

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.)

408 408 system). On Windows, this function will also resolve MS-DOS (also called 8.3)

409 409 style names such as ``C:\PROGRA~1`` to ``C:\Program Files``.

410 410

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

411 + By default, the path is evaluated up to the first component that does not  
 412 + exist, is a symlink loop, or whose evaluation raises :exc:`OSError`.  
 413 + All such components are appended unchanged to the existing part of the path.  
 414 +

415 + Some errors that are handled this way include "access denied", "not a  
 416 + directory", or "bad argument to internal function". Thus, the  
 417 + resulting path may be missing or inaccessible, may still contain  
 418 + links or loops, and may traverse non-directories.  
 419 +

420 + This behavior can be modified by keyword arguments:

421 +

422 + If *strict* is ``True``, the first error encountered when evaluating the  
 path is

423 + re-raised.

424 + In particular, :exc:`FileNotFoundError` is raised if *path* does not exist,  
 425 + or another :exc:`OSError` if it is otherwise inaccessible.

426 +

```

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	+	.. <code>versionadded:: next</code>
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 <code>``'data'``</code> filter.
1069	1078	In addition to what <code>``tar_filter``</code> does:
1070	1079	
1080	+	- Normalize link targets ( <code>:attr:TarInfo.linkname`</code> ) using
1081	+	<code>:func:os.path.normpath`</code> .
1082	+	Note that this removes internal <code>``..``</code> components, which may change the
1083	+	meaning of the link if the path in <code>:attr:TarInfo.linkname`</code> traverses
1084	+	symbolic links.
1085	+	
1071	1086	- <code>:ref:Refuse &lt;tarfile-extraction-refuse&gt;</code> 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 <i>all</i> dangerous archive features.
1100	1115	See <code>:ref:tarfile-further-verification`</code> for details.
1101	1116	
1117	+	.. <code>versionchanged:: next</code>
1118	+	
1119	+	Link targets are now normalized.
1120	+	
1102	1121	
1103	1122	.. <code>_tarfile-extraction-refuse:</code>
1104	1123	
↓		@@ -1127,6 +1146,7 @@ Here is an incomplete list of things to consider:
↑		
1127	1146	* Extract to a <code>:func:new temporary directory &lt;tempfile.mkdtemp&gt;</code>
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

116 116 (Contributed by Sergey B Kirpichev in [:gh:`132908`](#).)

117 117

118 118

119 + os.path

120 + -----

121 +

122 + \* The *strict* parameter to `:func:`os.path.realpath`` accepts a new value,123 + `:data:`os.path.ALLOW_MISSING``.124 + If used, errors other than `:exc:`FileNotFoundError`` will be re-raised;

125 + the resulting path can be missing but it will be free of symlinks.

126 + (Contributed by Petr Viktorin for [:cve:`2025-4517`](#).)

127 +

128 +

119 129 shelve

120 130 -----

121 131



@@ -132,6 +142,28 @@ ssl

132 142 (Contributed by Will Childs-Klein in [:gh:`133624`](#).)

133 143

134 144

145 + tarfile

146 + -----

147 +

148 + \* `:func:`~tarfile.data_filter`` now normalizes symbolic link targets in order to

149 + avoid path traversal attacks.

150 + (Contributed by Petr Viktorin in [:gh:`127987`](#) and [:cve:`2025-4138`](#).)151 + \* `:func:`~tarfile.TarFile.extractall`` now skips fixing up directory attributes

152 + when a directory was removed or replaced by another kind of file.

153 + (Contributed by Petr Viktorin in [:gh:`127987`](#) and [:cve:`2024-12718`](#).)154 + \* `:func:`~tarfile.TarFile.extract`` and `:func:`~tarfile.TarFile.extractall``

155 + now (re-)apply the extraction filter when substituting a link (hard or

156 + symbolic) with a copy of another archive member, and when fixing up

157 + directory attributes.

158 + The former raises a new exception, `:exc:`~tarfile.LinkFallbackError``.159 + (Contributed by Petr Viktorin for [:cve:`2025-4330`](#) and [:cve:`2024-12718`](#).)160 + \* `: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 Prodan 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 +
731 +     elif strict:
732 +         ignored_error = ()
733 +     else:
734 +         ignored_error = OSError

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 39         "commonpath", "isjunction", "isdevdrive"]
39 39 +         "commonpath", "isjunction", "isdevdrive", "ALLOW_MISSING"]
40 40
41 41
42 42     def _get_sep(path):
43 43
44 44     @@ -402,10 +402,18 @@ def realpath(filename, *, strict=False):
45 45
46 46         curdir = '.'
47 47         pardir = '..'
48 48         getcwd = os.getcwd
49 49
50 49 -     return _realpath(filename, strict, sep, curdir, pardir, getcwd)
51 50 +     if strict is ALLOW_MISSING:
52 51 +         ignored_error = FileNotFoundError
53 52 +         strict = True
54 53 +     elif strict:
55 54 +         ignored_error = ()
56 55 +     else:
57 56 +         ignored_error = OSError
58 57 +
59 58 +     lstat = os.lstat
60 59 +     readlink = os.readlink
61 60 +     maxlinks = None
62 61
63 62
64 63
65 64
66 65
67 66
68 67
69 68
70 69
71 70
72 71
73 72
74 73
75 74
76 75
77 76
78 77
79 78
80 79
81 80
82 81
83 82
84 83
85 84
86 85
87 86
88 87
89 88
90 89
91 90
92 91
93 92
94 93
95 94
96 95
97 96
98 97
99 98
100 99
101 100
102 101
103 102
104 103
105 104
106 105
107 106
108 107
109 108
110 109
111 110
112 111
113 112
114 113
115 114
116 115
117 116
118 117
119 118
120 119
121 120
122 121
123 122
124 123
125 124
126 125
127 126
128 127
129 128
130 129
131 130
132 131
133 132
134 133
135 134
136 135
137 136
138 137
139 138
140 139
141 140
142 141
143 142
144 143
145 144
146 145
147 146
148 147
149 148
150 149
151 150
152 151
153 152
154 153
155 154
156 155
157 156
158 157
159 158
160 159
161 160
162 161
163 162
164 163
165 164
166 165
167 166
168 167
169 168
170 169
171 170
172 171
173 172
174 173
175 174
176 175
177 176
178 177
179 178
180 179
181 180
182 181
183 182
184 183
185 184
186 185
187 186
188 187
189 188
190 189
191 190
192 191
193 192
194 193
195 194
196 195
197 196
198 197
199 198
200 199
201 200
202 201
203 202
204 203
205 204
206 205
207 206
208 207
209 208
210 209
211 210
212 211
213 212
214 213
215 214
216 215
217 216
218 217
219 218
220 219
221 220
222 221
223 222
224 223
225 224
226 225
227 226
228 227
229 228
230 229
231 230
232 231
233 232
234 233
235 234
236 235
237 236
238 237
239 238
240 239
241 240
242 241
243 242
244 243
245 244
246 245
247 246
248 247
249 248
250 249
251 250
252 251
253 252
254 253
255 254
256 255
257 256
258 257
259 258
260 259
261 260
262 261
263 262
264 263
265 264
266 265
267 266
268 267
269 268
270 269
271 270
272 271
273 272
274 273
275 274
276 275
277 276
278 277
279 278
280 279
281 280
282 281
283 282
284 283
285 284
286 285
287 286
288 287
289 288
290 289
291 290
292 291
293 292
294 293
295 294
296 295
297 296
298 297
299 298
300 299
301 300
302 301
303 302
304 303
305 304
306 305
307 306
308 307
309 308
310 309
311 310
312 311
313 312
314 313
315 314
316 315
317 316
318 317
319 318
320 319
321 320
322 321
323 322
324 323
325 324
326 325
327 326
328 327
329 328
330 329
331 330
332 331
333 332
334 333
335 334
336 335
337 336
338 337
339 338
340 339
341 340
342 341
343 342
344 343
345 344
346 345
347 346
348 347
349 348
350 349
351 350
352 351
353 352
354 353
355 354
356 355
357 356
358 357
359 358
360 359
361 360
362 361
363 362
364 363
365 364
366 365
367 366
368 367
369 368
370 369
371 370
372 371
373 372
374 373
375 374
376 375
377 376
378 377
379 378
380 379
381 380
382 381
383 382
384 383
385 384
386 385
387 386
388 387
389 388
390 389
391 390
392 391
393 392
394 393
395 394
396 395
397 396
398 397
399 398
400 399
401 400
402 401
403 402
404 403
405 404
406 405
407 406
408 407
409 408
410 409
411 410
412 411
413 412
414 413
415 414
416 415
417 416
418 417
419 418
420 419
421 420
422 421
423 422
424 423
425 424
426 425
427 426
428 427
429 428
430 429
431 430
432 431
433 432
434 433
435 434
436 435
437 436
438 437
439 438
440 439
441 440
442 441
443 442
444 443
445 444
446 445
447 446
448 447
449 448
450 449
451 450
452 451
453 452
454 453
455 454
456 455
457 456
458 457
459 458
460 459
461 460
462 461
463 462
464 463
465 464
466 465
467 466
468 467
469 468
470 469
471 470
472 471
473 472
474 473
475 474
476 475
477 476
478 477
479 478
480 479
481 480
482 481
483 482
484 483
485 484
486 485
487 486
488 487
489 488
490 489
491 490
492 491
493 492
494 493
495 494
496 495
497 496
498 497
499 498
500 499
501 500
502 501
503 502
504 503
505 504
506 505
507 506
508 507
509 508
510 509
511 510
512 511
513 512
514 513
515 514
516 515
517 516
518 517
519 518
520 519
521 520
522 521
523 522
524 523
525 524
526 525
527 526
528 527
529 528
530 529
531 530
532 531
533 532
534 533
535 534
536 535
537 536
538 537
539 538
540 539
541 540
542 541
543 542
544 543
545 544
546 545
547 546
548 547
549 548
550 549
551 550
552 551
553 552
554 553
555 554
556 555
557 556
558 557
559 558
560 559
561 560
562 561
563 562
564 563
565 564
566 565
567 566
568 567
569 568
570 569
571 570
572 571
573 572
574 573
575 574
576 575
577 576
578 577
579 578
580 579
581 580
582 581
583 582
584 583
585 584
586 585
587 586
588 587
589 588
590 589
591 590
592 591
593 592
594 593
595 594
596 595
597 596
598 597
599 598
600 599
601 600
602 601
603 602
604 603
605 604
606 605
607 606
608 607
609 608
610 609
611 610
612 611
613 612
614 613
615 614
616 615
617 616
618 617
619 618
620 619
621 620
622 621
623 622
624 623
625 624
626 625
627 626
628 627
629 628
630 629
631 630
632 631
633 632
634 633
635 634
636 635
637 636
638 637
639 638
640 639
641 640
642 641
643 642
644 643
645 644
646 645
647 646
648 647
649 648
650 649
651 650
652 651
653 652
654 653
655 654
656 655
657 656
658 657
659 658
660 659
661 660
662 661
663 662
664 663
665 664
666 665
667 666
668 667
669 668
670 669
671 670
672 671
673 672
674 673
675 674
676 675
677 676
678 677
679 678
680 679
681 680
682 681
683 682
684 683
685 684
686 685
687 686
688 687
689 688
690 689
691 690
692 691
693 692
694 693
695 694
696 695
697 696
698 697
699 698
700 699
701 700
702 701
703 702
704 703
705 704
706 705
707 706
708 707
709 708
710 709
711 710
712 711
713 712
714 713
715 714
716 715
717 716
718 717
719 718
720 719
721 720
722 721
723 722
724 723
725 724
726 725
727 726
728 727
729 728
730 729
731 730
732 731
733 732
734 733
735 734
736 735
737 736
738 737
739 738
740 739
741 740
742 741
743 742
744 743
745 744
746 745
747 746
748 747
749 748
750 749
751 750
752 751
753 752
754 753
755 754
756 755
757 756
758 757
759 758
760 759
761 760
762 761
763 762
764 763
765 764
766 765
767 766
768 767
769 768
770 769
771 770
772 771
773 772
774 773
775 774
776 775
777 776
778 777
779 778
780 779
781 780
782 781
783 782
784 783
785 784
786 785
787 786
788 787
789 788
790 789
791 790
792 791
793 792
794 793
795 794
796 795
797 796
798 797
799 798
800 799
801 800
802 801
803 802
804 803
805 804
806 805
807 806
808 807
809 808
810 809
811 810
812 811
813 812
814 813
815 814
816 815
817 816
818 817
819 818
820 819
821 820
822 821
823 822
824 823
825 824
826 825
827 826
828 827
829 828
830 829
831 830
832 831
833 832
834 833
835 834
836 835
837 836
838 837
839 838
840 839
841 840
842 841
843 842
844 843
845 844
846 845
847 846
848 847
849 848
850 849
851 850
852 851
853 852
854 853
855 854
856 855
857 856
858 857
859 858
860 859
861 860
862 861
863 862
864 863
865 864
866 865
867 866
868 867
869 868
870 869
871 870
872 871
873 872
874 873
875 874
876 875
877 876
878 877
879 878
880 879
881 880
882 881
883 882
884 883
885 884
886 885
887 886
888 887
889 888
890 889
891 890
892 891
893 892
894 893
895 894
896 895
897 896
898 897
899 898
900 899
901 900
902 901
903 902
904 903
905 904
906 905
907 906
908 907
909 908
910 909
911 910
912 911
913 912
914 913
915 914
916 915
917 916
918 917
919 918
920 919
921 920
922 921
923 922
924 923
925 924
926 925
927 926
928 927
929 928
930 929
931 930
932 931
933 932
934 933
935 934
936 935
937 936
938 937
939 938
940 939
941 940
942 941
943 942
944 943
945 944
946 945
947 946
948 947
949 948
950 949
951 950
952 951
953 952
954 953
955 954
956 955
957 956
958 957
959 958
960 959
961 960
962 961
963 962
964 963
965 964
966 965
967 966
968 967
969 968
970 969
971 970
972 971
973 972
974 973
975 974
976 975
977 976
978 977
979 978
980 979
981 980
982 981
983 982
984 983
985 984
986 985
987 986
988 987
989 988
990 989
991 990
992 991
993 992
994 993
995 994
996 995
997 996
998 997
999 998
1000 999

```

```
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)
773 784 +         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('/'):
@@ -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:

```

```

2389 -         tarinfo = self._get_extract_tarinfo(member, filter_function,
        path)
2406 +         tarinfo, unfiltered = self._get_extract_tarinfo(
2407 +             member, filter_function, path)
2390 2408         if tarinfo is None:
2391 2409             continue
2392 2410         if tarinfo.isdir():
2393 2411             # For directories, delay setting attributes until later,
2394 2412             # since permissions can interfere with extraction and
2395 2413             # extracting contents can reset mtime.
2396 -         directories.append(tarinfo)
2414 +         directories.append(unfiltered)
2397 2415         self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),
2398 -                             numeric_owner=numeric_owner)
2416 +                             numeric_owner=numeric_owner,
2417 +                             filter_function=filter_function)
2399 2418
2400 2419         # Reverse sort directories.
2401 2420         directories.sort(key=lambda a: a.name, reverse=True)
2402 2421
2422 +
2403 2423         # Set correct owner, mtime and filemode on directories.
2404 -         for tarinfo in directories:
2405 -             dirpath = os.path.join(path, tarinfo.name)
2424 +         for unfiltered in directories:
2406 2425             try:
2426 +                 # Need to re-apply any filter, to take the *current*
                filesystem
2427 +                 # state into account.
2428 +                 try:
2429 +                     tarinfo = filter_function(unfiltered, path)
2430 +                 except _FILTER_ERRORS as exc:
2431 +                     self._log_no_directory_fixup(unfiltered, repr(exc))
2432 +                     continue
2433 +                 if tarinfo is None:
2434 +                     self._log_no_directory_fixup(unfiltered,
2435 +                                                     'excluded by filter')
2436 +                     continue
2437 +                     dirpath = os.path.join(path, tarinfo.name)
2438 +                 try:

```

```

2439 +         lstat = os.lstat(dirpath)
2440 +     except FileNotFoundError:
2441 +         self._log_no_directory_fixup(tarinfo, 'missing')
2442 +         continue
2443 +     if not stat.S_ISDIR(lstat.st_mode):
2444 +         # This is no longer a directory; presumably a later
2445 +         # member overwrote the entry.
2446 +         self._log_no_directory_fixup(tarinfo, 'not a directory')
2447 +         continue
2407 2448         self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
2408 2449         self.utime(tarinfo, dirpath)
2409 2450         self.chmod(tarinfo, dirpath)
2410 2451     except ExtractError as e:
2411 2452         self._handle_nonfatal_error(e)
2412 2453
2454 +     def _log_no_directory_fixup(self, member, reason):
2455 +         self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
2456 +             (member.name, reason))
2457 +
2413 2458     def extract(self, member, path="", set_attrs=True, *,
2414 2459         numeric_owner=False,
2415 2460         filter=None):
2416 2461         """Extract a member from the archive to the current working
2417 2462         directory,
2418 2463
2419 2464         @@ -2425,41 +2470,56 @@ def extract(self, member, path="",
2420 2465         set_attrs=True, *, numeric_owner=False,
2421 2466
2422 2467         String names of common filters are accepted.
2423 2468         """
2424 2469         filter_function = self._get_filter_function(filter)
2425 2470         tarinfo = self._get_extract_tarinfo(member, filter_function, path)
2426 2471         tarinfo, unfiltered = self._get_extract_tarinfo(
2427 2472             member, filter_function, path)
2428 2473         if tarinfo is not None:
2429 2474             self._extract_one(tarinfo, path, set_attrs, numeric_owner)
2430 2475
2431 2476     def _get_extract_tarinfo(self, member, filter_function, path):
2432 2477         """Get filtered TarInfo (or None) from member, which might be a
2433 2478         str"""
2434 2479         """Get (filtered, unfiltered) TarInfos from *member*
2435 2480

```

```

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.

```

```

2454 2511
2455 - def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2456 -     """Extract from filtered tarinfo to disk"""
2512 +     filter_function is only used when extracting a *different*
2513 +     member (e.g. as fallback to creating a symlink)
2514 +     """
2457 2515         self._check("r")
2458 2516
2459 2517         try:
2460 2518             self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2461 2519                                 set_attrs=set_attrs,
2462 -                                     numeric_owner=numeric_owner)
2520 +                                     numeric_owner=numeric_owner,
2521 +                                     filter_function=filter_function,
2522 +                                     extraction_root=path)
2463 2523         except (OSError, UnicodeEncodeError) as e:
2464 2524             self._handle_fatal_error(e)
2465 2525         except ExtractError as e:
@@ -2517,9 +2577,13 @@ def extractfile(self, member):
2517 2577             return None
2518 2578
2519 2579         def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2520 -                             numeric_owner=False):
2521 -             """Extract the TarInfo object tarinfo to a physical
2580 +                             numeric_owner=False, *, filter_function=None,
2581 +                             extraction_root=None):
2582 +             """Extract the filtered TarInfo object tarinfo to a physical
2522 2583                 file called targetpath.
2584 +
2585 +                 filter_function is only used when extracting a *different*
2586 +                 member (e.g. as fallback to creating a symlink)
2523 2587                 """
2524 2588                 # Fetch the TarInfo object for the given name
2525 2589                 # and build the destination pathname, replacing
@@ -2548,7 +2612,10 @@ def _extract_member(self, tarinfo, targetpath,
set_attrs=True,
2548 2612                 elif tarinfo.ischr() or tarinfo.isblk():
2549 2613                     self.makedev(tarinfo, targetpath)
2550 2614                 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
```



... [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 @@
1 + Fixes multiple issues that allowed ``tarfile`` extraction filters
2 + (``filter="data"`` and ``filter="tar"``) to be bypassed using crafted
3 + symlinks and hard links.
4 +
5 + Addresses :cve:`2024-12718`, :cve:`2025-4138`, :cve:`2025-4330`, and :cve:`2025-
6 + 4517`.

```

Comments 0