

python / cpython Public

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

Commit 4633f3f



6 people authored on Jun 3, 2025 · ✓ 57 / 58 · Partially verified

```
[3.11] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (GH-135068)

Addresses CVEs 2024-12718, 2025-4138, 2025-4330, and 2025-4517.
(cherry picked from commit 3612d8f)
(cherry picked from commit c358142)

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>
```

🔗 3.11 (#98846, #135068) · v3.11.15 ... v3.11.13

1 parent 2c6ca1a commit 4633f3f

11 files changed +1017 -138 lines changed

↑ Top ⚙️

Filter files...

- ✓ Doc
 - ✓ library
 - os.path.rst
 - tarfile.rst
 - ✓ whatsnew
 - 3.11.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** +1017 -138 lines changed

🔍 Search within code



▼ Doc/library/os.path.rst ⏪ 📄 ⋮

```

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

```

```

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

359 375
360 376     .. note::
361 377         This function emulates the operating system's procedure for making a path
@@ -374,6 +390,15 @@ the :mod:`glob` module.)
374 390     .. versionchanged:: 3.10
375 391         The *strict* parameter was added.
376 392

393 +     .. versionchanged:: next
394 +         The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
395 +         was added.
396 +
397 + .. data:: ALLOW_MISSING
398 +
399 +     Special value used for the *strict* argument in :func:`realpath`.
400 +
401 +     .. versionadded:: next

377 402
378 403     .. function:: relpath(path, start=os.curdir)
379 404

```

Doc/library/tarfile.rst

```

@@ -239,6 +239,15 @@ The :mod:`tarfile` module defines the following
exceptions:

239 239     Raised to refuse extracting a symbolic link pointing outside the
destination
240 240     directory.
241 241

242 + .. exception:: LinkFallbackError
243 +
244 +     Raised to refuse emulating a link (hard or symbolic) by extracting another
245 +     archive member, when that member would be rejected by the filter location.
246 +     The exception that was raised to reject the replacement member is
available
247 +     as :attr:`!BaseException.__context__`.
248 +

```

249	+	<code>.. versionadded:: next</code>
250	+	
242	251	
243	252	The following constants are available at the module level:
244	253	
⋮ ↓ ↑ ⋮		@@ -1037,6 +1046,12 @@ reused in custom filters:
1037	1046	Implements the <code>``'data'``</code> filter.
1038	1047	In addition to what <code>``tar_filter``</code> does:
1039	1048	
1049	+	- Normalize link targets (<code>:attr:`TarInfo.linkname`</code>) using
1050	+	<code>:func:`os.path.normpath`</code> .
1051	+	Note that this removes internal <code>``..``</code> components, which may change the
1052	+	meaning of the link if the path in <code>:attr:`!TarInfo.linkname`</code> traverses
1053	+	symbolic links.
1054	+	
1040	1055	- <code>:ref:`Refuse <tarfile-extraction-refuse></code> to extract links (hard or soft)
1041	1056	that link to absolute paths, or ones that link outside the destination.
1042	1057	
⋮ ↓ ↑ ⋮		@@ -1065,6 +1080,10 @@ reused in custom filters:
1065	1080	
1066	1081	Return the modified <code>``TarInfo``</code> member.
1067	1082	
1083	+	<code>.. versionchanged:: next</code>
1084	+	
1085	+	Link targets are now normalized.
1086	+	
1068	1087	
1069	1088	<code>.. _tarfile-extraction-refuse:</code>
1070	1089	
⋮ ↕ ⋮		@@ -1091,6 +1110,7 @@ Here is an incomplete list of things to consider:
1091	1110	* Extract to a <code>:func:`new temporary directory <tempfile.mkdtemp></code>
1092	1111	to prevent e.g. exploiting pre-existing links, and to make it easier to
1093	1112	clean up after a failed extraction.
1113	+	* Disallow symbolic links if you do not need the functionality.
1094	1114	* When working with untrusted data, use external (e.g. OS-level) limits on
1095	1115	disk, memory and CPU usage.
1096	1116	* Check filenames against an allow-list of characters
⋮ ↓ ⋮		

Doc/whatsnew/3.11.rst



```

↑... @@ -2786,3 +2786,37 @@ email
2786 2786     check if the *strict* parameter is available.
2787 2787     (Contributed by Thomas Dwyer and Victor Stinner for :gh:`102988` to improve
2788 2788     the CVE-2023-27043 fix.)

2789 +
2790 +
2791 + Notable changes in 3.11.13
2792 + =====
2793 +
2794 + os.path
2795 + -----
2796 +
2797 + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
2798 + :data:`os.path.ALLOW_MISSING`.
2799 + If used, errors other than :exc:`FileNotFoundError` will be re-raised;
2800 + the resulting path can be missing but it will be free of symlinks.
2801 + (Contributed by Petr Viktorin for CVE 2025-4517.)
2802 +
2803 + tarfile
2804 + -----
2805 +
2806 + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
2807 + to
2808 + avoid path traversal attacks.
2809 + (Contributed by Petr Viktorin in :gh:`127987` and CVE 2025-4138.)
2810 +
2811 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
2812 + attributes
2813 + when a directory was removed or replaced by another kind of file.
2814 + (Contributed by Petr Viktorin in :gh:`127987` and CVE 2024-12718.)
2815 +
2816 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2817 + now (re-)apply the extraction filter when substituting a link (hard or
2818 + symbolic) with a copy of another archive member, and when fixing up
2819 + directory attributes.
2820 + The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
2821 + (Contributed by Petr Viktorin for CVE 2025-4330 and CVE 2024-12718.)
2822 +
2823 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2824 + no longer extract rejected members when
2825 + :func:`~tarfile.TarFile.errorlevel` is zero.

```

2821 + (Contributed by Matt Prodan and Petr Viktorin in :gh:`112887`
 2822 + 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



@@ -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     -         "samefile", "sameopenfile", "samestat", "commonpath"]
33     +         "samefile", "sameopenfile", "samestat", "commonpath",
34     +         "ALLOW_MISSING"]

```

```

34 35
35 36     def _get_bothseps(path):
36 37         if isinstance(path, bytes):
37 38             return path, path
38 39
39 40     def abspath(path):
40 41         """Return an absolute path.
41 42         If path is a relative path, it is converted to an absolute path
42 43         by prepending the current directory. If path is already an absolute
43 44         path, it is returned unchanged.
44 45         """
45 46         from nt import _getfinalpathname, readlink as _nt_readlink
46 47     except ImportError:
47 48         # realpath is a no-op on systems without _getfinalpathname support.
48 49         realpath = abspath
49 50     def realpath(path, *, strict=False):
50 51         """Return the real path for path.
51 52         If path is a symbolic link, the real path is returned. If path is
52 53         not a symbolic link, path is returned unchanged.
53 54         """
54 55         if not strict:
55 56             return abspath(path)
56 57         else:
57 58             def _readlink_deep(path):
58 59                 """Read the real path for path.
59 60                 # These error codes indicate that we should stop reading links and
60 61                 # return the path we currently have.
61 62                 # 1: ERROR_INVALID_FUNCTION
62 63                 """
63 64                 path = old_path
64 65                 break
65 66                 path = normpath(join(dirname(old_path), path))
66 67             except OSError as ex:
67 68                 except ignored_error as ex:
68 69                     if ex.winerror in allowed_winerror:
69 70                         break
70 71                     raise
71 72             def _readlink_deep(path):
72 73                 """Read the real path for path.
73 74                 # These error codes indicate that we should stop resolving the path
74 75                 # and return the value we currently have.
75 76                 # 1: ERROR_INVALID_FUNCTION
76 77                 """
77 78                 try:
78 79                     path = _getfinalpathname(path)
79 80                 except OSError as ex:
80 81                     if ex.winerror in allowed_winerror:
81 82                         break
82 83                     raise
83 84                 return join(path, tail) if tail else path

```

```

652 -         except OSError as ex:
654 +         except ignored_error as ex:
653 655             if ex.winerror not in allowed_winerror:
654 656                 raise
655 657             try:
656 658                 # The OS could not resolve this path fully, so we attempt
657 659                 # to follow the link ourselves. If we succeed, join the
658 660                 tail
658 660                 # and return.
659 -         new_path = _readlink_deep(path)
661 +         new_path = _readlink_deep(path,
662 +                                 ignored_error=ignored_error)
660 663             if new_path != path:
661 664                 return join(new_path, tail) if tail else new_path
662 -         except OSError:
665 +         except ignored_error:
663 666             # If we fail to readlink(), let's keep traversing
664 667             pass
665 668             path, name = split(path)
666 -         @@ -690,24 +693,32 @@ def realpath(path, *, strict=False):
690 693             if normcase(path) == normcase(devnull):
691 694                 return '\\\\.\NUL'
692 695             had_prefix = path.startswith(prefix)
696 +
697 +         if strict is ALLOW_MISSING:
698 +             ignored_error = FileNotFoundError
699 +             strict = True
700 +         elif strict:
701 +             ignored_error = ()
702 +         else:
703 +             ignored_error = OSError
704 +
693 705             if not had_prefix and not isabs(path):
694 706                 path = join(cwd, path)
695 707             try:
696 708                 path = _getfinalpathname(path)
697 709                 initial_winerror = 0
698 710             except ValueError as ex:
699 711                 # gh-106242: Raised for embedded null characters

```

```

700 - # In strict mode, we convert into an OSError.
712 + # In strict modes, we convert into an OSError.
701 713 # Non-strict mode returns the path as-is, since we've already
702 714 # made it absolute.
703 715 if strict:
704 716     raise OSError(str(ex)) from None
705 717     path = normpath(path)
706 - except OSError as ex:
707 -     if strict:
708 -         raise
718 + except ignored_error as ex:
709 719     initial_winerror = ex.winerror
710 - path = _getfinalpathname_nonstrict(path)
720 + path = _getfinalpathname_nonstrict(path,
721 +     ignored_error=ignored_error)
711 722 # The path returned by _getfinalpathname will always start with \\?\ -
712 723 # strip off that prefix unless it was already provided on the original
713 724 # 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):
@@ -427,6 +427,15 @@ def _joinrealpath(path, rest, strict, seen):
427 427     sep = '/'
428 428     curdir = '.'
429 429     pardir = '..'
430 +     getcwd = os.getcwd
431 +     if strict is ALLOW_MISSING:
432 +         ignored_error = FileNotFoundError
433 +     elif strict:
434 +         ignored_error = ()

```

```
435 +     else:
436 +         ignored_error = OSError
437 +
438 +     maxlinks = None
439
440     if isabs(rest):
441         rest = rest[1:]
442
443     @@ -449,9 +458,7 @@ def _joinrealpath(path, rest, strict, seen):
444
445         newpath = join(path, name)
446
447         try:
448             st = os.lstat(newpath)
449
450         except OSError:
451             if strict:
452                 raise
453
454         +     except ignored_error:
455
456             is_link = False
457
458         else:
459             is_link = stat.S_ISLNK(st.st_mode)
```

Comments 0