

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

Top

🔍 Filter files...

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

- 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



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

1683 + avoid path traversal attacks.

1684 + (Contributed by Petr Viktorin in `:gh:`127987`` and CVE 2025-4138.)

1685 + \* `:func:`~tarfile.TarFile.extractall`` now skips fixing up directory  
attributes

1686 + when a directory was removed or replaced by another kind of file.

1687 + (Contributed by Petr Viktorin in `:gh:`127987`` and CVE 2024-12718.)

1688 + \* `:func:`~tarfile.TarFile.extract`` and `:func:`~tarfile.TarFile.extractall``

1689 + now (re-)apply the extraction filter when substituting a link (hard or

1690 + symbolic) with a copy of another archive member, and when fixing up

1691 + directory attributes.

1692 + 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
+         except ignored_error:
431 438             is_link = False
432 439         else:
433 440             is_link = stat.S_ISLNK(st.st_mode)

```

Lib/tarfile.py



```

@@ -749,10 +749,22 @@ def __init__(self, tarinfo, path):
749 749         super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
750 750             + 'which is outside the destination')
751 751
+ class LinkFallbackError(FilterError):
752 +     def __init__(self, tarinfo, path):
753 +         self.tarinfo = tarinfo
754 +         self._path = path
755 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
756 +             + f'copy of {path!r}, which was rejected')
757 +
758 +
759 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
760 + # we consider to be issues with the argument, rather than a bug in the
761 + # filter function
762 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
763 +
752 764     def _get_filtered_attrs(member, dest_path, for_data=True):
753 765         new_attrs = {}
754 766         name = member.name
755 -         dest_path = os.path.realpath(dest_path)
+         dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
756 768         # Strip leading / (tar's directory separator) from filenames.
757 769         # Include os.sep (target OS directory separator) as well.

```

```

758     770         if name.startswith('/'):
@@ -762,7 +774,8 @@ def _get_filtered_attrs(member, dest_path,
    for_data=True):
762     774             # For example, 'C:/foo' on Windows.
763     775             raise AbsolutePathError(member)
764     776             # Ensure we stay in the destination
765     - target_path = os.path.realpath(os.path.join(dest_path, name))
    777     + target_path = os.path.realpath(os.path.join(dest_path, name),
    778     +                               strict=os.path.ALLOW_MISSING)
766     779         if os.path.commonpath([target_path, dest_path]) != dest_path:
767     780             raise OutsideDestinationError(member, target_path)
768     781             # Limit permissions (no high bits, and go-w)
@@ -800,14 +813,18 @@ def _get_filtered_attrs(member, dest_path,
    for_data=True):
800     813         if member.islnk() or member.issym():
801     814             if os.path.isabs(member.linkname):
802     815                 raise AbsoluteLinkError(member)
    816     + normalized = os.path.normpath(member.linkname)
    817     + if normalized != member.linkname:
    818     +     new_attrs['linkname'] = normalized
803     819         if member.issym():
804     820             target_path = os.path.join(dest_path,
805     821                                         os.path.dirname(name),
806     822                                         member.linkname)
807     823         else:
808     824             target_path = os.path.join(dest_path,
809     825                                         member.linkname)
810     - target_path = os.path.realpath(target_path)
    826     + target_path = os.path.realpath(target_path,
    827     +                               strict=os.path.ALLOW_MISSING)
811     828         if os.path.commonpath([target_path, dest_path]) != dest_path:
812     829             raise LinkOutsideDestinationError(member, target_path)
813     830         return new_attrs
@@ -2268,30 +2285,58 @@ def extractall(self, path=".", members=None, *,
    numeric_owner=False,
2268     2285             members = self
2269     2286
2270     2287             for member in members:
2271     - tarinfo = self._get_extract_tarinfo(member, filter_function,
    path)

```

```

2288 +         tarinfo, unfiltered = self._get_extract_tarinfo(
2289 +             member, filter_function, path)
2272 2290         if tarinfo is None:
2273 2291             continue
2274 2292         if tarinfo.isdir():
2275 2293             # For directories, delay setting attributes until later,
2276 2294             # since permissions can interfere with extraction and
2277 2295             # extracting contents can reset mtime.
2278 -         directories.append(tarinfo)
2296 +         directories.append(unfiltered)
2279 2297         self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
2280 -             numeric_owner=numeric_owner)
2298 +             numeric_owner=numeric_owner,
2299 +             filter_function=filter_function)
2281 2300
2282 2301         # Reverse sort directories.
2283 2302         directories.sort(key=lambda a: a.name, reverse=True)
2284 2303
2304 +
2285 2305         # Set correct owner, mtime and filemode on directories.
2286 -         for tarinfo in directories:
2287 -             dirpath = os.path.join(path, tarinfo.name)
2306 +         for unfiltered in directories:
2288 2307             try:
2308 +                 # Need to re-apply any filter, to take the *current*
filesystem
2309 +                 # state into account.
2310 +                 try:
2311 +                     tarinfo = filter_function(unfiltered, path)
2312 +                 except _FILTER_ERRORS as exc:
2313 +                     self._log_no_directory_fixup(unfiltered, repr(exc))
2314 +                     continue
2315 +                 if tarinfo is None:
2316 +                     self._log_no_directory_fixup(unfiltered,
2317 +                                                     'excluded by filter')
2318 +                     continue
2319 +                     dirpath = os.path.join(path, tarinfo.name)
2320 +                 try:
2321 +                     lstat = os.lstat(dirpath)
2322 +                 except FileNotFoundError:

```

```

2323 +         self._log_no_directory_fixup(tarinfo, 'missing')
2324 +         continue
2325 +         if not stat.S_ISDIR(lstat.st_mode):
2326 +             # This is no longer a directory; presumably a later
2327 +             # member overwrote the entry.
2328 +             self._log_no_directory_fixup(tarinfo, 'not a directory')
2329 +             continue
2289 2330         self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
2290 2331         self.utime(tarinfo, dirpath)
2291 2332         self.chmod(tarinfo, dirpath)
2292 2333         except ExtractError as e:
2293 2334             self._handle_nonfatal_error(e)
2294 2335
2336 +     def _log_no_directory_fixup(self, member, reason):
2337 +         self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
2338 +             (member.name, reason))
2339 +
2295 2340     def extract(self, member, path="", set_attrs=True, *,
2296 2341         numeric_owner=False,
2297 2342         filter=None):
2298 2343         """Extract a member from the archive to the current working
2299 2344         directory,
2300 2345
2301 2346         @@ -2307,41 +2352,56 @@ def extract(self, member, path="",
2302 2347         set_attrs=True, *, numeric_owner=False,
2303 2348
2304 2349         String names of common filters are accepted.
2305 2350         """
2306 2351         filter_function = self._get_filter_function(filter)
2307 2352         tarinfo = self._get_extract_tarinfo(member, filter_function, path)
2308 2353
2309 2354         tarinfo, unfiltered = self._get_extract_tarinfo(
2310 2355             member, filter_function, path)
2311 2356
2312 2357         if tarinfo is not None:
2313 2358             self._extract_one(tarinfo, path, set_attrs, numeric_owner)
2314 2359
2315 2360     def _get_extract_tarinfo(self, member, filter_function, path):
2316 2361         """Get filtered TarInfo (or None) from member, which might be a
2317 2362         str"""
2318 2363
2319 2364         """Get (filtered, unfiltered) TarInfos from *member*
2320 2365
2321 2366         *member* might be a string.

```

```

2365 +         Return (None, None) if not found.
2366 +         """
2367 +
2316 2368         if isinstance(member, str):
2317 -             tarinfo = self.getmember(member)
2369 +             unfiltered = self.getmember(member)
2318 2370         else:
2319 -             tarinfo = member
2371 +             unfiltered = member
2320 2372
2321 -             unfiltered = tarinfo
2373 +             filtered = None
2322 2374         try:
2323 -             tarinfo = filter_function(tarinfo, path)
2375 +             filtered = filter_function(unfiltered, path)
2324 2376         except (OSError, FilterError) as e:
2325 2377             self._handle_fatal_error(e)
2326 2378         except ExtractError as e:
2327 2379             self._handle_nonfatal_error(e)
2328 -             if tarinfo is None:
2380 +             if filtered is None:
2329 2381                 self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2330 -             return None
2382 +             return None, None
2383 +
2331 2384         # Prepare the link target for makelink().
2332 -             if tarinfo.islnk():
2333 -                 tarinfo = copy.copy(tarinfo)
2334 -                 tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2335 -             return tarinfo
2385 +             if filtered.islnk():
2386 +                 filtered = copy.copy(filtered)
2387 +                 filtered._link_target = os.path.join(path, filtered.linkname)
2388 +             return filtered, unfiltered
2336 2389
2337 -         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2338 -             """Extract from filtered tarinfo to disk"""
2390 +         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
2391 +                         filter_function=None):
2392 +             """Extract from filtered tarinfo to disk.

```

2393	+	
2394	+	filter_function is only used when extracting a *different*
2395	+	member (e.g. as fallback to creating a symlink)
2396	+	"""
2339	2397	self._check("r")
2340	2398	
2341	2399	try:
2342	2400	self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2343	2401	set_attrs=set_attrs,
2344	-	numeric_owner=numeric_owner)
2402	+	numeric_owner=numeric_owner,
2403	+	filter_function=filter_function,
2404	+	extraction_root=path)
2345	2405	except OSError as e:
2346	2406	self._handle_fatal_error(e)
2347	2407	except ExtractError as e:
⋮ ↓ ↑ ⋮		@@ -2399,9 +2459,13 @@ def extractfile(self, member):
2399	2459	return None
2400	2460	
2401	2461	def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2402	-	numeric_owner=False):
2403	-	"""Extract the TarInfo object tarinfo to a physical
2462	+	numeric_owner=False, *, filter_function=None,
2463	+	extraction_root=None):
2464	+	"""Extract the filtered TarInfo object tarinfo to a physical
2404	2465	file called targetpath.
2466	+	
2467	+	filter_function is only used when extracting a *different*
2468	+	member (e.g. as fallback to creating a symlink)
2405	2469	"""
2406	2470	# Fetch the TarInfo object for the given name
2407	2471	# and build the destination pathname, replacing
⋮ ↓ ↑ ⋮		@@ -2430,7 +2494,10 @@ def _extract_member(self, tarinfo, targetpath,
2430	2494	elif tarinfo.ischr() or tarinfo.isblk():
2431	2495	self.makedev(tarinfo, targetpath)
2432	2496	elif tarinfo.islnk() or tarinfo.issym():
2433	-	self.makelink(tarinfo, targetpath)
2497	+	self.makelink_with_filter(

```

2498 +         tarinfo, targetpath,
2499 +         filter_function=filter_function,
2500 +         extraction_root=extraction_root)
2434 2501         elif tarinfo.type not in SUPPORTED_TYPES:
2435 2502             self.makeunknown(tarinfo, targetpath)
2436 2503         else:
2512 2579         os.makedev(tarinfo.devmajor, tarinfo.devminor))
2513 2580
2514 2581     def makelink(self, tarinfo, targetpath):
2582 +         return self.makelink_with_filter(tarinfo, targetpath, None, None)
2583 +
2584 +     def makelink_with_filter(self, tarinfo, targetpath,
2585 +                             filter_function, extraction_root):
2515 2586         """Make a (symbolic) link called targetpath. If it cannot be created
2516 2587         (platform limitation), we try to make a copy of the referenced file
2517 2588         instead of a link.
2589 +
2590 +         filter_function is only used when extracting a *different*
2591 +         member (e.g. as fallback to creating a link).
2518 2592         """
2593 +         keyerror_to_extracterror = False
2519 2594         try:
2520 2595             # For systems that support symbolic and hard links.
2521 2596             if tarinfo.issym():
2522 2597                 if os.path.lexists(targetpath):
2523 2598                     # Avoid FileExistsError on following os.symlink.
2524 2599                     os.unlink(targetpath)
2525 2600                     os.symlink(tarinfo.linkname, targetpath)
2601 +                 return
2526 2602         else:
2527 2603             if os.path.exists(tarinfo._link_target):
2528 2604                 os.link(tarinfo._link_target, targetpath)
2529 -             else:
2530 -                 self._extract_member(self._find_link_target(tarinfo),
2531 -                                     targetpath)
2605 +                 return
2532 2606         except symlink_exception:
2607 +             keyerror_to_extracterror = True

```

```

2608 +
2609 +     try:
2610 +         unfiltered = self._find_link_target(tarinfo)
2611 +     except KeyError:
2612 +         if keyerror_to_extracterror:
2613 +             raise ExtractError(
2614 +                 "unable to resolve link inside archive")
2615 +         else:
2616 +             raise
2617 +
2618 +     if filter_function is None:
2619 +         filtered = unfiltered
2620 +     else:
2621 +         if extraction_root is None:
2622 +             raise ExtractError(
2623 +                 "makelink_with_filter: if filter_function is not None, "
2624 +                 + "extraction_root must also not be None")

```

```

2533 2625         try:
2534 -             self._extract_member(self._find_link_target(tarinfo),
2535 -                                 targetpath)
2536 -         except KeyError:
2537 -             raise ExtractError("unable to resolve link inside archive")

```

```

2626 +         filtered = filter_function(unfiltered, extraction_root)
2627 +     except _FILTER_ERRORS as cause:
2628 +         raise LinkFallbackError(tarinfo, unfiltered.name) from cause
2629 +     if filtered is not None:
2630 +         self._extract_member(filtered, targetpath,
2631 +                               filter_function=filter_function,
2632 +                               extraction_root=extraction_root)

```

```

2538 2633
2539 2634     def chown(self, tarinfo, targetpath, numeric_owner):
2540 2635         """Set owner of targetpath according to tarinfo. If numeric_owner

```



Lib/test/test\_ntpath.py



```

... @@ -1,8 +1,10 @@
1 1     import ntpath
2 2     import os
3 + import subprocess
3 4     import sys

```

```
4 5 import unittest
```

```
5 6 import warnings
```

```
7 + from ntpath import ALLOW_MISSING
```

```
6 8 from test.support import TestFailed, FakePath
```

```
7 9 from test import support, test_genericpath
```

```
8 10 from tempfile import TemporaryFile
```



```
@@ -72,6 +74,27 @@ def tester(fn, wantResult):
```

```
72 74         %(str(fn), str(wantResult), repr(gotResult)))
```

```
73 75
```

```
74 76
```

```
77 + def _parameterize(*parameters):
```

```
78 +     """Simplistic decorator to parametrize a test
```

```
79 +
```

```
80 +     Runs the decorated test multiple times in subTest, with a value from
```

```
81 +     'parameters' passed as an extra positional argument.
```

```
82 +     Calls doCleanups() after each run.
```

```
83 +
```

```
84 +     Not for general use. Intended to avoid indenting for easier backports.
```

```
85 +
```

```
86 +     See https://discuss.python.org/t/91827 for discussing generalizations.
```

```
87 +     """
```

```
88 +     def _parametrize_decorator(func):
```

```
89 +         def _parameterized(self, *args, **kwargs):
```

```
90 +             for parameter in parameters:
```

```
91 +                 with self.subTest(parameter):
```

```
92 +                     func(self, *args, parameter, **kwargs)
```

```
93 +                     self.doCleanups()
```

```
94 +             return _parameterized
```

```
95 +         return _parametrize_decorator
```

```
96 +
```

```
97 +
```

```
75 98 class NtpathTestCase(unittest.TestCase):
```

```
76 99     def assertPathEqual(self, path1, path2):
```

```
77 100         if path1 == path2 or _norm(path1) == _norm(path2):
```



```
@@ -242,6 +265,27 @@ def test_realpath_curdir(self):
```

```
242 265         tester("ntpath.realpath('.\\.\\.')", expected)
```

```
243 266         tester("ntpath.realpath('\\.\\.\\.join(['.'] * 100))", expected)
```

```
244 267
```

```

268 +     def test_realpath_curdir_strict(self):
269 +         expected = ntpath.normpath(os.getcwd())
270 +         tester("ntpath.realpath('.', strict=True)", expected)
271 +         tester("ntpath.realpath('./.', strict=True)", expected)
272 +         tester("ntpath.realpath('/'.join(['.'] * 100), strict=True)", expected)
273 +         tester("ntpath.realpath('..\.', strict=True)", expected)
274 +         tester("ntpath.realpath('\\".join(['.'] * 100), strict=True)",
expected)
275 +
276 +     def test_realpath_curdir_missing_ok(self):
277 +         expected = ntpath.normpath(os.getcwd())
278 +         tester("ntpath.realpath('.', strict=ALLOW_MISSING)",
expected)
279 +         tester("ntpath.realpath('./.', strict=ALLOW_MISSING)",
expected)
280 +         tester("ntpath.realpath('/'.join(['.'] * 100), strict=ALLOW_MISSING)",
expected)
281 +         tester("ntpath.realpath('..\.', strict=ALLOW_MISSING)",
expected)
282 +         tester("ntpath.realpath('\\".join(['.'] * 100), strict=ALLOW_MISSING)",
expected)
283 +         tester("ntpath.realpath('..\.', strict=ALLOW_MISSING)",
expected)
284 +         tester("ntpath.realpath('\\".join(['.'] * 100), strict=ALLOW_MISSING)",
expected)
285 +         tester("ntpath.realpath('..\.', strict=ALLOW_MISSING)",
expected)
286 +         tester("ntpath.realpath('\\".join(['.'] * 100), strict=ALLOW_MISSING)",
expected)
287 +         tester("ntpath.realpath('..\.', strict=ALLOW_MISSING)",
expected)
288 +
245 289     def test_realpath_pardir(self):
246 290         expected = ntpath.normpath(os.getcwd())
247 291         tester("ntpath.realpath('..')", ntpath.dirname(expected))
@@ -254,17 +298,43 @@ def test_realpath_pardir(self):
254 298         tester("ntpath.realpath('\\".join(['..'] * 50))",
ntpath.splitdrive(expected)[0] + '\\')
255 299
256 300
301 +     def test_realpath_pardir_strict(self):
302 +         expected = ntpath.normpath(os.getcwd())
303 +         tester("ntpath.realpath('..', strict=True)", ntpath.dirname(expected))
304 +         tester("ntpath.realpath('..../.', strict=True)",
ntpath.dirname(ntpath.dirname(expected)))
305 +         tester("ntpath.realpath('/'.join(['..'] * 50), strict=True)",
ntpath.splitdrive(expected)[0] + '\\')
306 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.dirname(ntpath.dirname(expected)))
307 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.splitdrive(expected)[0] + '\\')
308 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.dirname(ntpath.dirname(expected)))
309 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.splitdrive(expected)[0] + '\\')
310 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.splitdrive(expected)[0] + '\\')
311 +         tester("ntpath.realpath('..\\".join(['..'] * 50), strict=True)",
ntpath.splitdrive(expected)[0] + '\\')

```

```

312 +
313 +     def test_realpath_pardir_missing_ok(self):
314 +         expected = ntpath.normpath(os.getcwd())
315 +         tester("ntpath.realpath '..', strict=ALLOW_MISSING)",
316 +               ntpath.dirname(expected))
317 +         tester("ntpath.realpath('../..', strict=ALLOW_MISSING)",
318 +               ntpath.dirname(ntpath.dirname(expected)))
319 +         tester("ntpath.realpath('/'.join(['..'] * 50), strict=ALLOW_MISSING)",
320 +               ntpath.splitdrive(expected)[0] + '\\')
321 +         tester("ntpath.realpath('..\\..', strict=ALLOW_MISSING)",
322 +               ntpath.dirname(ntpath.dirname(expected)))
323 +         tester("ntpath.realpath('\\'.join(['..'] * 50), strict=ALLOW_MISSING)",
324 +               ntpath.splitdrive(expected)[0] + '\\')
325 +
257 326         @support.skip_unless_symlink
258 327         @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
259 -     def test_realpath_basic(self):
328 +         @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
329 +     def test_realpath_basic(self, kwargs):
260 330         ABSTFN = ntpath.abspath(support.TESTFN)
261 331         open(ABSTFN, "wb").close()
262 332         self.addCleanup(support.unlink, ABSTFN)
263 333         self.addCleanup(support.unlink, ABSTFN + "1")
264 334
265 335         os.symlink(ABSTFN, ABSTFN + "1")
266 -         self.assertEqual(ntpath.realpath(ABSTFN + "1"), ABSTFN)
267 -         self.assertEqual(ntpath.realpath(os.fsencode(ABSTFN + "1")),
336 +         self.assertEqual(ntpath.realpath(ABSTFN + "1", **kwargs), ABSTFN)
337 +         self.assertEqual(ntpath.realpath(os.fsencode(ABSTFN + "1"),
**kwargs),
268 338                             os.fsencode(ABSTFN))
269 339
270 340         @support.skip_unless_symlink
↕ @@ -280,14 +350,15 @@ def test_realpath_strict(self):
280 350
281 351         @support.skip_unless_symlink
282 352         @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
283 -     def test_realpath_relative(self):
353 +         @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
354 +     def test_realpath_relative(self, kwargs):

```

```

284 355         ABSTFN = ntpath.abspath(support.TESTFN)
285 356         open(ABSTFN, "wb").close()
286 357         self.addCleanup(support.unlink, ABSTFN)
287 358         self.addCleanup(support.unlink, ABSTFN + "1")
288 359
289 360         os.symlink(ABSTFN, ntpath.relpath(ABSTFN + "1"))
290 -         self.assertEqual(ntpath.realpath(ABSTFN + "1"), ABSTFN)
361 +         self.assertEqual(ntpath.realpath(ABSTFN + "1", **kwargs), ABSTFN)
291 362
292 363         @support.skip_unless_symlink
293 364         @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
    ↓
    ↑
@@ -439,7 +510,62 @@ def test_realpath_symlink_loops_strict(self):
439 510
440 511         @support.skip_unless_symlink
441 512         @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
442 -         def test_realpath_symlink_prefix(self):
513 +         def test_realpath_symlink_loops_raise(self):
514 +             # Symlink loops raise OSError in ALLOW_MISSING mode
515 +             ABSTFN = ntpath.abspath(support.TESTFN)
516 +             self.addCleanup(support.unlink, ABSTFN)
517 +             self.addCleanup(support.unlink, ABSTFN + "1")
518 +             self.addCleanup(support.unlink, ABSTFN + "2")
519 +             self.addCleanup(support.unlink, ABSTFN + "y")
520 +             self.addCleanup(support.unlink, ABSTFN + "c")
521 +             self.addCleanup(support.unlink, ABSTFN + "a")
522 +             self.addCleanup(support.unlink, ABSTFN + "x")
523 +
524 +             os.symlink(ABSTFN, ABSTFN)
525 +             self.assertRaises(OSError, ntpath.realpath, ABSTFN,
strict=ALLOW_MISSING)
526 +
527 +             os.symlink(ABSTFN + "1", ABSTFN + "2")
528 +             os.symlink(ABSTFN + "2", ABSTFN + "1")
529 +             self.assertRaises(OSError, ntpath.realpath, ABSTFN + "1",
strict=ALLOW_MISSING)
530 +
531 +             self.assertRaises(OSError, ntpath.realpath, ABSTFN + "2",
strict=ALLOW_MISSING)
532 +
533 +             self.assertRaises(OSError, ntpath.realpath, ABSTFN + "1\\x",
strict=ALLOW_MISSING)
534 +

```

```

535 +
536 +     # Windows eliminates '..' components before resolving links;
537 +     # realpath is not expected to raise if this removes the loop.
538 +     self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\.."),
539 +                          ntpath.dirname(ABSTFN))
540 +     self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\..\\x"),
541 +                          ntpath.dirname(ABSTFN) + "\\x")
542 +
543 +     os.symlink(ABSTFN + "x", ABSTFN + "y")
544 +     self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\..\\x"
545 +                                     + ntpath.basename(ABSTFN) + "y"),
546 +                          ABSTFN + "x")
547 +     self.assertRaises(
548 +         OSError, ntpath.realpath,
549 +         ABSTFN + "1\\..\\x" + ntpath.basename(ABSTFN) + "1",
550 +         strict=ALLOW_MISSING)
551 +
552 +     os.symlink(ntpath.basename(ABSTFN) + "a\\b", ABSTFN + "a")
553 +     self.assertRaises(OSError, ntpath.realpath, ABSTFN + "a",
554 +                       strict=ALLOW_MISSING)
555 +
556 +     os.symlink("../x" + ntpath.basename(ntpath.dirname(ABSTFN))
557 +               + "\\x" + ntpath.basename(ABSTFN) + "c", ABSTFN + "c")
558 +     self.assertRaises(OSError, ntpath.realpath, ABSTFN + "c",
559 +                       strict=ALLOW_MISSING)
560 +
561 +     # Test using relative path as well.
562 +     self.assertRaises(OSError, ntpath.realpath, ntpath.basename(ABSTFN),
563 +                       strict=ALLOW_MISSING)
564 +
565 +     @support.skip_unless_symlink
566 +     @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
567 +     @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
568 +     def test_realpath_symlink_prefix(self, kwargs):
443 569         ABSTFN = ntpath.abspath(support.TESTFN)
444 570         self.addCleanup(support.unlink, ABSTFN + "3")
445 571         self.addCleanup(support.unlink, "\\x\\?\\x" + ABSTFN + "3.")
@@ -454,9 +580,9 @@ def test_realpath_symlink_prefix(self):
454 580         f.write(b'1')
455 581         os.symlink("\\x\\?\\x" + ABSTFN + "3.", ABSTFN + "3.link")

```

```

456 582
457 - self.assertPathEqual(ntpath.realpath(ABSTFN + "3link"),
583 + self.assertPathEqual(ntpath.realpath(ABSTFN + "3link", **kwargs),
458 584 ABSTFN + "3")
459 - self.assertPathEqual(ntpath.realpath(ABSTFN + "3.link"),
585 + self.assertPathEqual(ntpath.realpath(ABSTFN + "3.link", **kwargs),
460 586 "\\\?\" + ABSTFN + "3.")
461 587
462 588 # Resolved paths should be usable to open target files
⚡ @@ -466,14 +592,17 @@ def test_realpath_symlink_prefix(self):
466 592 self.assertEqual(f.read(), b'1')
467 593
468 594 # When the prefix is included, it is not stripped
469 - self.assertPathEqual(ntpath.realpath("\\\\?\" + ABSTFN + "3link"),
595 + self.assertPathEqual(ntpath.realpath("\\\\?\" + ABSTFN + "3link",
**kwargs),
470 596 "\\\?\" + ABSTFN + "3")
471 - self.assertPathEqual(ntpath.realpath("\\\\?\" + ABSTFN + "3.link"),
597 + self.assertPathEqual(ntpath.realpath("\\\\?\" + ABSTFN + "3.link",
**kwargs),
472 598 "\\\?\" + ABSTFN + "3.")
473 599
474 600 @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
475 601 def test_realpath_nul(self):
476 602 tester("ntpath.realpath('NUL')", r'\\.NUL')
603 + tester("ntpath.realpath('NUL', strict=False)", r'\\.NUL')
604 + tester("ntpath.realpath('NUL', strict=True)", r'\\.NUL')
605 + tester("ntpath.realpath('NUL', strict=ALLOW_MISSING)", r'\\.NUL')
477 606
478 607 @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
479 608 @unittest.skipUnless(HAVE_GETSHORTPATHNAME, 'need _getshortpathname')
⚡ @@ -497,12 +626,20 @@ def test_realpath_cwd(self):
497 626
498 627 self.assertPathEqual(test_file_long, ntpath.realpath(test_file_short))
499 628
500 - with support.change_cwd(test_dir_long):
501 - self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
502 - with support.change_cwd(test_dir_long.lower()):
503 - self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
504 - with support.change_cwd(test_dir_short):

```

```

505 - self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
629 +     for kwargs in {}, {'strict': True}, {'strict': ALLOW_MISSING}:
630 +         with self.subTest(**kwargs):
631 +             with support.change_cwd(test_dir_long):
632 +                 self.assertPathEqual(
633 +                     test_file_long,
634 +                     ntpath.realpath("file.txt", **kwargs))
635 +             with support.change_cwd(test_dir_long.lower()):
636 +                 self.assertPathEqual(
637 +                     test_file_long,
638 +                     ntpath.realpath("file.txt", **kwargs))
639 +             with support.change_cwd(test_dir_short):
640 +                 self.assertPathEqual(
641 +                     test_file_long,
642 +                     ntpath.realpath("file.txt", **kwargs))
506 643
507 644     def test_expandvars(self):
508 645         with support.EnvironmentVarGuard() as env:

```

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 2025-4517.  
6 +
```

**Comments** 0