python / cpython Public

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

Commit 4633f3f

6 people authored on Jun 3, 2025 · 57 / 58 · Partially verified

```
[3.11] gh-135034: Normalize link targets in tarfile, add
os.path.realpath(strict='allow_missing') (GH-135037) (GH-135068)

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

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>
```

3.11 (#98846, #135068) · v3.11.15 ••• v3.11.13

1 parent 2c6ca1a commit 4633f3f

11 files changed +1,017 -138

Top

Filter files…

- Doc
  - library
    - os.path.rst
    - tarfile.rst
  - whatsnew
    - 3.11.rst
- Lib
  - genericpath.py

- ntpath.py
- posixpath.py
- tarfile.py
- test
  - test_ntpath.py
  - test_posixpath.py
  - test_tarfile.py
- Misc/NEWS.d/next/Security
  - 2025-06-02-11-32-23.gh-issue-135034.RLGjbp.rst

Search within code

Doc/library/os.path.rst

```
@@ -352,10 +352,26 @@ the :mod:`glob` module.)
352 352     links encountered in the path (if they are supported by the operating
353 353     system).
354 354
355     -   If a path doesn't exist or a symlink loop is encountered, and *strict* is
356     -   ``True``, :exc:`OSError` is raised. If *strict* is ``False``, the path is
357     -   resolved as far as possible and any remainder is appended without checking
358     -   whether it exists.
    355 +   By default, the path is evaluated up to the first component that does not
    356 +   exist, is a symlink loop, or whose evaluation raises :exc:`OSError`.
    357 +   All such components are appended unchanged to the existing part of the path.
    358 +
    359 +   Some errors that are handled this way include "access denied", "not a
    360 +   directory", or "bad argument to internal function". Thus, the
    361 +   resulting path may be missing or inaccessible, may still contain
    362 +   links or loops, and may traverse non-directories.
    363 +
    364 +   This behavior can be modified by keyword arguments:
    365 +
    366 +   If *strict* is ``True``, the first error encountered when evaluating the
      path is
    367 +   re-raised.
    368 +   In particular, :exc:`FileNotFoundError` is raised if *path* does not exist,
```

```
      369  +    or another :exc:`OSError` if it is otherwise inaccessible.
      370  +
      371  +    If *strict* is :py:data:`os.path.ALLOW_MISSING`, errors other than
      372  +    :exc:`FileNotFoundError` are re-raised (as with ``strict=True``).
      373  +    Thus, the returned path will not contain any symbolic links, but the named
      374  +    file and some of its parent directories may be missing.
359   375
360   376        .. note::
361   377           This function emulates the operating system's procedure for making a path
          @@ -374,6 +390,15 @@ the :mod:`glob` module.)
374   390        .. versionchanged:: 3.10
375   391           The *strict* parameter was added.
376   392
      393  +    .. versionchanged:: next
      394  +       The :py:data:`~os.path.ALLOW_MISSING` value for the *strict* parameter
      395  +       was added.
      396  +
      397  + .. data:: ALLOW_MISSING
      398  +
      399  +    Special value used for the *strict* argument in :func:`realpath`.
      400  +
      401  +    .. versionadded:: next
377   402
378   403     .. function:: relpath(path, start=os.curdir)
379   404
```

Doc/library/tarfile.rst

```
          @@ -239,6 +239,15 @@ The :mod:`tarfile` module defines the following exceptions:
239   239        Raised to refuse extracting a symbolic link pointing outside the destination
240   240        directory.
241   241
      242  + .. exception:: LinkFallbackError
      243  +
      244  +    Raised to refuse emulating a link (hard or symbolic) by extracting another
      245  +    archive member, when that member would be rejected by the filter location.
      246  +    The exception that was raised to reject the replacement member is available
```

```
                 +    as :attr:`!BaseException.__context__`.
                 +
                 +    .. versionadded:: next
                 +
242  251
243  252     The following constants are available at the module level:
244  253
                    @@ -1037,6 +1046,12 @@ reused in custom filters:
1037 1046       Implements the ``'data'`` filter.
1038 1047       In addition to what ``tar_filter`` does:
1039 1048
                 +   - 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.
                 +
1040 1055       - :ref:`Refuse <tarfile-extraction-refuse>` to extract links (hard or soft)
1041 1056         that link to absolute paths, or ones that link outside the destination.
1042 1057
                    @@ -1065,6 +1080,10 @@ reused in custom filters:
1065 1080
1066 1081       Return the modified ``TarInfo`` member.
1067 1082
                 +   .. versionchanged:: next
                 +
                 +      Link targets are now normalized.
                 +
1068 1087
1069 1088     .. _tarfile-extraction-refuse:
1070 1089
                    @@ -1091,6 +1110,7 @@ Here is an incomplete list of things to consider:
1091 1110     * Extract to a :func:`new temporary directory <tempfile.mkdtemp>`
1092 1111       to prevent e.g. exploiting pre-existing links, and to make it easier to
1093 1112       clean up after a failed extraction.
                 + * Disallow symbolic links if you do not need the functionality.
1094 1114     * When working with untrusted data, use external (e.g. OS-level) limits on
1095 1115       disk, memory and CPU usage.
```

```
1096  1116    * Check filenames against an allow-list of characters
```

Doc/whatsnew/3.11.rst

```
      @@ -2786,3 +2786,37 @@ email
2786  2786      check if the *strict* paramater is available.
2787  2787      (Contributed by Thomas Dwyer and Victor Stinner for :gh:`102988` to improve
2788  2788      the CVE-2023-27043 fix.)
      2789  +
      2790  +
      2791  + Notable changes in 3.11.13
      2792  + ==========================
      2793  +
      2794  + os.path
      2795  + -------
      2796  +
      2797  + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
      2798  +   :data:`os.path.ALLOW_MISSING`.
      2799  +   If used, errors other than :exc:`FileNotFoundError` will be re-raised;
      2800  +   the resulting path can be missing but it will be free of symlinks.
      2801  +   (Contributed by Petr Viktorin for CVE 2025-4517.)
      2802  +
      2803  + tarfile
      2804  + -------
      2805  +
      2806  + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
              to
      2807  +   avoid path traversal attacks.
      2808  +   (Contributed by Petr Viktorin in :gh:`127987` and CVE 2025-4138.)
      2809  + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
              attributes
      2810  +   when a directory was removed or replaced by another kind of file.
      2811  +   (Contributed by Petr Viktorin in :gh:`127987` and CVE 2024-12718.)
      2812  + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
      2813  +   now (re-)apply the extraction filter when substituting a link (hard or
      2814  +   symbolic) with a copy of another archive member, and when fixing up
      2815  +   directory attributes.
      2816  +   The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
      2817  +   (Contributed by Petr Viktorin for CVE 2025-4330 and CVE 2024-12718.)
      2818  + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
```

```
2819 +   no longer extract rejected members when
2820 +   :func:`~tarfile.TarFile.errorlevel` is zero.
2821 +   (Contributed by Matt Prodani and Petr Viktorin in :gh:`112887`
2822 +   and CVE 2025-4435.)
```

Lib/genericpath.py

```
@@ -8,7 +8,7 @@
 8  8
 9  9   __all__ = ['commonprefix', 'exists', 'getatime', 'getctime', 'getmtime',
10 10              'getsize', 'isdir', 'isfile', 'samefile', 'sameopenfile',
11    -            'samestat']
   11 +            'samestat', 'ALLOW_MISSING']
12 12
13 13
14 14   # Does a path exist?
@@ -153,3 +153,12 @@ def _check_arg_types(funcname, *args):
153 153                                f'os.PathLike object, not
    {s.__class__.__name__!r}') from None
154 154       if hasstr and hasbytes:
155 155           raise TypeError("Can't mix strings and bytes in path components") from
    None
   156 +
   157 + # A singleton with a true boolean value.
   158 + @object.__new__
   159 + class ALLOW_MISSING:
   160 +     """Special value for use in realpath()."""
   161 +     def __repr__(self):
   162 +         return 'os.path.ALLOW_MISSING'
   163 +     def __reduce__(self):
   164 +         return self.__class__.__name__
```

Lib/ntpath.py

```
@@ -30,7 +30,8 @@
30 30              "ismount", "expanduser","expandvars","normpath","abspath",
31 31              "curdir","pardir","sep","pathsep","defpath","altsep",
32 32
    "extsep","devnull","realpath","supports_unicode_filenames","relpath",
33    -            "samefile", "sameopenfile", "samestat", "commonpath"]
```

```
                +            "samefile", "sameopenfile", "samestat", "commonpath",
                +            "ALLOW_MISSING"]
34  35
35  36      def _get_bothseps(path):
36  37          if isinstance(path, bytes):
                 @@ -578,9 +579,10 @@ def abspath(path):
578 579          from nt import _getfinalpathname, readlink as _nt_readlink
579 580      except ImportError:
580 581          # realpath is a no-op on systems without _getfinalpathname support.
581         -        realpath = abspath
            582  +        def realpath(path, *, strict=False):
            583  +            return abspath(path)
582 584      else:
583         -        def _readlink_deep(path):
            585  +        def _readlink_deep(path, ignored_error=OSError):
584 586              # These error codes indicate that we should stop reading links and
585 587              # return the path we currently have.
586 588              # 1: ERROR_INVALID_FUNCTION
                 @@ -613,7 +615,7 @@ def _readlink_deep(path):
613 615                              path = old_path
614 616                              break
615 617                          path = normpath(join(dirname(old_path), path))
616         -                except OSError as ex:
            618  +                except ignored_error as ex:
617 619                      if ex.winerror in allowed_winerror:
618 620                          break
619 621                      raise
                 @@ -622,7 +624,7 @@ def _readlink_deep(path):
622 624                      break
623 625              return path
624 626
625         -    def _getfinalpathname_nonstrict(path):
            627  +    def _getfinalpathname_nonstrict(path, ignored_error=OSError):
626 628              # These error codes indicate that we should stop resolving the path
627 629              # and return the value we currently have.
628 630              # 1: ERROR_INVALID_FUNCTION
                 @@ -649,17 +651,18 @@ def _getfinalpathname_nonstrict(path):
649 651                  try:
```

```
650  652                      path = _getfinalpathname(path)
651  653                      return join(path, tail) if tail else path
652       -              except OSError as ex:
     654  +              except ignored_error as ex:
653  655                      if ex.winerror not in allowed_winerror:
654  656                          raise
655  657                      try:
656  658                          # The OS could not resolve this path fully, so we attempt
657  659                          # to follow the link ourselves. If we succeed, join the
                                tail
658  660                          # and return.
659       -                      new_path = _readlink_deep(path)
     661  +                      new_path = _readlink_deep(path,
     662  +                                                ignored_error=ignored_error)
660  663                      if new_path != path:
661  664                          return join(new_path, tail) if tail else new_path
662       -                  except OSError:
     665  +                  except ignored_error:
663  666                          # If we fail to readlink(), let's keep traversing
664  667                          pass
665  668                      path, name = split(path)
          @@ -690,24 +693,32 @@ def realpath(path, *, strict=False):
690  693              if normcase(path) == normcase(devnull):
691  694                  return '\\\\.\\NUL'
692  695          had_prefix = path.startswith(prefix)
     696  +
     697  +    if strict is ALLOW_MISSING:
     698  +        ignored_error = FileNotFoundError
     699  +        strict = True
     700  +    elif strict:
     701  +        ignored_error = ()
     702  +    else:
     703  +        ignored_error = OSError
     704  +
693  705          if not had_prefix and not isabs(path):
694  706              path = join(cwd, path)
695  707          try:
696  708              path = _getfinalpathname(path)
697  709              initial_winerror = 0
```

```
698  710          except ValueError as ex:
699  711              # gh-106242: Raised for embedded null characters
700       -          # In strict mode, we convert into an OSError.
     712  +          # In strict modes, we convert into an OSError.
701  713          # Non-strict mode returns the path as-is, since we've already
702  714          # made it absolute.
703  715          if strict:
704  716              raise OSError(str(ex)) from None
705  717          path = normpath(path)
706       -      except OSError as ex:
707       -          if strict:
708       -              raise
     718  +      except ignored_error as ex:
709  719          initial_winerror = ex.winerror
710       -          path = _getfinalpathname_nonstrict(path)
     720  +          path = _getfinalpathname_nonstrict(path,
     721  +                                             ignored_error=ignored_error)
711  722          # The path returned by _getfinalpathname will always start with \\?\ -
712  723          # strip off that prefix unless it was already provided on the original
713  724          # path.
```

Lib/posixpath.py

```
@@ -35,7 +35,7 @@
35  35              "samefile","sameopenfile","samestat",
36  36              "curdir","pardir","sep","pathsep","defpath","altsep","extsep",
37  37              "devnull","realpath","supports_unicode_filenames","relpath",
38       -          "commonpath"]
     38  +          "commonpath", "ALLOW_MISSING"]
39  39
40  40
41  41  def _get_sep(path):
@@ -427,6 +427,15 @@ def _joinrealpath(path, rest, strict, seen):
427  427          sep = '/'
428  428          curdir = '.'
429  429          pardir = '..'
     430  +        getcwd = os.getcwd
     431  +    if strict is ALLOW_MISSING:
     432  +        ignored_error = FileNotFoundError
```

```
                433 +     elif strict:
                434 +         ignored_error = ()
                435 +     else:
                436 +         ignored_error = OSError
                437 +
                438 +     maxlinks = None
430 439
431 440         if isabs(rest):
432 441             rest = rest[1:]
    @@ -449,9 +458,7 @@ def _joinrealpath(path, rest, strict, seen):
449 458             newpath = join(path, name)
450 459             try:
451 460                 st = os.lstat(newpath)
452     -         except OSError:
453     -             if strict:
454     -                 raise
                461 +         except ignored_error:
455 462                 is_link = False
456 463             else:
457 464                 is_link = stat.S_ISLNK(st.st_mode)
```

Lib/tarfile.py

```
    @@ -751,10 +751,22 @@ def __init__(self, tarinfo, path):
751 751             super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
752 752                              + 'which is outside the destination')
753 753
                754 + class LinkFallbackError(FilterError):
                755 +     def __init__(self, tarinfo, path):
                756 +         self.tarinfo = tarinfo
                757 +         self._path = path
                758 +         super().__init__(f'link {tarinfo.name!r} would be extracted as a '
                759 +                          + f'copy of {path!r}, which was rejected')
                760 +
                761 + # Errors caused by filters -- both "fatal" and "non-fatal" -- that
                762 + # we consider to be issues with the argument, rather than a bug in the
                763 + # filter function
                764 + _FILTER_ERRORS = (FilterError, OSError, ExtractError)
                765 +
754 766     def _get_filtered_attrs(member, dest_path, for_data=True):
```

```
755  767          new_attrs = {}
756  768          name = member.name
757       -      dest_path = os.path.realpath(dest_path)
     769  +      dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
758  770          # Strip leading / (tar's directory separator) from filenames.
759  771          # Include os.sep (target OS directory separator) as well.
760  772          if name.startswith(('/', os.sep)):
@@ -764,7 +776,8 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
764  776              # For example, 'C:/foo' on Windows.
765  777              raise AbsolutePathError(member)
766  778          # Ensure we stay in the destination
767       -      target_path = os.path.realpath(os.path.join(dest_path, name))
     779  +      target_path = os.path.realpath(os.path.join(dest_path, name),
     780  +                                     strict=os.path.ALLOW_MISSING)
768  781          if os.path.commonpath([target_path, dest_path]) != dest_path:
769  782              raise OutsideDestinationError(member, target_path)
770  783          # Limit permissions (no high bits, and go-w)
@@ -802,14 +815,18 @@ def _get_filtered_attrs(member, dest_path,
for_data=True):
802  815              if member.islnk() or member.issym():
803  816                  if os.path.isabs(member.linkname):
804  817                      raise AbsoluteLinkError(member)
     818  +               normalized = os.path.normpath(member.linkname)
     819  +               if normalized != member.linkname:
     820  +                   new_attrs['linkname'] = normalized
805  821                  if member.issym():
806  822                      target_path = os.path.join(dest_path,
807  823                                                 os.path.dirname(name),
808  824                                                 member.linkname)
809  825                  else:
810  826                      target_path = os.path.join(dest_path,
811  827                                                 member.linkname)
812       -              target_path = os.path.realpath(target_path)
     828  +              target_path = os.path.realpath(target_path,
     829  +                                             strict=os.path.ALLOW_MISSING)
813  830                  if os.path.commonpath([target_path, dest_path]) != dest_path:
814  831                      raise LinkOutsideDestinationError(member, target_path)
815  832          return new_attrs
```

```
@@ -2283,30 +2300,58 @@ def extractall(self, path=".", members=None, *,
                numeric_owner=False,
2283 2300             members = self
2284 2301
2285 2302             for member in members:
2286      -                tarinfo = self._get_extract_tarinfo(member, filter_function,
                           path)
     2303 +                tarinfo, unfiltered = self._get_extract_tarinfo(
     2304 +                    member, filter_function, path)
2287 2305                 if tarinfo is None:
2288 2306                     continue
2289 2307                 if tarinfo.isdir():
2290 2308                     # For directories, delay setting attributes until later,
2291 2309                     # since permissions can interfere with extraction and
2292 2310                     # extracting contents can reset mtime.
2293      -                    directories.append(tarinfo)
     2311 +                    directories.append(unfiltered)
2294 2312                 self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
2295      -                                  numeric_owner=numeric_owner)
     2313 +                                  numeric_owner=numeric_owner,
     2314 +                                  filter_function=filter_function)
2296 2315
2297 2316             # Reverse sort directories.
2298 2317             directories.sort(key=lambda a: a.name, reverse=True)
2299 2318
     2319 +
2300 2320             # Set correct owner, mtime and filemode on directories.
2301      -            for tarinfo in directories:
2302      -                dirpath = os.path.join(path, tarinfo.name)
     2321 +            for unfiltered in directories:
2303 2322                 try:
     2323 +                    # Need to re-apply any filter, to take the *current*
                           filesystem
     2324 +                    # state into account.
     2325 +                    try:
     2326 +                        tarinfo = filter_function(unfiltered, path)
     2327 +                    except _FILTER_ERRORS as exc:
     2328 +                        self._log_no_directory_fixup(unfiltered, repr(exc))
     2329 +                        continue
     2330 +                    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,
```

```
           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)
```

```
2329  2375          def _get_extract_tarinfo(self, member, filter_function, path):
2330        -         """Get filtered TarInfo (or None) from member, which might be a
                    str"""
      2376  +       """Get (filtered, unfiltered) TarInfos from *member*
      2377  +
      2378  +       *member* might be a string.
      2379  +
      2380  +       Return (None, None) if not found.
      2381  +       """
      2382  +
2331  2383          if isinstance(member, str):
2332        -             tarinfo = self.getmember(member)
      2384  +             unfiltered = self.getmember(member)
2333  2385          else:
2334        -             tarinfo = member
      2386  +             unfiltered = member
2335  2387
2336        -       unfiltered = tarinfo
      2388  +       filtered = None
2337  2389          try:
2338        -             tarinfo = filter_function(tarinfo, path)
      2390  +             filtered = filter_function(unfiltered, path)
2339  2391          except (OSError, FilterError) as e:
2340  2392              self._handle_fatal_error(e)
2341  2393          except ExtractError as e:
2342  2394              self._handle_nonfatal_error(e)
2343        -       if tarinfo is None:
      2395  +       if filtered is None:
2344  2396              self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2345        -             return None
      2397  +             return None, None
      2398  +
2346  2399          # Prepare the link target for makelink().
2347        -       if tarinfo.islnk():
2348        -             tarinfo = copy.copy(tarinfo)
2349        -             tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2350        -       return tarinfo
      2400  +       if filtered.islnk():
      2401  +             filtered = copy.copy(filtered)
      2402  +             filtered._link_target = os.path.join(path, filtered.linkname)
```

```
                2403 +          return filtered, unfiltered
2351            2404
2352            -       def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2353            -           """Extract from filtered tarinfo to disk"""
                2405 +       def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
                2406 +                        filter_function=None):
                2407 +           """Extract from filtered tarinfo to disk.
                2408 +
                2409 +              filter_function is only used when extracting a *different*
                2410 +              member (e.g. as fallback to creating a symlink)
                2411 +           """
2354            2412               self._check("r")
2355            2413
2356            2414               try:
2357            2415                   self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2358            2416                                        set_attrs=set_attrs,
2359            -                                       numeric_owner=numeric_owner)
                2417 +                                       numeric_owner=numeric_owner,
                2418 +                                       filter_function=filter_function,
                2419 +                                       extraction_root=path)
2360            2420               except OSError as e:
2361            2421                   self._handle_fatal_error(e)
2362            2422               except ExtractError as e:
                    @@ -2414,9 +2474,13 @@ def extractfile(self, member):
2414            2474                   return None
2415            2475
2416            2476           def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2417            -                          numeric_owner=False):
2418            -           """Extract the TarInfo object tarinfo to a physical
                2477 +                          numeric_owner=False, *, filter_function=None,
                2478 +                          extraction_root=None):
                2479 +           """Extract the filtered TarInfo object tarinfo to a physical
2419            2480               file called targetpath.
                2481 +
                2482 +              filter_function is only used when extracting a *different*
                2483 +              member (e.g. as fallback to creating a symlink)
2420            2484               """
2421            2485               # Fetch the TarInfo object for the given name
2422            2486               # and build the destination pathname, replacing
```

```
@@ -2445,7 +2509,10 @@ def _extract_member(self, tarinfo, targetpath,
set_attrs=True,
2445 2509             elif tarinfo.ischr() or tarinfo.isblk():
2446 2510                 self.makedev(tarinfo, targetpath)
2447 2511             elif tarinfo.islnk() or tarinfo.issym():
2448      -              self.makelink(tarinfo, targetpath)
     2512 +              self.makelink_with_filter(
     2513 +                  tarinfo, targetpath,
     2514 +                  filter_function=filter_function,
     2515 +                  extraction_root=extraction_root)
2449 2516             elif tarinfo.type not in SUPPORTED_TYPES:
2450 2517                 self.makeunknown(tarinfo, targetpath)
2451 2518             else:
@@ -2528,29 +2595,57 @@ def makedev(self, tarinfo, targetpath):
2528 2595                      os.makedev(tarinfo.devmajor, tarinfo.devminor))
2529 2596
2530 2597         def makelink(self, tarinfo, targetpath):
     2598 +           return self.makelink_with_filter(tarinfo, targetpath, None, None)
     2599 +
     2600 +       def makelink_with_filter(self, tarinfo, targetpath,
     2601 +                                filter_function, extraction_root):
2531 2602             """Make a (symbolic) link called targetpath. If it cannot be created
2532 2603               (platform limitation), we try to make a copy of the referenced file
2533 2604               instead of a link.
     2605 +
     2606 +           filter_function is only used when extracting a *different*
     2607 +           member (e.g. as fallback to creating a link).
2534 2608             """
     2609 +           keyerror_to_extracterror = False
2535 2610             try:
2536 2611                 # For systems that support symbolic and hard links.
2537 2612                 if tarinfo.issym():
2538 2613                     if os.path.lexists(targetpath):
2539 2614                         # Avoid FileExistsError on following os.symlink.
2540 2615                         os.unlink(targetpath)
2541 2616                     os.symlink(tarinfo.linkname, targetpath)
     2617 +                   return
2542 2618                 else:
2543 2619                     if os.path.exists(tarinfo._link_target):
```

```
2544  2620                      os.link(tarinfo._link_target, targetpath)
2545        -                 else:
2546        -                     self._extract_member(self._find_link_target(tarinfo),
2547        -                                          targetpath)
      2621  +                     return
2548  2622              except symlink_exception:
      2623  +             keyerror_to_extracterror = True
      2624  +
      2625  +         try:
      2626  +             unfiltered = self._find_link_target(tarinfo)
      2627  +         except KeyError:
      2628  +             if keyerror_to_extracterror:
      2629  +                 raise ExtractError(
      2630  +                     "unable to resolve link inside archive") from None
      2631  +             else:
      2632  +                 raise
      2633  +
      2634  +         if filter_function is None:
      2635  +             filtered = unfiltered
      2636  +         else:
      2637  +             if extraction_root is None:
      2638  +                 raise ExtractError(
      2639  +                     "makelink_with_filter: if filter_function is not None, "
      2640  +                     + "extraction_root must also not be None")
2549  2641              try:
2550        -                 self._extract_member(self._find_link_target(tarinfo),
2551        -                                      targetpath)
2552        -             except KeyError:
2553        -                 raise ExtractError("unable to resolve link inside archive")
            from None
      2642  +                 filtered = filter_function(unfiltered, extraction_root)
      2643  +             except _FILTER_ERRORS as cause:
      2644  +                 raise LinkFallbackError(tarinfo, unfiltered.name) from cause
      2645  +         if filtered is not None:
      2646  +             self._extract_member(filtered, targetpath,
      2647  +                                  filter_function=filter_function,
      2648  +                                  extraction_root=extraction_root)
2554  2649
2555  2650      def chown(self, tarinfo, targetpath, numeric_owner):
2556  2651          """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