> ) using
967	+	<code>:func:`os.path.normpath`.</code>
968	+	Note that this removes internal <code>``..``</code> components, which may change the
969	+	meaning of the link if the path in <code>:attr:`!TarInfo.linkname`</code> traverses
970	+	symbolic links.
971	+	
957	972	- <code>:ref:`Refuse &lt;tarfile-extraction-refuse&gt;`</code> to extract links (hard or soft)
958	973	that link to absolute paths, or ones that link outside the destination.
959	974	
		@@ -982,6 +997,10 @@ reused in custom filters:
982	997	
983	998	Return the modified <code>``TarInfo``</code> member.
984	999	
1000	+	<code>.. versionchanged:: next</code>
1001	+	
1002	+	Link targets are now normalized.
1003	+	
985	1004	
986	1005	<code>.. _tarfile-extraction-refuse:</code>
987	1006	
		@@ -1008,6 +1027,7 @@ Here is an incomplete list of things to consider:
1008	1027	* Extract to a <code>:func:`new temporary directory &lt;tempfile.mkdtemp&gt;`</code>
1009	1028	to prevent e.g. exploiting pre-existing links, and to make it easier to
1010	1029	clean up after a failed extraction.
1030	+	* Disallow symbolic links if you do not need the functionality.

```

1011 1031 * When working with untrusted data, use external (e.g. OS-level) limits on
1012 1032 disk, memory and CPU usage.
1013 1033 * Check filenames against an allow-list of characters

```



Doc/whatsnew/3.9.rst



@@ -1662,3 +1662,37 @@ email

```

1662 1662 check if the *strict* paramater is available.
1663 1663 (Contributed by Thomas Dwyer and Victor Stinner for :gh:`102988` to improve
1664 1664 the CVE-2023-27043 fix.)
1665 +
1666 +
1667 + Notable changes in 3.9.23
1668 + =====
1669 +
1670 + os.path
1671 + -----
1672 +
1673 + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
1674 + :data:`os.path.ALLOW_MISSING`.
1675 + If used, errors other than :exc:`FileNotFoundError` will be re-raised;
1676 + the resulting path can be missing but it will be free of symlinks.
1677 + (Contributed by Petr Viktorin for CVE 2025-4517.)
1678 +
1679 + tarfile
1680 + -----
1681 +
1682 + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
1683 + to
1684 + avoid path traversal attacks.
1685 + (Contributed by Petr Viktorin in :gh:`127987` and CVE 2025-4138.)
1686 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
1687 + attributes
1688 + when a directory was removed or replaced by another kind of file.
1689 + (Contributed by Petr Viktorin in :gh:`127987` and CVE 2024-12718.)
1690 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
1691 + now (re-)apply the extraction filter when substituting a link (hard or
1692 + symbolic) with a copy of another archive member, and when fixing up
1693 + directory attributes.
1694 + The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.

```

```

1693 + (Contributed by Petr Viktorin for CVE 2025-4330 and CVE 2024-12718.)
1694 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
1695 + no longer extract rejected members when
1696 + :func:`~tarfile.TarFile.errorlevel` is zero.
1697 + (Contributed by Matt Prodan and Petr Viktorin in :gh:`112887`
1698 + 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', 'samefile', 'sameopenfile',
11 -             'samestat']
11 +             'samestat', 'ALLOW_MISSING']

```

```

12 12
13 13
14 14     # Does a path exist?

```

↓

↑

@@ -153,3 +153,12 @@ def \_check\_arg\_types(funcname, \*args):

```

153 153             f'os.PathLike object, not
           {s.__class__.__name__!r}') from None
154 154         if hasstr and hasbytes:
155 155             raise TypeError("Can't mix strings and bytes in path components") from
           None

```

```

156 +
157 + # A singleton with a true boolean value.
158 + @object.__new__
159 + class ALLOW_MISSING:
160 +     """Special value for use in realpath()."""
161 +     def __repr__(self):
162 +         return 'os.path.ALLOW_MISSING'
163 +     def __reduce__(self):
164 +         return self.__class__.__name__

```

Lib/ntpath.py

↑

@@ -29,7 +29,8 @@

```

29 29         "ismount", "expanduser", "expandvars", "normpath", "abspath",
30 30         "curdir", "pardir", "sep", "pathsep", "defpath", "altsep",

```

```

31 31
    "extsep", "devnull", "realpath", "supports_unicode_filenames", "relpath",
32 -     "samefile", "sameopenfile", "samestat", "commonpath"]
32 +     "samefile", "sameopenfile", "samestat", "commonpath",
33 +     "ALLOW_MISSING"]
33 34
34 35     def _get_bothseps(path):
35 36         if isinstance(path, bytes):
@@ -532,9 +533,10 @@ def abspath(path):
532 533         from nt import _getfinalpathname, readlink as _nt_readlink
533 534     except ImportError:
534 535         # realpath is a no-op on systems without _getfinalpathname support.
535 -     realpath = abspath
536 +     def realpath(path, *, strict=False):
537 +         return abspath(path)
536 538     else:
537 -     def _readlink_deep(path):
539 +     def _readlink_deep(path, ignored_error=OSError):
538 540         # These error codes indicate that we should stop reading links and
539 541         # return the path we currently have.
540 542         # 1: ERROR_INVALID_FUNCTION
@@ -567,7 +569,7 @@ def _readlink_deep(path):
567 569             path = old_path
568 570             break
569 571             path = normpath(join(dirname(old_path), path))
570 -     except OSError as ex:
572 +     except ignored_error as ex:
571 573         if ex.winerror in allowed_winerror:
572 574             break
573 575             raise
@@ -576,7 +578,7 @@ def _readlink_deep(path):
576 578             break
577 579             return path
578 580
579 -     def _getfinalpathname_nonstrict(path):
581 +     def _getfinalpathname_nonstrict(path, ignored_error=OSError):
580 582         # These error codes indicate that we should stop resolving the path
581 583         # and return the value we currently have.

```

```

582 584 # 1: ERROR_INVALID_FUNCTION
@@ -600,17 +602,18 @@ def _getfinalpathname_nonstrict(path):
    try:
        path = _getfinalpathname(path)
        return join(path, tail) if tail else path
    - except OSError as ex:
    + except ignored_error as ex:
        if ex.winerror not in allowed_winerror:
            raise
        try:
            # The OS could not resolve this path fully, so we attempt
            # to follow the link ourselves. If we succeed, join the
            tail
            # and return.
    - new_path = _readlink_deep(path)
    + new_path = _readlink_deep(path,
    + ignored_error=ignored_error)
        if new_path != path:
            return join(new_path, tail) if tail else new_path
    - except OSError:
    + except ignored_error:
            # If we fail to readlink(), let's keep traversing
            pass
        path, name = split(path)
@@ -641,16 +644,24 @@ def realpath(path, *, strict=False):
    if normcase(path) == normcase(devnull):
        return '\\\\.\\NUL'
    had_prefix = path.startswith(prefix)
    +
    + if strict is ALLOW_MISSING:
    +     ignored_error = FileNotFoundError
    +     strict = True
    + elif strict:
    +     ignored_error = ()
    + else:
    +     ignored_error = OSError
    +
    if not had_prefix and not isabs(path):
        path = join(cwd, path)

```

```

646 658         try:
647 659             path = _getfinalpathname(path)
648 660             initial_winerror = 0
649 -         except OSError as ex:
650 -             if strict:
651 -                 raise
661 +         except ignored_error as ex:
652 662             initial_winerror = ex.winerror
653 -         path = _getfinalpathname_nonstrict(path)
663 +         path = _getfinalpathname_nonstrict(path,
664 +             ignored_error=ignored_error)
654 665     # The path returned by _getfinalpathname will always start with \\?\ -
655 666     # strip off that prefix unless it was already provided on the original
656 667     # 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"]
38 +         "commonpath", "ALLOW_MISSING"]

```

```

39 39
40 40
41 41     def _get_sep(path):

```



@@ -403,6 +403,15 @@ def \_joinrealpath(path, rest, strict, seen):

```

403 403         sep = '/'
404 404         curdir = '.'
405 405         pardir = '..'
406 +         getcwd = os.getcwd
407 +         if strict is ALLOW_MISSING:
408 +             ignored_error = FileNotFoundError
409 +         elif strict:
410 +             ignored_error = ()
411 +         else:
412 +             ignored_error = OSError
413 +
414 +         maxlinks = None

```

```
406 415
407 416     if isabs(rest):
408 417         rest = rest[1:]
@@ -425,9 +434,7 @@ def _joinrealpath(path, rest, strict, seen):
425 434         newpath = join(path, name)
426 435         try:
427 436             st = os.lstat(newpath)
428 -         except OSError:
429 -             if strict:
430 -                 raise
437 +         except ignored_error:
431 438             is_link = False
432 439         else:
433 440             is_link = stat.S_ISLNK(st.st_mode)
```

Comments 0