python / **cpython** Public

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

Commit 9c1110e

6 people authored on Jun 3, 2025 · ✕ 24 / 26 · Partially verified

```
[3.10] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (#135070)

Addresses CVEs 2024-12718, 2025-4138, 2025-4330, and 2025-4517.
(cherry picked from commit 3612d8f)
(cherry picked from commit c358142)
(cherry picked from commit 371b4ea)

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>
```

3.10 (#135070) · v3.10.20 ••• v3.10.18

1 parent c71ea4b commit 9c1110e

11 files changed +944 -134

Top

Filter files…

- Doc
  - library
    - os.path.rst
    - tarfile.rst
  - whatsnew
    - 3.10.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

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
        path is
    367 +   re-raised.
    368 +   In particular, :exc:`FileNotFoundError` is raised if *path* does not exist,
```

```
      369 +    or another :exc:`OSError` if it is otherwise inaccessible.
      370 +
      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

```
          @@ -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
```

```
                 +    as :attr:`!BaseException.__context__`.
                 +
                 +    .. versionadded:: next
                 +
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 ``'data'`` filter.
955  964      In addition to what ``tar_filter`` does:
956  965
                 +    - Normalize link targets (:attr:`TarInfo.linkname`) using
                 +      :func:`os.path.normpath`.
                 +      Note that this removes internal ``..`` components, which may change the
                 +      meaning of the link if the path in :attr:`!TarInfo.linkname` traverses
                 +      symbolic links.
                 +
957  972      - :ref:`Refuse <tarfile-extraction-refuse>` 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 ``TarInfo`` member.
984  999
                 +    .. versionchanged:: next
                 +
                 +       Link targets are now normalized.
                 +
985  1004
986  1005   .. _tarfile-extraction-refuse:
987  1006
                    @@ -1008,6 +1027,7 @@ Here is an incomplete list of things to consider:
1008 1027   * Extract to a :func:`new temporary directory <tempfile.mkdtemp>`
1009 1028     to prevent e.g. exploiting pre-existing links, and to make it easier to
1010 1029     clean up after a failed extraction.
                 + * 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/3.10.rst

```
      @@ -2394,3 +2394,37 @@ email
2394  2394      check if the *strict* paramater is available.
2395  2395      (Contributed by Thomas Dwyer and Victor Stinner for :gh:`102988` to improve
2396  2396      the CVE-2023-27043 fix.)
      2397  +
      2398  +
      2399  + Notable changes in 3.10.18
      2400  + ==========================
      2401  +
      2402  + os.path
      2403  + -------
      2404  +
      2405  + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
      2406  +   :data:`os.path.ALLOW_MISSING`.
      2407  +   If used, errors other than :exc:`FileNotFoundError` will be re-raised;
      2408  +   the resulting path can be missing but it will be free of symlinks.
      2409  +   (Contributed by Petr Viktorin for CVE 2025-4517.)
      2410  +
      2411  + tarfile
      2412  + -------
      2413  +
      2414  + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
              to
      2415  +   avoid path traversal attacks.
      2416  +   (Contributed by Petr Viktorin in :gh:`127987` and CVE 2025-4138.)
      2417  + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
              attributes
      2418  +   when a directory was removed or replaced by another kind of file.
      2419  +   (Contributed by Petr Viktorin in :gh:`127987` and CVE 2024-12718.)
      2420  + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
      2421  +   now (re-)apply the extraction filter when substituting a link (hard or
      2422  +   symbolic) with a copy of another archive member, and when fixing up
      2423  +   directory attributes.
      2424  +   The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
      2425  +   (Contributed by Petr Viktorin for CVE 2025-4330 and CVE 2024-12718.)
      2426  + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
```

```
2427 +   no longer extract rejected members when
2428 +   :func:`~tarfile.TarFile.errorlevel` is zero.
2429 +   (Contributed by Matt Prodani and Petr Viktorin in :gh:`112887`
2430 +   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"]
```

```
                +            "samefile", "sameopenfile", "samestat", "commonpath",
                +            "ALLOW_MISSING"]
34  35
35  36      def _get_bothseps(path):
36  37          if isinstance(path, bytes):
                @@ -571,9 +572,10 @@ def abspath(path):
571 572          from nt import _getfinalpathname, readlink as _nt_readlink
572 573      except ImportError:
573 574          # realpath is a no-op on systems without _getfinalpathname support.
574         -        realpath = abspath
            +    def realpath(path, *, strict=False):
            +        return abspath(path)
575 577      else:
576         -    def _readlink_deep(path):
            +    def _readlink_deep(path, ignored_error=OSError):
577 579          # These error codes indicate that we should stop reading links and
578 580          # return the path we currently have.
579 581          # 1: ERROR_INVALID_FUNCTION
                @@ -606,7 +608,7 @@ def _readlink_deep(path):
606 608                          path = old_path
607 609                          break
608 610                  path = normpath(join(dirname(old_path), path))
609         -            except OSError as ex:
            +            except ignored_error as ex:
610 612                  if ex.winerror in allowed_winerror:
611 613                      break
612 614                  raise
                @@ -615,7 +617,7 @@ def _readlink_deep(path):
615 617                  break
616 618          return path
617 619
618         -    def _getfinalpathname_nonstrict(path):
            +    def _getfinalpathname_nonstrict(path, ignored_error=OSError):
619 621          # These error codes indicate that we should stop resolving the path
620 622          # and return the value we currently have.
621 623          # 1: ERROR_INVALID_FUNCTION
                @@ -642,17 +644,18 @@ def _getfinalpathname_nonstrict(path):
642 644              try:
```

```
643  645                      path = _getfinalpathname(path)
644  646                      return join(path, tail) if tail else path
645       -              except OSError as ex:
     647  +              except ignored_error as ex:
646  648                      if ex.winerror not in allowed_winerror:
647  649                          raise
648  650                      try:
649  651                          # The OS could not resolve this path fully, so we attempt
650  652                          # to follow the link ourselves. If we succeed, join the
                                tail
651  653                          # and return.
652       -                      new_path = _readlink_deep(path)
     654  +                      new_path = _readlink_deep(path,
     655  +                                                ignored_error=ignored_error)
653  656                      if new_path != path:
654  657                          return join(new_path, tail) if tail else new_path
655       -                  except OSError:
     658  +                  except ignored_error:
656  659                          # If we fail to readlink(), let's keep traversing
657  660                          pass
658  661                      path, name = split(path)
                          @@ -683,16 +686,24 @@ def realpath(path, *, strict=False):
683  686              if normcase(path) == normcase(devnull):
684  687                  return '\\\\.\\NUL'
685  688          had_prefix = path.startswith(prefix)
     689  +
     690  +        if strict is ALLOW_MISSING:
     691  +            ignored_error = FileNotFoundError
     692  +            strict = True
     693  +        elif strict:
     694  +            ignored_error = ()
     695  +        else:
     696  +            ignored_error = OSError
     697  +
686  698          if not had_prefix and not isabs(path):
687  699              path = join(cwd, path)
688  700          try:
689  701              path = _getfinalpathname(path)
690  702              initial_winerror = 0
```

```
691        -         except OSError as ex:
692        -             if strict:
693        -                 raise
           703 +         except ignored_error as ex:
694    704                 initial_winerror = ex.winerror
695        -             path = _getfinalpathname_nonstrict(path)
           705 +             path = _getfinalpathname_nonstrict(path,
           706 +                                                ignored_error=ignored_error)
696    707             # The path returned by _getfinalpathname will always start with \\?\ -
697    708             # strip off that prefix unless it was already provided on the original
698    709             # 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):
@@ -407,6 +407,15 @@ def _joinrealpath(path, rest, strict, seen):
407    407             sep = '/'
408    408             curdir = '.'
409    409             pardir = '..'
           410 +         getcwd = os.getcwd
           411 +     if strict is ALLOW_MISSING:
           412 +         ignored_error = FileNotFoundError
           413 +     elif strict:
           414 +         ignored_error = ()
           415 +     else:
           416 +         ignored_error = OSError
           417 +
           418 +     maxlinks = None
410    419
411    420         if isabs(rest):
412    421             rest = rest[1:]
```

```
@@ -429,9 +438,7 @@ def _joinrealpath(path, rest, strict, seen):
429 438             newpath = join(path, name)
430 439             try:
431 440                 st = os.lstat(newpath)
432     -           except OSError:
433     -               if strict:
434     -                   raise
    441 +           except ignored_error:
435 442                 is_link = False
436 443             else:
437 444                 is_link = stat.S_ISLNK(st.st_mode)
```

Lib/tarfile.py

```
@@ -750,10 +750,22 @@ def __init__(self, tarinfo, path):
750 750             super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
751 751                              + 'which is outside the destination')
752 752
    753 + class LinkFallbackError(FilterError):
    754 +     def __init__(self, tarinfo, path):
    755 +         self.tarinfo = tarinfo
    756 +         self._path = path
    757 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
    758 +                          + f'copy of {path!r}, which was rejected')
    759 +
    760 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
    761 + # we consider to be issues with the argument, rather than a bug in the
    762 + # filter function
    763 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
    764 +
753 765   def _get_filtered_attrs(member, dest_path, for_data=True):
754 766       new_attrs = {}
755 767       name = member.name
756     -     dest_path = os.path.realpath(dest_path)
    768 +     dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
757 769       # Strip leading / (tar's directory separator) from filenames.
758 770       # Include os.sep (target OS directory separator) as well.
759 771       if name.startswith(('/', os.sep)):
```

```
@@ -763,7 +775,8 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
```

```
 763   775             # For example, 'C:/foo' on Windows.
 764   776             raise AbsolutePathError(member)
 765   777         # Ensure we stay in the destination
 766       -     target_path = os.path.realpath(os.path.join(dest_path, name))
       778 +     target_path = os.path.realpath(os.path.join(dest_path, name),
       779 +                                    strict=os.path.ALLOW_MISSING)
 767   780         if os.path.commonpath([target_path, dest_path]) != dest_path:
 768   781             raise OutsideDestinationError(member, target_path)
 769   782         # Limit permissions (no high bits, and go-w)
         ⇣       @@ -801,14 +814,18 @@ def _get_filtered_attrs(member, dest_path,
         ⇡       for_data=True):
 801   814             if member.islnk() or member.issym():
 802   815                 if os.path.isabs(member.linkname):
 803   816                     raise AbsoluteLinkError(member)
       817 +             normalized = os.path.normpath(member.linkname)
       818 +             if normalized != member.linkname:
       819 +                 new_attrs['linkname'] = normalized
 804   820                 if member.issym():
 805   821                     target_path = os.path.join(dest_path,
 806   822                                                os.path.dirname(name),
 807   823                                                member.linkname)
 808   824                 else:
 809   825                     target_path = os.path.join(dest_path,
 810   826                                                member.linkname)
 811       -             target_path = os.path.realpath(target_path)
       827 +             target_path = os.path.realpath(target_path,
       828 +                                            strict=os.path.ALLOW_MISSING)
 812   829                 if os.path.commonpath([target_path, dest_path]) != dest_path:
 813   830                     raise LinkOutsideDestinationError(member, target_path)
 814   831         return new_attrs
         ⇣       @@ -2282,30 +2299,58 @@ def extractall(self, path=".", members=None, *,
         ⇡       numeric_owner=False,
2282  2299             members = self
2283  2300
2284  2301             for member in members:
2285      -                 tarinfo = self._get_extract_tarinfo(member, filter_function,
                          path)
      2302 +             tarinfo, unfiltered = self._get_extract_tarinfo(
      2303 +                 member, filter_function, path)
2286  2304                 if tarinfo is None:
```

```
2287  2305                      continue
2288  2306                  if tarinfo.isdir():
2289  2307                      # For directories, delay setting attributes until later,
2290  2308                      # since permissions can interfere with extraction and
2291  2309                      # extracting contents can reset mtime.
2292        -                   directories.append(tarinfo)
      2310  +                   directories.append(unfiltered)
2293  2311                  self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
2294        -                                   numeric_owner=numeric_owner)
      2312  +                                   numeric_owner=numeric_owner,
      2313  +                                   filter_function=filter_function)
2295  2314
2296  2315          # Reverse sort directories.
2297  2316          directories.sort(key=lambda a: a.name, reverse=True)
2298  2317
      2318  +
2299  2319          # Set correct owner, mtime and filemode on directories.
2300        -         for tarinfo in directories:
2301        -             dirpath = os.path.join(path, tarinfo.name)
      2320  +         for unfiltered in directories:
2302  2321              try:
      2322  +                   # Need to re-apply any filter, to take the *current*
                                filesystem
      2323  +                   # state into account.
      2324  +                   try:
      2325  +                       tarinfo = filter_function(unfiltered, path)
      2326  +                   except _FILTER_ERRORS as exc:
      2327  +                       self._log_no_directory_fixup(unfiltered, repr(exc))
      2328  +                       continue
      2329  +                   if tarinfo is None:
      2330  +                       self._log_no_directory_fixup(unfiltered,
      2331  +                                                    'excluded by filter')
      2332  +                       continue
      2333  +                   dirpath = os.path.join(path, tarinfo.name)
      2334  +                   try:
      2335  +                       lstat = os.lstat(dirpath)
      2336  +                   except FileNotFoundError:
      2337  +                       self._log_no_directory_fixup(tarinfo, 'missing')
      2338  +                       continue
      2339  +                   if not stat.S_ISDIR(lstat.st_mode):
```

```
                                # This is no longer a directory; presumably a later
                                # member overwrote the entry.
                                self._log_no_directory_fixup(tarinfo, 'not a directory')
                                continue
                    self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
                    self.utime(tarinfo, dirpath)
                    self.chmod(tarinfo, dirpath)
                except ExtractError as e:
                    self._handle_nonfatal_error(e)

    def _log_no_directory_fixup(self, member, reason):
        self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
                  (member.name, reason))

    def extract(self, member, path="", set_attrs=True, *,
                numeric_owner=False,
                filter=None):
        """Extract a member from the archive to the current working
           directory,
```

```
           String names of common filters are accepted.
        """
        filter_function = self._get_filter_function(filter)
        tarinfo = self._get_extract_tarinfo(member, filter_function, path)
        tarinfo, unfiltered = self._get_extract_tarinfo(
            member, filter_function, path)
        if tarinfo is not None:
            self._extract_one(tarinfo, path, set_attrs, numeric_owner)

    def _get_extract_tarinfo(self, member, filter_function, path):
        """Get filtered TarInfo (or None) from member, which might be a
           str"""
        """Get (filtered, unfiltered) TarInfos from *member*

        *member* might be a string.

        Return (None, None) if not found.
        """
```

```
2330  2382            if isinstance(member, str):
2331        -             tarinfo = self.getmember(member)
      2383  +             unfiltered = self.getmember(member)
2332  2384            else:
2333        -             tarinfo = member
      2385  +             unfiltered = member
2334  2386
2335        -         unfiltered = tarinfo
      2387  +         filtered = None
2336  2388            try:
2337        -             tarinfo = filter_function(tarinfo, path)
      2389  +             filtered = filter_function(unfiltered, path)
2338  2390            except (OSError, FilterError) as e:
2339  2391                self._handle_fatal_error(e)
2340  2392            except ExtractError as e:
2341  2393                self._handle_nonfatal_error(e)
2342        -         if tarinfo is None:
      2394  +         if filtered is None:
2343  2395                self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2344        -             return None
      2396  +             return None, None
      2397  +
2345  2398            # Prepare the link target for makelink().
2346        -         if tarinfo.islnk():
2347        -             tarinfo = copy.copy(tarinfo)
2348        -             tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2349        -         return tarinfo
      2399  +         if filtered.islnk():
      2400  +             filtered = copy.copy(filtered)
      2401  +             filtered._link_target = os.path.join(path, filtered.linkname)
      2402  +         return filtered, unfiltered
2350  2403
2351        -     def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2352        -         """Extract from filtered tarinfo to disk"""
      2404  +     def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
      2405  +                      filter_function=None):
      2406  +         """Extract from filtered tarinfo to disk.
      2407  +
      2408  +            filter_function is only used when extracting a *different*
      2409  +            member (e.g. as fallback to creating a symlink)
```

```
                2410 +          """
2353            2411            self._check("r")
2354            2412
2355            2413            try:
2356            2414                self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2357            2415                                     set_attrs=set_attrs,
2358                 -                                   numeric_owner=numeric_owner)
                2416 +                                   numeric_owner=numeric_owner,
                2417 +                                   filter_function=filter_function,
                2418 +                                   extraction_root=path)
2359            2419            except OSError as e:
2360            2420                self._handle_fatal_error(e)
2361            2421            except ExtractError as e:
                   ↓
                   ↑        @@ -2413,9 +2473,13 @@ def extractfile(self, member):
2413            2473                return None
2414            2474
2415            2475        def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2416                 -                          numeric_owner=False):
2417                 -          """Extract the TarInfo object tarinfo to a physical
                2476 +                          numeric_owner=False, *, filter_function=None,
                2477 +                          extraction_root=None):
                2478 +          """Extract the filtered TarInfo object tarinfo to a physical
2418            2479               file called targetpath.
                2480 +
                2481 +          filter_function is only used when extracting a *different*
                2482 +          member (e.g. as fallback to creating a symlink)
2419            2483            """
2420            2484            # Fetch the TarInfo object for the given name
2421            2485            # and build the destination pathname, replacing
                   ↓
                   ↑        @@ -2444,7 +2508,10 @@ def _extract_member(self, tarinfo, targetpath,
                            set_attrs=True,
2444            2508            elif tarinfo.ischr() or tarinfo.isblk():
2445            2509                self.makedev(tarinfo, targetpath)
2446            2510            elif tarinfo.islnk() or tarinfo.issym():
2447                 -              self.makelink(tarinfo, targetpath)
                2511 +              self.makelink_with_filter(
                2512 +                  tarinfo, targetpath,
                2513 +                  filter_function=filter_function,
                2514 +                  extraction_root=extraction_root)
```

```
2448  2515            elif tarinfo.type not in SUPPORTED_TYPES:
2449  2516                self.makeunknown(tarinfo, targetpath)
2450  2517            else:
      ↕   ↕   @@ -2526,29 +2593,57 @@ def makedev(self, tarinfo, targetpath):
2526  2593                    os.makedev(tarinfo.devmajor, tarinfo.devminor))
2527  2594
2528  2595        def makelink(self, tarinfo, targetpath):
      2596  +         return self.makelink_with_filter(tarinfo, targetpath, None, None)
      2597  +
      2598  +     def makelink_with_filter(self, tarinfo, targetpath,
      2599  +                              filter_function, extraction_root):
2529  2600            """Make a (symbolic) link called targetpath. If it cannot be created
2530  2601              (platform limitation), we try to make a copy of the referenced file
2531  2602              instead of a link.
      2603  +
      2604  +           filter_function is only used when extracting a *different*
      2605  +           member (e.g. as fallback to creating a link).
2532  2606            """
      2607  +         keyerror_to_extracterror = False
2533  2608            try:
2534  2609                # For systems that support symbolic and hard links.
2535  2610                if tarinfo.issym():
2536  2611                    if os.path.lexists(targetpath):
2537  2612                        # Avoid FileExistsError on following os.symlink.
2538  2613                        os.unlink(targetpath)
2539  2614                    os.symlink(tarinfo.linkname, targetpath)
      2615  +                 return
2540  2616                else:
2541  2617                    if os.path.exists(tarinfo._link_target):
2542  2618                        os.link(tarinfo._link_target, targetpath)
2543        -                 else:
2544        -                     self._extract_member(self._find_link_target(tarinfo),
2545        -                                          targetpath)
      2619  +                 return
2546  2620            except symlink_exception:
      2621  +             keyerror_to_extracterror = True
      2622  +
      2623  +         try:
      2624  +             unfiltered = self._find_link_target(tarinfo)
```

```
        2625 +         except KeyError:
        2626 +             if keyerror_to_extracterror:
        2627 +                 raise ExtractError(
        2628 +                     "unable to resolve link inside archive") from None
        2629 +             else:
        2630 +                 raise
        2631 +
        2632 +         if filter_function is None:
        2633 +             filtered = unfiltered
        2634 +         else:
        2635 +             if extraction_root is None:
        2636 +                 raise ExtractError(
        2637 +                     "makelink_with_filter: if filter_function is not None, "
        2638 +                     + "extraction_root must also not be None")
2547    2639               try:
2548         -                 self._extract_member(self._find_link_target(tarinfo),
2549         -                                      targetpath)
2550         -             except KeyError:
2551         -                 raise ExtractError("unable to resolve link inside archive")
                from None
        2640 +                 filtered = filter_function(unfiltered, extraction_root)
        2641 +             except _FILTER_ERRORS as cause:
        2642 +                 raise LinkFallbackError(tarinfo, unfiltered.name) from cause
        2643 +         if filtered is not None:
        2644 +             self._extract_member(filtered, targetpath,
        2645 +                                  filter_function=filter_function,
        2646 +                                  extraction_root=extraction_root)
2552    2647
2553    2648       def chown(self, tarinfo, targetpath, numeric_owner):
2554    2649           """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 os_helper
7     9    from test.support import TestFailed
8     10   from test.support.os_helper import FakePath
           @@ -74,6 +76,27 @@ def tester(fn, wantResult):
74    76                   %(str(fn), str(wantResult), repr(gotResult)))
75    77
76    78
      79 + def _parameterize(*parameters):
      80 +     """Simplistic decorator to parametrize a test
      81 +
      82 +     Runs the decorated test multiple times in subTest, with a value from
      83 +     'parameters' passed as an extra positional argument.
      84 +     Calls doCleanups() after each run.
      85 +
      86 +     Not for general use. Intended to avoid indenting for easier backports.
      87 +
      88 +     See https://discuss.python.org/t/91827 for discussing generalizations.
      89 +     """
      90 +     def _parametrize_decorator(func):
      91 +         def _parameterized(self, *args, **kwargs):
      92 +             for parameter in parameters:
      93 +                 with self.subTest(parameter):
      94 +                     func(self, *args, parameter, **kwargs)
      95 +                 self.doCleanups()
      96 +         return _parameterized
      97 +     return _parametrize_decorator
      98 +
      99 +
77    100  class NtpathTestCase(unittest.TestCase):
78    101      def assertPathEqual(self, path1, path2):
79    102          if path1 == path2 or _norm(path1) == _norm(path2):
           @@ -244,6 +267,27 @@ def test_realpath_curdir(self):
244   267          tester("ntpath.realpath('.\\.')", expected)
245   268          tester("ntpath.realpath('\\'.join(['.'] * 100))", expected)
246   269
      270 +     def test_realpath_curdir_strict(self):
      271 +         expected = ntpath.normpath(os.getcwd())
```

```
272 +         tester("ntpath.realpath('.', strict=True)", expected)
273 +         tester("ntpath.realpath('./.', strict=True)", expected)
274 +         tester("ntpath.realpath('/'.join(['.'] * 100), strict=True)", expected)
275 +         tester("ntpath.realpath('.\\.', strict=True)", expected)
276 +         tester("ntpath.realpath('\\'.join(['.'] * 100), strict=True)",
      expected)
277 +
278 +     def test_realpath_curdir_missing_ok(self):
279 +         expected = ntpath.normpath(os.getcwd())
280 +         tester("ntpath.realpath('.', strict=ALLOW_MISSING)",
281 +                expected)
282 +         tester("ntpath.realpath('./.', strict=ALLOW_MISSING)",
283 +                expected)
284 +         tester("ntpath.realpath('/'.join(['.'] * 100), strict=ALLOW_MISSING)",
285 +                expected)
286 +         tester("ntpath.realpath('.\\.', strict=ALLOW_MISSING)",
287 +                expected)
288 +         tester("ntpath.realpath('\\'.join(['.'] * 100), strict=ALLOW_MISSING)",
289 +                expected)
290 +
247 291         def test_realpath_pardir(self):
248 292             expected = ntpath.normpath(os.getcwd())
249 293             tester("ntpath.realpath('..')", ntpath.dirname(expected))
      @@ -256,17 +300,43 @@ def test_realpath_pardir(self):
256 300             tester("ntpath.realpath('\\'.join(['..'] * 50))",
257 301                    ntpath.splitdrive(expected)[0] + '\\')
258 302
303 +     def test_realpath_pardir_strict(self):
304 +         expected = ntpath.normpath(os.getcwd())
305 +         tester("ntpath.realpath('..', strict=True)", ntpath.dirname(expected))
306 +         tester("ntpath.realpath('../..', strict=True)",
307 +                ntpath.dirname(ntpath.dirname(expected)))
308 +         tester("ntpath.realpath('/'.join(['..'] * 50), strict=True)",
309 +                ntpath.splitdrive(expected)[0] + '\\')
310 +         tester("ntpath.realpath('..\\..', strict=True)",
311 +                ntpath.dirname(ntpath.dirname(expected)))
312 +         tester("ntpath.realpath('\\'.join(['..'] * 50), strict=True)",
313 +                ntpath.splitdrive(expected)[0] + '\\')
314 +
315 +     def test_realpath_pardir_missing_ok(self):
```

```
                +         expected = ntpath.normpath(os.getcwd())
                +         tester("ntpath.realpath('..', strict=ALLOW_MISSING)",
                +                ntpath.dirname(expected))
                +         tester("ntpath.realpath('../..', strict=ALLOW_MISSING)",
                +                ntpath.dirname(ntpath.dirname(expected)))
                +         tester("ntpath.realpath('/'.join(['..'] * 50), strict=ALLOW_MISSING)",
                +                ntpath.splitdrive(expected)[0] + '\\')
                +         tester("ntpath.realpath('..\\..', strict=ALLOW_MISSING)",
                +                ntpath.dirname(ntpath.dirname(expected)))
                +         tester("ntpath.realpath('\\'.join(['..'] * 50), strict=ALLOW_MISSING)",
                +                ntpath.splitdrive(expected)[0] + '\\')
                +
259  328            @os_helper.skip_unless_symlink
260  329            @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
261       -        def test_realpath_basic(self):
             330 +      @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
             331 +      def test_realpath_basic(self, kwargs):
262  332              ABSTFN = ntpath.abspath(os_helper.TESTFN)
263  333              open(ABSTFN, "wb").close()
264  334              self.addCleanup(os_helper.unlink, ABSTFN)
265  335              self.addCleanup(os_helper.unlink, ABSTFN + "1")
266  336
267  337              os.symlink(ABSTFN, ABSTFN + "1")
268       -          self.assertPathEqual(ntpath.realpath(ABSTFN + "1"), ABSTFN)
269       -          self.assertPathEqual(ntpath.realpath(os.fsencode(ABSTFN + "1")),
             338 +          self.assertPathEqual(ntpath.realpath(ABSTFN + "1", **kwargs), ABSTFN)
             339 +          self.assertPathEqual(ntpath.realpath(os.fsencode(ABSTFN + "1"),
                **kwargs),
270  340                                  os.fsencode(ABSTFN))
271  341
272  342            @os_helper.skip_unless_symlink
                @@ -282,14 +352,15 @@ def test_realpath_strict(self):
282  352
283  353            @os_helper.skip_unless_symlink
284  354            @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
285       -        def test_realpath_relative(self):
             355 +      @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
             356 +      def test_realpath_relative(self, kwargs):
286  357              ABSTFN = ntpath.abspath(os_helper.TESTFN)
287  358              open(ABSTFN, "wb").close()
```

```
288  359            self.addCleanup(os_helper.unlink, ABSTFN)
289  360            self.addCleanup(os_helper.unlink, ABSTFN + "1")
290  361
291  362            os.symlink(ABSTFN, ntpath.relpath(ABSTFN + "1"))
292       -        self.assertPathEqual(ntpath.realpath(ABSTFN + "1"), ABSTFN)
     363  +        self.assertPathEqual(ntpath.realpath(ABSTFN + "1", **kwargs), ABSTFN)
293  364
294  365        @os_helper.skip_unless_symlink
295  366        @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
                 @@ -441,7 +512,62 @@ def test_realpath_symlink_loops_strict(self):
441  512
442  513        @os_helper.skip_unless_symlink
443  514        @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
444       -    def test_realpath_symlink_prefix(self):
     515  +    def test_realpath_symlink_loops_raise(self):
     516  +        # Symlink loops raise OSError in ALLOW_MISSING mode
     517  +        ABSTFN = ntpath.abspath(os_helper.TESTFN)
     518  +        self.addCleanup(os_helper.unlink, ABSTFN)
     519  +        self.addCleanup(os_helper.unlink, ABSTFN + "1")
     520  +        self.addCleanup(os_helper.unlink, ABSTFN + "2")
     521  +        self.addCleanup(os_helper.unlink, ABSTFN + "y")
     522  +        self.addCleanup(os_helper.unlink, ABSTFN + "c")
     523  +        self.addCleanup(os_helper.unlink, ABSTFN + "a")
     524  +        self.addCleanup(os_helper.unlink, ABSTFN + "x")
     525  +
     526  +        os.symlink(ABSTFN, ABSTFN)
     527  +        self.assertRaises(OSError, ntpath.realpath, ABSTFN,
                 strict=ALLOW_MISSING)
     528  +
     529  +        os.symlink(ABSTFN + "1", ABSTFN + "2")
     530  +        os.symlink(ABSTFN + "2", ABSTFN + "1")
     531  +        self.assertRaises(OSError, ntpath.realpath, ABSTFN + "1",
     532  +                          strict=ALLOW_MISSING)
     533  +        self.assertRaises(OSError, ntpath.realpath, ABSTFN + "2",
     534  +                          strict=ALLOW_MISSING)
     535  +        self.assertRaises(OSError, ntpath.realpath, ABSTFN + "1\\x",
     536  +                          strict=ALLOW_MISSING)
     537  +
     538  +        # Windows eliminates '..' components before resolving links;
```

```
539 +         # realpath is not expected to raise if this removes the loop.
540 +         self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\.."),
541 +                              ntpath.dirname(ABSTFN))
542 +         self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\..\\x"),
543 +                              ntpath.dirname(ABSTFN) + "\\x")
544 +
545 +         os.symlink(ABSTFN + "x", ABSTFN + "y")
546 +         self.assertPathEqual(ntpath.realpath(ABSTFN + "1\\..\\"
547 +                                              + ntpath.basename(ABSTFN) + "y"),
548 +                              ABSTFN + "x")
549 +         self.assertRaises(
550 +             OSError, ntpath.realpath,
551 +             ABSTFN + "1\\..\\" + ntpath.basename(ABSTFN) + "1",
552 +             strict=ALLOW_MISSING)
553 +
554 +         os.symlink(ntpath.basename(ABSTFN) + "a\\b", ABSTFN + "a")
555 +         self.assertRaises(OSError, ntpath.realpath, ABSTFN + "a",
556 +                           strict=ALLOW_MISSING)
557 +
558 +         os.symlink("..\\" + ntpath.basename(ntpath.dirname(ABSTFN))
559 +                 + "\\" + ntpath.basename(ABSTFN) + "c", ABSTFN + "c")
560 +         self.assertRaises(OSError, ntpath.realpath, ABSTFN + "c",
561 +                           strict=ALLOW_MISSING)
562 +
563 +         # Test using relative path as well.
564 +         self.assertRaises(OSError, ntpath.realpath, ntpath.basename(ABSTFN),
565 +                           strict=ALLOW_MISSING)
566 +
567 +     @os_helper.skip_unless_symlink
568 +     @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
569 +     @_parameterize({}, {'strict': True}, {'strict': ALLOW_MISSING})
570 +     def test_realpath_symlink_prefix(self, kwargs):
445 571           ABSTFN = ntpath.abspath(os_helper.TESTFN)
446 572           self.addCleanup(os_helper.unlink, ABSTFN + "3")
447 573           self.addCleanup(os_helper.unlink, "\\\\?\\" + ABSTFN + "3.")
    @@ -456,9 +582,9 @@ def test_realpath_symlink_prefix(self):
456 582               f.write(b'1')
457 583           os.symlink("\\\\?\\" + ABSTFN + "3.", ABSTFN + "3.link")
458 584
459     -         self.assertPathEqual(ntpath.realpath(ABSTFN + "3link"),
```

```
                +          self.assertPathEqual(ntpath.realpath(ABSTFN + "3link", **kwargs),
460     586                                     ABSTFN + "3")
461             -          self.assertPathEqual(ntpath.realpath(ABSTFN + "3.link"),
        587     +          self.assertPathEqual(ntpath.realpath(ABSTFN + "3.link", **kwargs),
462     588                                     "\\\\?\\" + ABSTFN + "3.")
463     589
464     590                # Resolved paths should be usable to open target files
                   @@ -468,14 +594,17 @@ def test_realpath_symlink_prefix(self):
468     594                    self.assertEqual(f.read(), b'1')
469     595
470     596                # When the prefix is included, it is not stripped
471             -          self.assertPathEqual(ntpath.realpath("\\\\?\\" + ABSTFN + "3link"),
        597     +          self.assertPathEqual(ntpath.realpath("\\\\?\\" + ABSTFN + "3link",
                           **kwargs),
472     598                                     "\\\\?\\" + ABSTFN + "3")
473             -          self.assertPathEqual(ntpath.realpath("\\\\?\\" + ABSTFN + "3.link"),
        599     +          self.assertPathEqual(ntpath.realpath("\\\\?\\" + ABSTFN + "3.link",
                           **kwargs),
474     600                                     "\\\\?\\" + ABSTFN + "3.")
475     601
476     602            @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
477     603            def test_realpath_nul(self):
478     604                tester("ntpath.realpath('NUL')", r'\\.\NUL')
        605     +          tester("ntpath.realpath('NUL', strict=False)", r'\\.\NUL')
        606     +          tester("ntpath.realpath('NUL', strict=True)", r'\\.\NUL')
        607     +          tester("ntpath.realpath('NUL', strict=ALLOW_MISSING)", r'\\.\NUL')
479     608
480     609            @unittest.skipUnless(HAVE_GETFINALPATHNAME, 'need _getfinalpathname')
481     610            @unittest.skipUnless(HAVE_GETSHORTPATHNAME, 'need _getshortpathname')
                   @@ -499,12 +628,20 @@ def test_realpath_cwd(self):
499     628
500     629                self.assertPathEqual(test_file_long, ntpath.realpath(test_file_short))
501     630
502             -          with os_helper.change_cwd(test_dir_long):
503             -              self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
504             -          with os_helper.change_cwd(test_dir_long.lower()):
505             -              self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
506             -          with os_helper.change_cwd(test_dir_short):
507             -              self.assertPathEqual(test_file_long, ntpath.realpath("file.txt"))
        631     +          for kwargs in {}, {'strict': True}, {'strict': ALLOW_MISSING}:
```

```
      632  +             with self.subTest(**kwargs):
      633  +                 with os_helper.change_cwd(test_dir_long):
      634  +                     self.assertPathEqual(
      635  +                         test_file_long,
      636  +                         ntpath.realpath("file.txt", **kwargs))
      637  +                 with os_helper.change_cwd(test_dir_long.lower()):
      638  +                     self.assertPathEqual(
      639  +                         test_file_long,
      640  +                         ntpath.realpath("file.txt", **kwargs))
      641  +                 with os_helper.change_cwd(test_dir_short):
      642  +                     self.assertPathEqual(
      643  +                         test_file_long,
      644  +                         ntpath.realpath("file.txt", **kwargs))
508   645
509   646         def test_expandvars(self):
510   647             with os_helper.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