python / cpython Public

Code Issues 5k+ Pull requests 2.1k Actions Projects Security and q

Commit 19de092

6 people authored on Jun 3, 2025 · 22 / 25 · Partially verified

```
[3.12] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (GH-135066)

Addresses CVEs 2024-12718, 2025-4138, 2025-4330, and 2025-4517.

(cherry picked from commit 3612d8f)

Co-authored-by: Łukasz Langa <lukasz@langa.pl>
Signed-off-by: Łukasz Langa <lukasz@langa.pl>
Co-authored-by: Petr Viktorin <encukou@gmail.com>
Co-authored-by: Seth Michael Larson <seth@python.org>
Co-authored-by: Adam Turner <9087854+AA-Turner@users.noreply.github.com>
Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>
```

3.12 (#135066) · v3.12.13 ••• v3.12.11

1 parent f3272d8 commit 19de092

11 files changed +1,064 -139

Top

Filter files…

- Doc
  - library
    - os.path.rst
    - tarfile.rst
  - whatsnew
    - 3.12.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

```diff
@@ -377,10 +377,26 @@ the :mod:`glob` module.)
     links encountered in the path (if they are supported by the operating
     system).

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

```
+    or another :exc:`OSError` if it is otherwise inaccessible.
+
+    If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
+    :exc:`FileNotFoundError` are re-raised (as with ``strict=True``).
+    Thus, the returned path will not contain any symbolic links, but the named
+    file and some of its parent directories may be missing.
     .. note::
        This function emulates the operating system's procedure for making a path
@@ -399,6 +415,15 @@ the :mod:`glob` module.)
     .. versionchanged:: 3.10
        The *strict* parameter was added.

+    .. versionchanged:: next
+       The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
+       was added.
+
+ .. data:: ALLOW_MISSING
+
+    Special value used for the *strict* argument in :func:`realpath`.
+
+    .. versionadded:: next

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

```

Doc/library/tarfile.rst

```
@@ -249,6 +249,15 @@ The :mod:`tarfile` module defines the following exceptions:
       Raised to refuse extracting a symbolic link pointing outside the destination
       directory.

+ .. exception:: LinkFallbackError
+
+    Raised to refuse emulating a link (hard or symbolic) by extracting another
+    archive member, when that member would be rejected by the filter location.
+    The exception that was raised to reject the replacement member is available
```

```
+    as :attr:`!BaseException.__context__`.
+
+    .. versionadded:: next
+
 252  261
 253  262   The following constants are available at the module level:
 254  263
@@ -1039,6 +1048,12 @@ reused in custom filters:
1039 1048     Implements the ``'data'`` filter.
1040 1049     In addition to what ``tar_filter`` does:
1041 1050
+    1051 +   - Normalize link targets (:attr:`TarInfo.linkname`) using
+    1052 +     :func:`os.path.normpath`.
+    1053 +     Note that this removes internal ``..`` components, which may change the
+    1054 +     meaning of the link if the path in :attr:`!TarInfo.linkname` traverses
+    1055 +     symbolic links.
+    1056 +
1042 1057     - :ref:`Refuse <tarfile-extraction-refuse>` to extract links (hard or soft)
1043 1058       that link to absolute paths, or ones that link outside the destination.
1044 1059
@@ -1067,6 +1082,10 @@ reused in custom filters:
1067 1082
1068 1083     Return the modified ``TarInfo`` member.
1069 1084
+    1085 +   .. versionchanged:: next
+    1086 +
+    1087 +     Link targets are now normalized.
+    1088 +
1070 1089
1071 1090   .. _tarfile-extraction-refuse:
1072 1091
@@ -1093,6 +1112,7 @@ Here is an incomplete list of things to consider:
1093 1112   * Extract to a :func:`new temporary directory <tempfile.mkdtemp>`
1094 1113     to prevent e.g. exploiting pre-existing links, and to make it easier to
1095 1114     clean up after a failed extraction.
+    1115 + * Disallow symbolic links if you do not need the functionality.
1096 1116   * When working with untrusted data, use external (e.g. OS-level) limits on
1097 1117     disk, memory and CPU usage.
```

```
1098  1118   * Check filenames against an allow-list of characters
```

Doc/whatsnew/3.12.rst

```
@@ -2320,3 +2320,37 @@ sys
2320  2320   * The previously undocumented special function :func:`sys.getobjects`,
2321  2321     which only exists in specialized builds of Python, may now return objects
2322  2322     from other interpreters than the one it's called in.
      2323 +
      2324 +
      2325 + Notable changes in 3.12.10
      2326 + ==========================
      2327 +
      2328 + os.path
      2329 + -------
      2330 +
      2331 + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
      2332 +   :data:`os.path.ALLOW_MISSING`.
      2333 +   If used, errors other than :exc:`FileNotFoundError` will be re-raised;
      2334 +   the resulting path can be missing but it will be free of symlinks.
      2335 +   (Contributed by Petr Viktorin for :cve:`2025-4517`.)
      2336 +
      2337 + tarfile
      2338 + -------
      2339 +
      2340 + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order to
      2341 +   avoid path traversal attacks.
      2342 +   (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2025-4138`.)
      2343 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory attributes
      2344 +   when a directory was removed or replaced by another kind of file.
      2345 +   (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2024-12718`.)
      2346 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
      2347 +   now (re-)apply the extraction filter when substituting a link (hard or
      2348 +   symbolic) with a copy of another archive member, and when fixing up
      2349 +   directory attributes.
      2350 +   The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
      2351 +   (Contributed by Petr Viktorin for :cve:`2025-4330` and :cve:`2024-12718`.)
      2352 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
```

```
+   no longer extract rejected members when
+   :func:`~tarfile.TarFile.errorlevel` is zero.
+   (Contributed by Matt Prodani and Petr Viktorin in :gh:`112887`
+   and :cve:`2025-4435`.)
```

Lib/genericpath.py

```
@@ -8,7 +8,7 @@

__all__ = ['commonprefix', 'exists', 'getatime', 'getctime', 'getmtime',
           'getsize', 'isdir', 'isfile', 'islink', 'samefile', 'sameopenfile',
-          'samestat']
+          'samestat', 'ALLOW_MISSING']


# Does a path exist?
@@ -165,3 +165,12 @@ def _check_arg_types(funcname, *args):
                            f'os.PathLike object, not {s.__class__.__name__!r}') from None
        if hasstr and hasbytes:
            raise TypeError("Can't mix strings and bytes in path components") from None
+
+ # A singleton with a true boolean value.
+ @object.__new__
+ class ALLOW_MISSING:
+     """Special value for use in realpath()."""
+     def __repr__(self):
+         return 'os.path.ALLOW_MISSING'
+     def __reduce__(self):
+         return self.__class__.__name__
```

Lib/ntpath.py

```
@@ -30,7 +30,8 @@
           "ismount", "expanduser","expandvars","normpath","abspath",
           "curdir","pardir","sep","pathsep","defpath","altsep",
           "extsep","devnull","realpath","supports_unicode_filenames","relpath",
-          "samefile", "sameopenfile", "samestat", "commonpath", "isjunction"]
```

```
+            "samefile", "sameopenfile", "samestat", "commonpath", "isjunction",
+            "ALLOW_MISSING"]

def _get_bothseps(path):
    if isinstance(path, bytes):
@@ -609,9 +610,10 @@ def abspath(path):
    from nt import _getfinalpathname, readlink as _nt_readlink
except ImportError:
    # realpath is a no-op on systems without _getfinalpathname support.
-    realpath = abspath
+    def realpath(path, *, strict=False):
+        return abspath(path)
else:
-    def _readlink_deep(path):
+    def _readlink_deep(path, ignored_error=OSError):
        # These error codes indicate that we should stop reading links and
        # return the path we currently have.
        # 1: ERROR_INVALID_FUNCTION
@@ -644,7 +646,7 @@ def _readlink_deep(path):
                        path = old_path
                        break
                    path = normpath(join(dirname(old_path), path))
-            except OSError as ex:
+            except ignored_error as ex:
                if ex.winerror in allowed_winerror:
                    break
                raise
@@ -653,7 +655,7 @@ def _readlink_deep(path):
                break
        return path

-    def _getfinalpathname_nonstrict(path):
+    def _getfinalpathname_nonstrict(path, ignored_error=OSError):
        # These error codes indicate that we should stop resolving the path
        # and return the value we currently have.
        # 1: ERROR_INVALID_FUNCTION
@@ -680,17 +682,18 @@ 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)
@@ -721,24 +724,32 @@ 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)
            try:
                path = _getfinalpathname(path)
                initial_winerror = 0
```

```
729  741            except ValueError as ex:
730  742                # gh-106242: Raised for embedded null characters
731     -              # In strict mode, we convert into an OSError.
     743 +              # In strict modes, we convert into an OSError.
732  744                # Non-strict mode returns the path as-is, since we've already
733  745                # made it absolute.
734  746                if strict:
735  747                    raise OSError(str(ex)) from None
736  748                path = normpath(path)
737     -          except OSError as ex:
738     -              if strict:
739     -                  raise
     749 +          except ignored_error as ex:
740  750                initial_winerror = ex.winerror
741     -              path = _getfinalpathname_nonstrict(path)
     751 +              path = _getfinalpathname_nonstrict(path,
     752 +                                                 ignored_error=ignored_error)
742  753            # The path returned by _getfinalpathname will always start with \\?\ -
743  754            # strip off that prefix unless it was already provided on the original
744  755            # 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", "isjunction"]
     38 +             "commonpath", "isjunction", "ALLOW_MISSING"]
39   39
40   40
41   41    def _get_sep(path):
     @@ -438,6 +438,15 @@ def _joinrealpath(path, rest, strict, seen):
438  438            sep = '/'
439  439            curdir = '.'
440  440            pardir = '..'
     441 +          getcwd = os.getcwd
     442 +      if strict is ALLOW_MISSING:
     443 +          ignored_error = FileNotFoundError
```

```
+     elif strict:
+         ignored_error = ()
+     else:
+         ignored_error = OSError
+
+     maxlinks = None

        if isabs(rest):
            rest = rest[1:]
```

@@ -460,9 +469,7 @@ def _joinrealpath(path, rest, strict, seen):

```
            newpath = join(path, name)
            try:
                st = os.lstat(newpath)
-         except OSError:
-             if strict:
-                 raise
+         except ignored_error:
                is_link = False
            else:
                is_link = stat.S_ISLNK(st.st_mode)
```

Lib/tarfile.py

@@ -752,10 +752,22 @@ def __init__(self, tarinfo, path):

```
            super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
                             + 'which is outside the destination')

+ class LinkFallbackError(FilterError):
+     def __init__(self, tarinfo, path):
+         self.tarinfo = tarinfo
+         self._path = path
+         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
+                          + f'copy of {path!r}, which was rejected')
+
+ # Errors caused by filters -- both "fatal" and "non-fatal" -- that
+ # we consider to be issues with the argument, rather than a bug in the
+ # filter function
+ _FILTER_ERRORS = (FilterError, OSError, ExtractError)
+
  def _get_filtered_attrs(member, dest_path, for_data=True):
```

```
 768        new_attrs = {}
 769        name = member.name
-        dest_path = os.path.realpath(dest_path)
+ 770    dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
 771        # Strip leading / (tar's directory separator) from filenames.
 772        # Include os.sep (target OS directory separator) as well.
 773        if name.startswith(('/', os.sep)):
@@ -765,7 +777,8 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
 777            # For example, 'C:/foo' on Windows.
 778            raise AbsolutePathError(member)
 779        # Ensure we stay in the destination
-        target_path = os.path.realpath(os.path.join(dest_path, name))
+ 780    target_path = os.path.realpath(os.path.join(dest_path, name),
+ 781                                   strict=os.path.ALLOW_MISSING)
 782        if os.path.commonpath([target_path, dest_path]) != dest_path:
 783            raise OutsideDestinationError(member, target_path)
 784        # Limit permissions (no high bits, and go-w)
@@ -803,14 +816,18 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
 816            if member.islnk() or member.issym():
 817                if os.path.isabs(member.linkname):
 818                    raise AbsoluteLinkError(member)
+ 819                normalized = os.path.normpath(member.linkname)
+ 820                if normalized != member.linkname:
+ 821                    new_attrs['linkname'] = normalized
 822                if member.issym():
 823                    target_path = os.path.join(dest_path,
 824                                               os.path.dirname(name),
 825                                               member.linkname)
 826                else:
 827                    target_path = os.path.join(dest_path,
 828                                               member.linkname)
-                target_path = os.path.realpath(target_path)
+ 829                target_path = os.path.realpath(target_path,
+ 830                                               strict=os.path.ALLOW_MISSING)
 831                if os.path.commonpath([target_path, dest_path]) != dest_path:
 832                    raise LinkOutsideDestinationError(member, target_path)
 833        return new_attrs
```

```diff
@@ -2291,30 +2308,58 @@ def extractall(self, path=".", members=None, *, numeric_owner=False,
 2291  2308                members = self
 2292  2309
 2293  2310            for member in members:
 2294     -                tarinfo = self._get_extract_tarinfo(member, filter_function, path)
      2311 +                tarinfo, unfiltered = self._get_extract_tarinfo(
      2312 +                    member, filter_function, path)
 2295  2313                if tarinfo is None:
 2296  2314                    continue
 2297  2315                if tarinfo.isdir():
 2298  2316                    # For directories, delay setting attributes until later,
 2299  2317                    # since permissions can interfere with extraction and
 2300  2318                    # extracting contents can reset mtime.
 2301     -                    directories.append(tarinfo)
      2319 +                    directories.append(unfiltered)
 2302  2320                self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
 2303     -                                  numeric_owner=numeric_owner)
      2321 +                                  numeric_owner=numeric_owner,
      2322 +                                  filter_function=filter_function)
 2304  2323
 2305  2324            # Reverse sort directories.
 2306  2325            directories.sort(key=lambda a: a.name, reverse=True)
 2307  2326
      2327 +
 2308  2328            # Set correct owner, mtime and filemode on directories.
 2309     -            for tarinfo in directories:
 2310     -                dirpath = os.path.join(path, tarinfo.name)
      2329 +            for unfiltered in directories:
 2311  2330                try:
      2331 +                    # Need to re-apply any filter, to take the *current* filesystem
      2332 +                    # state into account.
      2333 +                    try:
      2334 +                        tarinfo = filter_function(unfiltered, path)
      2335 +                    except _FILTER_ERRORS as exc:
      2336 +                        self._log_no_directory_fixup(unfiltered, repr(exc))
      2337 +                        continue
      2338 +                    if tarinfo is None:
```

```
+                    self._log_no_directory_fixup(unfiltered,
+                                                 'excluded by filter')
+                    continue
+                dirpath = os.path.join(path, tarinfo.name)
+                try:
+                    lstat = os.lstat(dirpath)
+                except FileNotFoundError:
+                    self._log_no_directory_fixup(tarinfo, 'missing')
+                    continue
+                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,
@@ -2330,41 +2375,56 @@ def extract(self, member, path="", set_attrs=True, *, numeric_owner=False,
           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)

```

```diff
@@ -2337,2338 +2383,2384,2385,2386,2387,2388,2389,2390,2391,2392,2393,2394,2395,2396,2397,2398,2399,2400,2401,2402,2403,2404,2405,2406,2407,2408,2409,2410 @@
 2337 2383     def _get_extract_tarinfo(self, member, filter_function, path):
-2338      """Get filtered TarInfo (or None) from member, which might be a str"""
     2384 +    """Get (filtered, unfiltered) TarInfos from *member*
     2385 +
     2386 +    *member* might be a string.
     2387 +
     2388 +    Return (None, None) if not found.
     2389 +    """
     2390 +
 2339 2391     if isinstance(member, str):
-2340         tarinfo = self.getmember(member)
     2392 +        unfiltered = self.getmember(member)
 2341 2393     else:
-2342         tarinfo = member
     2394 +        unfiltered = member
 2343 2395
-2344     unfiltered = tarinfo
     2396 +    filtered = None
 2345 2397     try:
-2346         tarinfo = filter_function(tarinfo, path)
     2398 +        filtered = filter_function(unfiltered, path)
 2347 2399     except (OSError, FilterError) as e:
 2348 2400         self._handle_fatal_error(e)
 2349 2401     except ExtractError as e:
 2350 2402         self._handle_nonfatal_error(e)
-2351     if tarinfo is None:
     2403 +    if filtered is None:
 2352 2404         self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
-2353         return None
     2405 +        return None, None
     2406 +
 2354 2407     # Prepare the link target for makelink().
-2355     if tarinfo.islnk():
-2356         tarinfo = copy.copy(tarinfo)
-2357         tarinfo._link_target = os.path.join(path, tarinfo.linkname)
-2358     return tarinfo
     2408 +    if filtered.islnk():
     2409 +        filtered = copy.copy(filtered)
     2410 +        filtered._link_target = os.path.join(path, filtered.linkname)
```

```
+         return filtered, unfiltered

-     def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
-         """Extract from filtered tarinfo to disk"""
+     def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
+                      filter_function=None):
+         """Extract from filtered tarinfo to disk.
+
+            filter_function is only used when extracting a *different*
+            member (e.g. as fallback to creating a symlink)
+         """
            self._check("r")

            try:
                self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
                                     set_attrs=set_attrs,
-                                    numeric_owner=numeric_owner)
+                                    numeric_owner=numeric_owner,
+                                    filter_function=filter_function,
+                                    extraction_root=path)
            except OSError as e:
                self._handle_fatal_error(e)
            except ExtractError as e:
@@ -2422,9 +2482,13 @@ def extractfile(self, member):
                return None

        def _extract_member(self, tarinfo, targetpath, set_attrs=True,
-                           numeric_owner=False):
-         """Extract the TarInfo object tarinfo to a physical
+                           numeric_owner=False, *, filter_function=None,
+                           extraction_root=None):
+         """Extract the filtered TarInfo object tarinfo to a physical
            file called targetpath.
+
+            filter_function is only used when extracting a *different*
+            member (e.g. as fallback to creating a symlink)
            """
            # Fetch the TarInfo object for the given name
            # and build the destination pathname, replacing
```

```diff
@@ -2453,7 +2517,10 @@ def _extract_member(self, tarinfo, targetpath, set_attrs=True,
            elif tarinfo.ischr() or tarinfo.isblk():
                self.makedev(tarinfo, targetpath)
            elif tarinfo.islnk() or tarinfo.issym():
-               self.makelink(tarinfo, targetpath)
+               self.makelink_with_filter(
+                   tarinfo, targetpath,
+                   filter_function=filter_function,
+                   extraction_root=extraction_root)
            elif tarinfo.type not in SUPPORTED_TYPES:
                self.makeunknown(tarinfo, targetpath)
            else:
@@ -2536,29 +2603,57 @@ def makedev(self, tarinfo, targetpath):
                     os.makedev(tarinfo.devmajor, tarinfo.devminor))

        def makelink(self, tarinfo, targetpath):
+           return self.makelink_with_filter(tarinfo, targetpath, None, None)
+
+       def makelink_with_filter(self, tarinfo, targetpath,
+                                filter_function, extraction_root):
            """Make a (symbolic) link called targetpath. If it cannot be created
              (platform limitation), we try to make a copy of the referenced file
              instead of a link.
+
+             filter_function is only used when extracting a *different*
+             member (e.g. as fallback to creating a link).
            """
+           keyerror_to_extracterror = False
            try:
                # For systems that support symbolic and hard links.
                if tarinfo.issym():
                    if os.path.lexists(targetpath):
                        # Avoid FileExistsError on following os.symlink.
                        os.unlink(targetpath)
                    os.symlink(tarinfo.linkname, targetpath)
+                   return
                else:
                    if os.path.exists(tarinfo._link_target):
```

```
2552  2628                        os.link(tarinfo._link_target, targetpath)
2553  -                 else:
2554  -                     self._extract_member(self._find_link_target(tarinfo),
2555  -                                          targetpath)
      2629  +                     return
2556  2630            except symlink_exception:
      2631  +             keyerror_to_extracterror = True
      2632  +
      2633  +         try:
      2634  +             unfiltered = self._find_link_target(tarinfo)
      2635  +         except KeyError:
      2636  +             if keyerror_to_extracterror:
      2637  +                 raise ExtractError(
      2638  +                     "unable to resolve link inside archive") from None
      2639  +             else:
      2640  +                 raise
      2641  +
      2642  +         if filter_function is None:
      2643  +             filtered = unfiltered
      2644  +         else:
      2645  +             if extraction_root is None:
      2646  +                 raise ExtractError(
      2647  +                     "makelink_with_filter: if filter_function is not None, "
      2648  +                     + "extraction_root must also not be None")
2557  2649                try:
2558  -                 self._extract_member(self._find_link_target(tarinfo),
2559  -                                      targetpath)
2560  -             except KeyError:
2561  -                 raise ExtractError("unable to resolve link inside archive")
        from None
      2650  +                 filtered = filter_function(unfiltered, extraction_root)
      2651  +             except _FILTER_ERRORS as cause:
      2652  +                 raise LinkFallbackError(tarinfo, unfiltered.name) from cause
      2653  +         if filtered is not None:
      2654  +             self._extract_member(filtered, targetpath,
      2655  +                                  filter_function=filter_function,
      2656  +                                  extraction_root=extraction_root)
2562  2657
2563  2658        def chown(self, tarinfo, targetpath, numeric_owner):
2564  2659            """Set owner of targetpath according to tarinfo. If numeric_owner
```

Lib/test/test_ntpath.py

Load Diff

Large diffs are not rendered by default.

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 @@
+ Fixes multiple issues that allowed ``tarfile`` extraction filters
+ (``filter="data"`` and ``filter="tar"``) to be bypassed using crafted
+ symlinks and hard links.
+
+ Addresses :cve:`2024-12718`, :cve:`2025-4138`, :cve:`2025-4330`, and :cve:`2025-4517`.
+
```

Comments 0