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

```
@@ -377,10 +377,26 @@ the :mod:`glob` module.)
377 377     links encountered in the path (if they are supported by the operating
378 378     system).
379 379
380     -   If a path doesn't exist or a symlink loop is encountered, and *strict* is
381     -   ``True``, :exc:`OSError` is raised. If *strict* is ``False``, the path is
382     -   resolved as far as possible and any remainder is appended without checking
383     -   whether it exists.
        380 +   By default, the path is evaluated up to the first component that does not
        381 +   exist, is a symlink loop, or whose evaluation raises :exc:`OSError`.
        382 +   All such components are appended unchanged to the existing part of the path.
        383 +
        384 +   Some errors that are handled this way include "access denied", "not a
        385 +   directory", or "bad argument to internal function". Thus, the
        386 +   resulting path may be missing or inaccessible, may still contain
        387 +   links or loops, and may traverse non-directories.
        388 +
        389 +   This behavior can be modified by keyword arguments:
        390 +
        391 +   If *strict* is ``True``, the first error encountered when evaluating the
            path is
        392 +   re-raised.
        393 +   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.
384 400
385 401     .. note::
386 402        This function emulates the operating system's procedure for making a path
    @@ -399,6 +415,15 @@ the :mod:`glob` module.)
399 415     .. versionchanged:: 3.10
400 416        The *strict* parameter was added.
401 417
+    .. 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
402 427
403 428  .. function:: relpath(path, start=os.curdir)
404 429
```

Doc/library/tarfile.rst

```
    @@ -249,6 +249,15 @@ The :mod:`tarfile` module defines the following exceptions:
249 249        Raised to refuse extracting a symbolic link pointing outside the destination
250 250        directory.
251 251
+ .. 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
```

```diff
+    as :attr:`!BaseException.__context__`.
+
+    .. versionadded:: next
+

   The following constants are available at the module level:


@@ -1039,6 +1048,12 @@ reused in custom filters:
     Implements the ``'data'`` filter.
     In addition to what ``tar_filter`` does:

+    - 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.
+
     - :ref:`Refuse <tarfile-extraction-refuse>` to extract links (hard or soft)
       that link to absolute paths, or ones that link outside the destination.

@@ -1067,6 +1082,10 @@ reused in custom filters:

     Return the modified ``TarInfo`` member.

+    .. versionchanged:: next
+
+       Link targets are now normalized.
+

   .. _tarfile-extraction-refuse:

@@ -1093,6 +1112,7 @@ Here is an incomplete list of things to consider:
   * Extract to a :func:`new temporary directory <tempfile.mkdtemp>`
     to prevent e.g. exploiting pre-existing links, and to make it easier to
     clean up after a failed extraction.
+ * Disallow symbolic links if you do not need the functionality.
   * When working with untrusted data, use external (e.g. OS-level) limits on
     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`
```

```
2353 +   no longer extract rejected members when
2354 +   :func:`~tarfile.TarFile.errorlevel` is zero.
2355 +   (Contributed by Matt Prodani and Petr Viktorin in :gh:`112887`
2356 +   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', 'islink', 'samefile', 'sameopenfile',
11    -            'samestat']
   11 +            'samestat', 'ALLOW_MISSING']
12 12
13 13
14 14  # Does a path exist?
@@ -165,3 +165,12 @@ def _check_arg_types(funcname, *args):
165 165                                f'os.PathLike object, not
    {s.__class__.__name__!r}') from None
166 166      if hasstr and hasbytes:
167 167          raise TypeError("Can't mix strings and bytes in path components") from
    None
   168 +
   169 + # A singleton with a true boolean value.
   170 + @object.__new__
   171 + class ALLOW_MISSING:
   172 +     """Special value for use in realpath()."""
   173 +     def __repr__(self):
   174 +         return 'os.path.ALLOW_MISSING'
   175 +     def __reduce__(self):
   176 +         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", "isjunction"]
```

```
                +            "samefile", "sameopenfile", "samestat", "commonpath", "isjunction",
                +            "ALLOW_MISSING"]
34    35
35    36    def _get_bothseps(path):
36    37        if isinstance(path, bytes):
          @@ -609,9 +610,10 @@ def abspath(path):
609   610        from nt import _getfinalpathname, readlink as _nt_readlink
610   611    except ImportError:
611   612        # realpath is a no-op on systems without _getfinalpathname support.
612         -     realpath = abspath
              +     def realpath(path, *, strict=False):
              +         return abspath(path)
613   615    else:
614         -     def _readlink_deep(path):
              +     def _readlink_deep(path, ignored_error=OSError):
615   617            # These error codes indicate that we should stop reading links and
616   618            # return the path we currently have.
617   619            # 1: ERROR_INVALID_FUNCTION
          @@ -644,7 +646,7 @@ def _readlink_deep(path):
644   646                            path = old_path
645   647                            break
646   648                        path = normpath(join(dirname(old_path), path))
647         -             except OSError as ex:
              +             except ignored_error as ex:
648   650                     if ex.winerror in allowed_winerror:
649   651                         break
650   652                     raise
          @@ -653,7 +655,7 @@ def _readlink_deep(path):
653   655                     break
654   656            return path
655   657
656         - def _getfinalpathname_nonstrict(path):
              + def _getfinalpathname_nonstrict(path, ignored_error=OSError):
657   659        # These error codes indicate that we should stop resolving the path
658   660        # and return the value we currently have.
659   661        # 1: ERROR_INVALID_FUNCTION
          @@ -680,17 +682,18 @@ def _getfinalpathname_nonstrict(path):
680   682                try:
```

```
681  683                      path = _getfinalpathname(path)
682  684                      return join(path, tail) if tail else path
683       -              except OSError as ex:
     685  +              except ignored_error as ex:
684  686                      if ex.winerror not in allowed_winerror:
685  687                          raise
686  688                      try:
687  689                          # The OS could not resolve this path fully, so we attempt
688  690                          # to follow the link ourselves. If we succeed, join the
                                tail
689  691                          # and return.
690       -                      new_path = _readlink_deep(path)
     692  +                      new_path = _readlink_deep(path,
     693  +                                                ignored_error=ignored_error)
691  694                      if new_path != path:
692  695                          return join(new_path, tail) if tail else new_path
693       -                  except OSError:
     696  +                  except ignored_error:
694  697                          # If we fail to readlink(), let's keep traversing
695  698                          pass
696  699                      path, name = split(path)
                          @@ -721,24 +724,32 @@ def realpath(path, *, strict=False):
721  724                  if normcase(path) == normcase(devnull):
722  725                      return '\\\\.\\NUL'
723  726              had_prefix = path.startswith(prefix)
     727  +
     728  +        if strict is ALLOW_MISSING:
     729  +            ignored_error = FileNotFoundError
     730  +            strict = True
     731  +        elif strict:
     732  +            ignored_error = ()
     733  +        else:
     734  +            ignored_error = OSError
     735  +
724  736              if not had_prefix and not isabs(path):
725  737                  path = join(cwd, path)
726  738              try:
727  739                  path = _getfinalpathname(path)
728  740                  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
```

```
       444 +     elif strict:
       445 +         ignored_error = ()
       446 +     else:
       447 +         ignored_error = OSError
       448 +
       449 +     maxlinks = None
441    450
442    451         if isabs(rest):
443    452             rest = rest[1:]
          @@ -460,9 +469,7 @@ def _joinrealpath(path, rest, strict, seen):
460    469             newpath = join(path, name)
461    470             try:
462    471                 st = os.lstat(newpath)
463       -         except OSError:
464       -             if strict:
465       -                 raise
       472 +         except ignored_error:
466    473                 is_link = False
467    474             else:
468    475                 is_link = stat.S_ISLNK(st.st_mode)
```

Lib/tarfile.py

```
          @@ -752,10 +752,22 @@ def __init__(self, tarinfo, path):
752    752             super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
753    753                              + 'which is outside the destination')
754    754
       755 + class LinkFallbackError(FilterError):
       756 +     def __init__(self, tarinfo, path):
       757 +         self.tarinfo = tarinfo
       758 +         self._path = path
       759 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
       760 +                          + f'copy of {path!r}, which was rejected')
       761 +
       762 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
       763 + # we consider to be issues with the argument, rather than a bug in the
       764 + # filter function
       765 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
       766 +
755    767     def _get_filtered_attrs(member, dest_path, for_data=True):
```

```
756  768          new_attrs = {}
757  769          name = member.name
758       -      dest_path = os.path.realpath(dest_path)
     770  +      dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
759  771          # Strip leading / (tar's directory separator) from filenames.
760  772          # Include os.sep (target OS directory separator) as well.
761  773          if name.startswith(('/', os.sep)):
          @@ -765,7 +777,8 @@ def _get_filtered_attrs(member, dest_path,
          for_data=True):
765  777              # For example, 'C:/foo' on Windows.
766  778              raise AbsolutePathError(member)
767  779          # Ensure we stay in the destination
768       -      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)
769  782          if os.path.commonpath([target_path, dest_path]) != dest_path:
770  783              raise OutsideDestinationError(member, target_path)
771  784          # Limit permissions (no high bits, and go-w)
          @@ -803,14 +816,18 @@ def _get_filtered_attrs(member, dest_path,
          for_data=True):
803  816              if member.islnk() or member.issym():
804  817                  if os.path.isabs(member.linkname):
805  818                      raise AbsoluteLinkError(member)
     819  +              normalized = os.path.normpath(member.linkname)
     820  +              if normalized != member.linkname:
     821  +                  new_attrs['linkname'] = normalized
806  822                  if member.issym():
807  823                      target_path = os.path.join(dest_path,
808  824                                                 os.path.dirname(name),
809  825                                                 member.linkname)
810  826                  else:
811  827                      target_path = os.path.join(dest_path,
812  828                                                 member.linkname)
813       -              target_path = os.path.realpath(target_path)
     829  +              target_path = os.path.realpath(target_path,
     830  +                                             strict=os.path.ALLOW_MISSING)
814  831                  if os.path.commonpath([target_path, dest_path]) != dest_path:
815  832                      raise LinkOutsideDestinationError(member, target_path)
816  833          return new_attrs
```

```
@@ -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)

```

```
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)
```

```
                2411 +          return filtered, unfiltered
2359            2412
2360            2413 -      def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2361            2414 -          """Extract from filtered tarinfo to disk"""
                2413 +      def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
                2414 +                       filter_function=None):
                2415 +          """Extract from filtered tarinfo to disk.
                2416 +
                2417 +             filter_function is only used when extracting a *different*
                2418 +             member (e.g. as fallback to creating a symlink)
                2419 +          """
2362            2420              self._check("r")
2363            2421
2364            2422              try:
2365            2423                  self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2366            2424                                       set_attrs=set_attrs,
2367            2425 -                                     numeric_owner=numeric_owner)
                2425 +                                     numeric_owner=numeric_owner,
                2426 +                                     filter_function=filter_function,
                2427 +                                     extraction_root=path)
2368            2428              except OSError as e:
2369            2429                  self._handle_fatal_error(e)
2370            2430              except ExtractError as e:
                   @@ -2422,9 +2482,13 @@ def extractfile(self, member):
2422            2482                  return None
2423            2483
2424            2484          def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2425            2485 -                            numeric_owner=False):
2426            2486 -          """Extract the TarInfo object tarinfo to a physical
                2485 +                            numeric_owner=False, *, filter_function=None,
                2486 +                            extraction_root=None):
                2487 +          """Extract the filtered TarInfo object tarinfo to a physical
2427            2488              file called targetpath.
                2489 +
                2490 +             filter_function is only used when extracting a *different*
                2491 +             member (e.g. as fallback to creating a symlink)
2428            2492              """
2429            2493              # Fetch the TarInfo object for the given name
2430            2494              # and build the destination pathname, replacing
```

```
@@ -2453,7 +2517,10 @@ def _extract_member(self, tarinfo, targetpath,
set_attrs=True,
2453 2517             elif tarinfo.ischr() or tarinfo.isblk():
2454 2518                 self.makedev(tarinfo, targetpath)
2455 2519             elif tarinfo.islnk() or tarinfo.issym():
2456      -             self.makelink(tarinfo, targetpath)
     2520 +             self.makelink_with_filter(
     2521 +                 tarinfo, targetpath,
     2522 +                 filter_function=filter_function,
     2523 +                 extraction_root=extraction_root)
2457 2524             elif tarinfo.type not in SUPPORTED_TYPES:
2458 2525                 self.makeunknown(tarinfo, targetpath)
2459 2526             else:
@@ -2536,29 +2603,57 @@ def makedev(self, tarinfo, targetpath):
2536 2603                      os.makedev(tarinfo.devmajor, tarinfo.devminor))
2537 2604
2538 2605         def makelink(self, tarinfo, targetpath):
     2606 +         return self.makelink_with_filter(tarinfo, targetpath, None, None)
     2607 +
     2608 +     def makelink_with_filter(self, tarinfo, targetpath,
     2609 +                              filter_function, extraction_root):
2539 2610             """Make a (symbolic) link called targetpath. If it cannot be created
2540 2611               (platform limitation), we try to make a copy of the referenced file
2541 2612               instead of a link.
     2613 +
     2614 +           filter_function is only used when extracting a *different*
     2615 +           member (e.g. as fallback to creating a link).
2542 2616             """
     2617 +         keyerror_to_extracterror = False
2543 2618             try:
2544 2619                 # For systems that support symbolic and hard links.
2545 2620                 if tarinfo.issym():
2546 2621                     if os.path.lexists(targetpath):
2547 2622                         # Avoid FileExistsError on following os.symlink.
2548 2623                         os.unlink(targetpath)
2549 2624                     os.symlink(tarinfo.linkname, targetpath)
     2625 +                   return
2550 2626                 else:
2551 2627                     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