python / **cpython** Public

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

Commit 3612d8f

5 people authored on Jun 3, 2025 · ✕ 106 / 110 · Partially verified

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

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

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>
```

main (#135037) · v3.15.0a8 ••• v3.15.0a1

1 parent ec12559 commit 3612d8f

11 files changed +966 -169

Top

Filter files…

- Doc
  - library
    - os.path.rst
    - tarfile.rst
  - whatsnew
    - 3.15.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

```
@@ -408,9 +408,26 @@ the :mod:`glob` module.)
system). On Windows, this function will also resolve MS-DOS (also called 8.3)
style names such as ``C:\\PROGRA~1`` to ``C:\\Program Files``.

-   If a path doesn't exist or a symlink loop is encountered, and *strict* is
-   ``True``, :exc:`OSError` is raised. If *strict* is ``False`` these errors
-   are ignored, and so the result might be missing or otherwise inaccessible.
+   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.
+
```

```
                427 +    If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
                428 +    :exc:`FileNotFoundError` are re-raised (as with ``strict=True``).
                429 +    Thus, the returned path will not contain any symbolic links, but the named
                430 +    file and some of its parent directories may be missing.
414             431
415             432        .. note::
416             433           This function emulates the operating system's procedure for making a path
                   @@ -429,6 +446,15 @@ the :mod:`glob` module.)
429             446        .. versionchanged:: 3.10
430             447           The *strict* parameter was added.
431             448
                449 +    .. versionchanged:: next
                450 +       The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
                451 +       was added.
                452 +
                453 + .. data:: ALLOW_MISSING
                454 +
                455 +    Special value used for the *strict* argument in :func:`realpath`.
                456 +
                457 +    .. versionadded:: next
432             458
433             459     .. function:: relpath(path, start=os.curdir)
434             460
```

Doc/library/tarfile.rst

```
                   @@ -255,6 +255,15 @@ The :mod:`tarfile` module defines the following
                   exceptions:
255             255        Raised to refuse extracting a symbolic link pointing outside the
                   destination
256             256        directory.
257             257
                258 + .. exception:: LinkFallbackError
                259 +
                260 +    Raised to refuse emulating a link (hard or symbolic) by extracting another
                261 +    archive member, when that member would be rejected by the filter location.
                262 +    The exception that was raised to reject the replacement member is
                   available
                263 +    as :attr:`!BaseException.__context__`.
                264 +
```

```
                265 +    .. versionadded:: next
                266 +
 258            267
 259            268     The following constants are available at the module level:
 260            269
                        @@ -1068,6 +1077,12 @@ reused in custom filters:
1068           1077       Implements the ``'data'`` filter.
1069           1078       In addition to what ``tar_filter`` does:
1070           1079
                1080 +   - Normalize link targets (:attr:`TarInfo.linkname`) using
                1081 +     :func:`os.path.normpath`.
                1082 +     Note that this removes internal ``..`` components, which may change the
                1083 +     meaning of the link if the path in :attr:`!TarInfo.linkname` traverses
                1084 +     symbolic links.
                1085 +
1071           1086       - :ref:`Refuse <tarfile-extraction-refuse>` to extract links (hard or soft)
1072           1087         that link to absolute paths, or ones that link outside the destination.
1073           1088
                        @@ -1099,6 +1114,10 @@ reused in custom filters:
1099           1114       Note that this filter does not block *all* dangerous archive features.
1100           1115       See :ref:`tarfile-further-verification`  for details.
1101           1116
                1117 +   .. versionchanged:: next
                1118 +
                1119 +      Link targets are now normalized.
                1120 +
1102           1121
1103           1122     .. _tarfile-extraction-refuse:
1104           1123
                        @@ -1127,6 +1146,7 @@ Here is an incomplete list of things to consider:
1127           1146     * Extract to a :func:`new temporary directory <tempfile.mkdtemp>`
1128           1147       to prevent e.g. exploiting pre-existing links, and to make it easier to
1129           1148       clean up after a failed extraction.
                1149 + * Disallow symbolic links if you do not need the functionality.
1130           1150     * When working with untrusted data, use external (e.g. OS-level) limits on
1131           1151       disk, memory and CPU usage.
1132           1152     * Check filenames against an allow-list of characters
```

Doc/whatsnew/3.15.rst

```
@@ -116,6 +116,16 @@ math
  (Contributed by Sergey B Kirpichev in :gh:`132908`.)


+os.path
+-------
+
+* The *strict* parameter to :func:`os.path.realpath` accepts a new value,
+  :data:`os.path.ALLOW_MISSING`.
+  If used, errors other than :exc:`FileNotFoundError` will be re-raised;
+  the resulting path can be missing but it will be free of symlinks.
+  (Contributed by Petr Viktorin for :cve:`2025-4517`.)
+
+
 shelve
 ------

@@ -132,6 +142,28 @@ ssl
  (Contributed by Will Childs-Klein in :gh:`133624`.)


+tarfile
+-------
+
+* :func:`~tarfile.data_filter` now normalizes symbolic link targets in order to
+  avoid path traversal attacks.
+  (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2025-4138`.)
+* :func:`~tarfile.TarFile.extractall` now skips fixing up directory attributes
+  when a directory was removed or replaced by another kind of file.
+  (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2024-12718`.)
+* :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
+  now (re-)apply the extraction filter when substituting a link (hard or
+  symbolic) with a copy of another archive member, and when fixing up
+  directory attributes.
+  The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
+  (Contributed by Petr Viktorin for :cve:`2025-4330` and :cve:`2024-12718`.)
+* :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
```

```
      161 +   no longer extract rejected members when
      162 +   :func:`~tarfile.TarFile.errorlevel` is zero.
      163 +   (Contributed by Matt Prodani and Petr Viktorin in :gh:`112887`
      164 +   and :cve:`2025-4435`.)
      165 +
      166 +
135   167     zlib
136   168     ----
137   169
```

Lib/genericpath.py

```
@@ -8,7 +8,7 @@
8     8
9     9     __all__ = ['commonprefix', 'exists', 'getatime', 'getctime', 'getmtime',
10    10              'getsize', 'isdevdrive', 'isdir', 'isfile', 'isjunction', 'islink',
11          -           'lexists', 'samefile', 'sameopenfile', 'samestat']
      11    +           'lexists', 'samefile', 'sameopenfile', 'samestat', 'ALLOW_MISSING']
12    12
13    13
14    14    # Does a path exist?
@@ -189,3 +189,12 @@ def _check_arg_types(funcname, *args):
189   189                               f'os.PathLike object, not
      {s.__class__.__name__!r}') from None
190   190         if hasstr and hasbytes:
191   191             raise TypeError("Can't mix strings and bytes in path components") from
      None
      192   +
      193   + # A singleton with a true boolean value.
      194   + @object.__new__
      195   + class ALLOW_MISSING:
      196   +     """Special value for use in realpath()."""
      197   +     def __repr__(self):
      198   +         return 'os.path.ALLOW_MISSING'
      199   +     def __reduce__(self):
      200   +         return self.__class__.__name__
```

Lib/ntpath.py

```
@@ -29,7 +29,7 @@
29  29              "abspath","curdir","pardir","sep","pathsep","defpath","altsep",
30  30
                "extsep","devnull","realpath","supports_unicode_filenames","relpath",
31  31              "samefile", "sameopenfile", "samestat", "commonpath", "isjunction",
32      -           "isdevdrive"]
    32  +           "isdevdrive", "ALLOW_MISSING"]
33  33
34  34  def _get_bothseps(path):
35  35      if isinstance(path, bytes):
@@ -601,9 +601,10 @@ def abspath(path):
601 601      from nt import _findfirstfile, _getfinalpathname, readlink as _nt_readlink
602 602  except ImportError:
603 603      # realpath is a no-op on systems without _getfinalpathname support.
604     -    realpath = abspath
    604 +    def realpath(path, *, strict=False):
    605 +        return abspath(path)
605 606  else:
606     -    def _readlink_deep(path):
    607 +    def _readlink_deep(path, ignored_error=OSError):
607 608          # These error codes indicate that we should stop reading links and
608 609          # return the path we currently have.
609 610          # 1: ERROR_INVALID_FUNCTION
@@ -636,7 +637,7 @@ def _readlink_deep(path):
636 637                          path = old_path
637 638                          break
638 639                      path = normpath(join(dirname(old_path), path))
639     -            except OSError as ex:
    640 +            except ignored_error as ex:
640 641                  if ex.winerror in allowed_winerror:
641 642                      break
642 643                  raise
@@ -645,7 +646,7 @@ def _readlink_deep(path):
645 646                  break
646 647          return path
647 648
648     -    def _getfinalpathname_nonstrict(path):
    649 +    def _getfinalpathname_nonstrict(path, ignored_error=OSError):
```

```
649  650            # These error codes indicate that we should stop resolving the path
650  651            # and return the value we currently have.
651  652            # 1: ERROR_INVALID_FUNCTION
     @@ -673,25 +674,26 @@ def _getfinalpathname_nonstrict(path):
673  674                try:
674  675                    path = _getfinalpathname(path)
675  676                    return join(path, tail) if tail else path
676       -            except OSError as ex:
     677  +            except ignored_error as ex:
677  678                    if ex.winerror not in allowed_winerror:
678  679                        raise
679  680                    try:
680  681                        # The OS could not resolve this path fully, so we attempt
681  682                        # to follow the link ourselves. If we succeed, join the tail
682  683                        # and return.
683       -                    new_path = _readlink_deep(path)
     684  +                    new_path = _readlink_deep(path,
     685  +                                              ignored_error=ignored_error)
684  686                    if new_path != path:
685  687                        return join(new_path, tail) if tail else new_path
686       -                except OSError:
     688  +                except ignored_error:
687  689                    # If we fail to readlink(), let's keep traversing
688  690                    pass
689  691                # If we get these errors, try to get the real name of the file without accessing it.
690  692                if ex.winerror in (1, 5, 32, 50, 87, 1920, 1921):
691  693                    try:
692  694                        name = _findfirstfile(path)
693  695                        path, _ = split(path)
694       -                except OSError:
     696  +                except ignored_error:
695  697                        path, name = split(path)
696  698                else:
697  699                    path, name = split(path)
     @@ -721,24 +723,32 @@ def realpath(path, *, strict=False):
721  723                if normcase(path) == devnull:
```

```
722  724                    return '\\\\.\\NUL'
723  725            had_prefix = path.startswith(prefix)
     726  +
     727  +         if strict is ALLOW_MISSING:
     728  +             ignored_error = FileNotFoundError
     729  +             strict = True
     730  +         elif strict:
     731  +             ignored_error = ()
     732  +         else:
     733  +             ignored_error = OSError
     734  +
724  735            if not had_prefix and not isabs(path):
725  736                path = join(cwd, path)
726  737            try:
727  738                path = _getfinalpathname(path)
728  739                initial_winerror = 0
729  740            except ValueError as ex:
730  741                # gh-106242: Raised for embedded null characters
731       -            # In strict mode, we convert into an OSError.
     742  +            # In strict modes, we convert into an OSError.
732  743                # Non-strict mode returns the path as-is, since we've already
733  744                # made it absolute.
734  745                if strict:
735  746                    raise OSError(str(ex)) from None
736  747                path = normpath(path)
737       -        except OSError as ex:
738       -            if strict:
739       -                raise
     748  +        except ignored_error as ex:
740  749                initial_winerror = ex.winerror
741       -            path = _getfinalpathname_nonstrict(path)
     750  +            path = _getfinalpathname_nonstrict(path,
     751  +                                               ignored_error=ignored_error)
742  752            # The path returned by _getfinalpathname will always start with \\?\ -
743  753            # strip off that prefix unless it was already provided on the original
744  754            # path.
```

Lib/posixpath.py

```
@@ -36,7 +36,7 @@
```

```
36  36              "samefile","sameopenfile","samestat",
37  37              "curdir","pardir","sep","pathsep","defpath","altsep","extsep",
38  38              "devnull","realpath","supports_unicode_filenames","relpath",
39      -           "commonpath", "isjunction","isdevdrive"]
    39  +           "commonpath", "isjunction","isdevdrive","ALLOW_MISSING"]
40  40
41  41
42  42  def _get_sep(path):
        @@ -402,10 +402,18 @@ def realpath(filename, *, strict=False):
402 402         curdir = '.'
403 403         pardir = '..'
404 404         getcwd = os.getcwd
405     -     return _realpath(filename, strict, sep, curdir, pardir, getcwd)
    405 +     if strict is ALLOW_MISSING:
    406 +         ignored_error = FileNotFoundError
    407 +         strict = True
    408 +     elif strict:
    409 +         ignored_error = ()
    410 +     else:
    411 +         ignored_error = OSError
    412 +
    413 +     lstat = os.lstat
    414 +     readlink = os.readlink
    415 +     maxlinks = None
406 416
407     - def _realpath(filename, strict=False, sep=sep, curdir=curdir, pardir=pardir,
408     -               getcwd=os.getcwd, lstat=os.lstat, readlink=os.readlink,
        maxlinks=None):
409 417     # The stack of unresolved path parts. When popped, a special value of None
410 418     # indicates that a symlink target has been resolved, and that the original
411 419     # symlink path can be retrieved by popping again. The [::-1] slice is a
        @@ -477,27 +485,28 @@ def _realpath(filename, strict=False, sep=sep,
        curdir=curdir, pardir=pardir,
477 485                 path = newpath
478 486                 continue
479 487             target = readlink(newpath)
480     -         except OSError:
481     -             if strict:
482     -                 raise
```

```
483     -             path = newpath
        488 +         except ignored_error:
        489 +             pass
        490 +         else:
        491 +             # Resolve the symbolic link
        492 +             if target.startswith(sep):
        493 +                 # Symlink target is absolute; reset resolved path.
        494 +                 path = sep
        495 +             if maxlinks is None:
        496 +                 # Mark this symlink as seen but not fully resolved.
        497 +                 seen[newpath] = None
        498 +                 # Push the symlink path onto the stack, and signal its
              specialness
        499 +                 # by also pushing None. When these entries are popped, we'll
        500 +                 # record the fully-resolved symlink target in the 'seen'
              mapping.
        501 +                 rest.append(newpath)
        502 +                 rest.append(None)
        503 +             # Push the unresolved symlink target parts onto the stack.
        504 +             target_parts = target.split(sep)[::-1]
        505 +             rest.extend(target_parts)
        506 +             part_count += len(target_parts)
484     507               continue
485     -         # Resolve the symbolic link
486     -         if target.startswith(sep):
487     -             # Symlink target is absolute; reset resolved path.
488     -             path = sep
489     -         if maxlinks is None:
490     -             # Mark this symlink as seen but not fully resolved.
491     -             seen[newpath] = None
492     -             # Push the symlink path onto the stack, and signal its specialness
493     -             # by also pushing None. When these entries are popped, we'll
494     -             # record the fully-resolved symlink target in the 'seen' mapping.
495     -             rest.append(newpath)
496     -             rest.append(None)
497     -         # Push the unresolved symlink target parts onto the stack.
498     -         target_parts = target.split(sep)[::-1]
499     -         rest.extend(target_parts)
500     -         part_count += len(target_parts)
        508 +         # An error occurred and was ignored.
```

```
                509 +          path = newpath
501             510
502             511          return path
503             512
```

Lib/tarfile.py

```
                    @@ -67,7 +67,7 @@
67              67              "DEFAULT_FORMAT", "open","fully_trusted_filter", "data_filter",
68              68              "tar_filter", "FilterError", "AbsoluteLinkError",
69              69              "OutsideDestinationError", "SpecialFileError",
                "AbsolutePathError",
70                  -           "LinkOutsideDestinationError"]
                70  +           "LinkOutsideDestinationError", "LinkFallbackError"]
71              71
72              72
73              73      #--------------------------------------------------------
                    @@ -766,10 +766,22 @@ def __init__(self, tarinfo, path):
766             766             super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
767             767                              + 'which is outside the destination')
768             768
                769 + class LinkFallbackError(FilterError):
                770 +     def __init__(self, tarinfo, path):
                771 +         self.tarinfo = tarinfo
                772 +         self._path = path
                773 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
                774 +                          + f'copy of {path!r}, which was rejected')
                775 +
                776 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
                777 + # we consider to be issues with the argument, rather than a bug in the
                778 + # filter function
                779 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
                780 +
769             781     def _get_filtered_attrs(member, dest_path, for_data=True):
770             782         new_attrs = {}
771             783         name = member.name
772                 -       dest_path = os.path.realpath(dest_path)
                784 +       dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
773             785         # Strip leading / (tar's directory separator) from filenames.
```

```
774  786          # Include os.sep (target OS directory separator) as well.
775  787          if name.startswith(('/', os.sep)):
     @@ -779,7 +791,8 @@ def _get_filtered_attrs(member, dest_path,
     for_data=True):
779  791              # For example, 'C:/foo' on Windows.
780  792              raise AbsolutePathError(member)
781  793          # Ensure we stay in the destination
782       -      target_path = os.path.realpath(os.path.join(dest_path, name))
          794 +      target_path = os.path.realpath(os.path.join(dest_path, name),
          795 +                                     strict=os.path.ALLOW_MISSING)
783  796          if os.path.commonpath([target_path, dest_path]) != dest_path:
784  797              raise OutsideDestinationError(member, target_path)
785  798          # Limit permissions (no high bits, and go-w)
     @@ -817,14 +830,18 @@ def _get_filtered_attrs(member, dest_path,
     for_data=True):
817  830              if member.islnk() or member.issym():
818  831                  if os.path.isabs(member.linkname):
819  832                      raise AbsoluteLinkError(member)
          833 +                normalized = os.path.normpath(member.linkname)
          834 +                if normalized != member.linkname:
          835 +                    new_attrs['linkname'] = normalized
820  836                  if member.issym():
821  837                      target_path = os.path.join(dest_path,
822  838                                                 os.path.dirname(name),
823  839                                                 member.linkname)
824  840                  else:
825  841                      target_path = os.path.join(dest_path,
826  842                                                 member.linkname)
827       -              target_path = os.path.realpath(target_path)
          843 +              target_path = os.path.realpath(target_path,
          844 +                                             strict=os.path.ALLOW_MISSING)
828  845                  if os.path.commonpath([target_path, dest_path]) != dest_path:
829  846                      raise LinkOutsideDestinationError(member, target_path)
830  847          return new_attrs
     @@ -2386,30 +2403,58 @@ def extractall(self, path=".", members=None, *,
     numeric_owner=False,
2386 2403              members = self
2387 2404
2388 2405          for member in members:
```

```diff
-            tarinfo = self._get_extract_tarinfo(member, filter_function, path)
+            tarinfo, unfiltered = self._get_extract_tarinfo(
+                member, filter_function, path)
             if tarinfo is None:
                 continue
             if tarinfo.isdir():
                 # For directories, delay setting attributes until later,
                 # since permissions can interfere with extraction and
                 # extracting contents can reset mtime.
-                directories.append(tarinfo)
+                directories.append(unfiltered)
             self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
-                              numeric_owner=numeric_owner)
+                              numeric_owner=numeric_owner,
+                              filter_function=filter_function)
 
         # Reverse sort directories.
         directories.sort(key=lambda a: a.name, reverse=True)
 
+
         # Set correct owner, mtime and filemode on directories.
-        for tarinfo in directories:
-            dirpath = os.path.join(path, tarinfo.name)
+        for unfiltered in directories:
             try:
+                # Need to re-apply any filter, to take the *current* filesystem
+                # state into account.
+                try:
+                    tarinfo = filter_function(unfiltered, path)
+                except _FILTER_ERRORS as exc:
+                    self._log_no_directory_fixup(unfiltered, repr(exc))
+                    continue
+                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,
```

```
        @@ -2425,41 +2470,56 @@ 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)

    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*
```

```
        2481 +          *member* might be a string.
        2482 +
        2483 +          Return (None, None) if not found.
        2484 +          """
        2485 +
2434    2486            if isinstance(member, str):
2435         -              tarinfo = self.getmember(member)
        2487 +              unfiltered = self.getmember(member)
2436    2488            else:
2437         -              tarinfo = member
        2489 +              unfiltered = member
2438    2490
2439         -          unfiltered = tarinfo
        2491 +          filtered = None
2440    2492            try:
2441         -              tarinfo = filter_function(tarinfo, path)
        2493 +              filtered = filter_function(unfiltered, path)
2442    2494            except (OSError, UnicodeEncodeError, FilterError) as e:
2443    2495                self._handle_fatal_error(e)
2444    2496            except ExtractError as e:
2445    2497                self._handle_nonfatal_error(e)
2446         -          if tarinfo is None:
        2498 +          if filtered is None:
2447    2499                self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2448         -              return None
        2500 +              return None, None
        2501 +
2449    2502            # Prepare the link target for makelink().
2450         -          if tarinfo.islnk():
2451         -              tarinfo = copy.copy(tarinfo)
2452         -              tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2453         -          return tarinfo
        2503 +          if filtered.islnk():
        2504 +              filtered = copy.copy(filtered)
        2505 +              filtered._link_target = os.path.join(path, filtered.linkname)
        2506 +          return filtered, unfiltered
        2507 +
        2508 +      def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
        2509 +                       filter_function=None):
        2510 +          """Extract from filtered tarinfo to disk.
```

```diff
 
-    def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
-        """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, UnicodeEncodeError) as e:
             self._handle_fatal_error(e)
         except ExtractError as e:
@@ -2517,9 +2577,13 @@ 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
@@ -2548,7 +2612,10 @@ 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():
```

```
2551      -             self.makelink(tarinfo, targetpath)
      2615 +             self.makelink_with_filter(
      2616 +                 tarinfo, targetpath,
      2617 +                 filter_function=filter_function,
      2618 +                 extraction_root=extraction_root)
2552  2619           elif tarinfo.type not in SUPPORTED_TYPES:
2553  2620               self.makeunknown(tarinfo, targetpath)
2554  2621           else:
          @@ -2631,29 +2698,57 @@ def makedev(self, tarinfo, targetpath):
2631  2698                   os.makedev(tarinfo.devmajor, tarinfo.devminor))
2632  2699
2633  2700       def makelink(self, tarinfo, targetpath):
      2701 +         return self.makelink_with_filter(tarinfo, targetpath, None, None)
      2702 +
      2703 +     def makelink_with_filter(self, tarinfo, targetpath,
      2704 +                              filter_function, extraction_root):
2634  2705           """Make a (symbolic) link called targetpath. If it cannot be created
2635  2706             (platform limitation), we try to make a copy of the referenced file
2636  2707             instead of a link.
      2708 +
      2709 +           filter_function is only used when extracting a *different*
      2710 +           member (e.g. as fallback to creating a link).
2637  2711           """
      2712 +         keyerror_to_extracterror = False
2638  2713           try:
2639  2714               # For systems that support symbolic and hard links.
2640  2715               if tarinfo.issym():
2641  2716                   if os.path.lexists(targetpath):
2642  2717                       # Avoid FileExistsError on following os.symlink.
2643  2718                       os.unlink(targetpath)
2644  2719                   os.symlink(tarinfo.linkname, targetpath)
      2720 +                 return
2645  2721               else:
2646  2722                   if os.path.exists(tarinfo._link_target):
2647  2723                       os.link(tarinfo._link_target, targetpath)
2648      -                 else:
2649      -                     self._extract_member(self._find_link_target(tarinfo),
2650      -                                          targetpath)
      2724 +                     return
```

```
2651  2725            except symlink_exception:
      2726  +             keyerror_to_extracterror = True
      2727  +
      2728  +         try:
      2729  +             unfiltered = self._find_link_target(tarinfo)
      2730  +         except KeyError:
      2731  +             if keyerror_to_extracterror:
      2732  +                 raise ExtractError(
      2733  +                     "unable to resolve link inside archive") from None
      2734  +             else:
      2735  +                 raise
      2736  +
      2737  +         if filter_function is None:
      2738  +             filtered = unfiltered
      2739  +         else:
      2740  +             if extraction_root is None:
      2741  +                 raise ExtractError(
      2742  +                     "makelink_with_filter: if filter_function is not None, "
      2743  +                     + "extraction_root must also not be None")
2652  2744              try:
2653        -                 self._extract_member(self._find_link_target(tarinfo),
2654        -                                      targetpath)
2655        -             except KeyError:
2656        -                 raise ExtractError("unable to resolve link inside archive")
                            from None
      2745  +                 filtered = filter_function(unfiltered, extraction_root)
      2746  +             except _FILTER_ERRORS as cause:
      2747  +                 raise LinkFallbackError(tarinfo, unfiltered.name) from cause
      2748  +         if filtered is not None:
      2749  +             self._extract_member(filtered, targetpath,
      2750  +                                  filter_function=filter_function,
      2751  +                                  extraction_root=extraction_root)
2657  2752
2658  2753      def chown(self, tarinfo, targetpath, numeric_owner):
2659  2754          """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