

python / cpython Public

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

Commit aa9eb5f



6 people authored on Jun 3, 2025 · ✖ 78 / 88 · Partially verified

[3.13] [gh-135034](#): Normalize link targets in tarfile, add `os.path.realpath(strict='allow_missing')` ([GH-135037](#)) ([GH-135064](#))

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>

[3.13](#) (AcreationOS-Linux/python#2, #135064) · v3.13.13 ... v3.13.4

1 parent [9f3d999](#) commit aa9eb5f

11 files changed +965 -164 ●●●●● ●

↑ Top

🔍 Filter files...

- ✓ Doc
 - ✓ library
 - os.path.rst
 - tarfile.rst
 - ✓ whatsnew
 - 3.13.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

```

@@ -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

252 + .. exception:: LinkFallbackError
253 +
254 +     Raised to refuse emulating a link (hard or symbolic) by extracting another
255 +     archive member, when that member would be rejected by the filter location.
256 +     The exception that was raised to reject the replacement member is
available
257 +     as :attr:`!BaseException.__context__`.
258 +

```

259	+	.. <code>versionadded:: next</code>
260	+	
252	261	
253	262	The following constants are available at the module level:
254	263	
⋮ ↓ ↑ ⋮		@@ -1052,6 +1061,12 @@ reused in custom filters:
1052	1061	Implements the ``'data'`` filter.
1053	1062	In addition to what ``tar_filter`` does:
1054	1063	
1064	+	- Normalize link targets (<code>:attr:TarInfo.linkname`</code>) using
1065	+	<code>:func:os.path.normpath`</code> .
1066	+	Note that this removes internal ``..`` components, which may change the
1067	+	meaning of the link if the path in <code>:attr:!TarInfo.linkname`</code> traverses
1068	+	symbolic links.
1069	+	
1055	1070	- <code>:ref:Refuse <tarfile-extraction-refuse></code> to extract links (hard or soft)
1056	1071	that link to absolute paths, or ones that link outside the destination.
1057	1072	
⋮ ↓ ↑ ⋮		@@ -1080,6 +1095,10 @@ reused in custom filters:
1080	1095	
1081	1096	Return the modified ``TarInfo`` member.
1082	1097	
1098	+	.. <code>versionchanged:: next</code>
1099	+	
1100	+	Link targets are now normalized.
1101	+	
1083	1102	
1084	1103	.. <code>_tarfile-extraction-refuse:</code>
1085	1104	
⋮ ↕ ⋮		@@ -1106,6 +1125,7 @@ Here is an incomplete list of things to consider:
1106	1125	* Extract to a <code>:func:new temporary directory <tempfile.mkdtemp></code>
1107	1126	to prevent e.g. exploiting pre-existing links, and to make it easier to
1108	1127	clean up after a failed extraction.
1128	+	* Disallow symbolic links if you do not need the functionality.
1109	1129	* When working with untrusted data, use external (e.g. OS-level) limits on
1110	1130	disk, memory and CPU usage.
1111	1131	* Check filenames against an allow-list of characters
⋮ ↓ ⋮		

Doc/whatsnew/3.13.rst



```

↑... @@ -2829,3 +2829,36 @@ sys
2829 2829 * The previously undocumented special function :func:`sys.getobjects`,
2830 2830     which only exists in specialized builds of Python, may now return objects
2831 2831     from other interpreters than the one it's called in.

2832 +
2833 + Notable changes in 3.13.4
2834 + =====
2835 +
2836 + os.path
2837 + -----
2838 +
2839 + * The *strict* parameter to :func:`os.path.realpath` accepts a new value,
2840 + :data:`os.path.ALLOW_MISSING`.
2841 + If used, errors other than :exc:`FileNotFoundError` will be re-raised;
2842 + the resulting path can be missing but it will be free of symlinks.
2843 + (Contributed by Petr Viktorin for :cve:`2025-4517`.)
2844 +
2845 + tarfile
2846 + -----
2847 +
2848 + * :func:`~tarfile.data_filter` now normalizes symbolic link targets in order
2849 + to
2850 + avoid path traversal attacks. Add commentMore actions
2851 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2025-4138`.)
2852 + * :func:`~tarfile.TarFile.extractall` now skips fixing up directory
2853 + attributes
2854 + when a directory was removed or replaced by another kind of file.
2855 + (Contributed by Petr Viktorin in :gh:`127987` and :cve:`2024-12718`.)
2856 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2857 + now (re-)apply the extraction filter when substituting a link (hard or
2858 + symbolic) with a copy of another archive member, and when fixing up
2859 + directory attributes.
2860 + The former raises a new exception, :exc:`~tarfile.LinkFallbackError`.
2861 + (Contributed by Petr Viktorin for :cve:`2025-4330` and :cve:`2024-12718`.)
2862 + * :func:`~tarfile.TarFile.extract` and :func:`~tarfile.TarFile.extractall`
2863 + no longer extract rejected members when
2864 + :func:`~tarfile.TarFile.errorlevel` is zero.
2865 + (Contributed by Matt Prodan and Petr Viktorin in :gh:`112887`)

```

2864 + and :cve:`2025-4435`.)

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):
36         ↓
37         ↑
38         @@ -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     -     604         realpath = abspath
605     +     604         def realpath(path, *, strict=False):
606     +     605             return abspath(path)
607     606         else:
608     -     606         def _readlink_deep(path):
609     +     607         def _readlink_deep(path, ignored_error=OSError):
610     608             # These error codes indicate that we should stop reading links and
611     609             # return the path we currently have.
612     610             # 1: ERROR_INVALID_FUNCTION
613         ↓
614         ↑
615         @@ -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     -     639         except OSError as ex:
640     +     640         except ignored_error as ex:
641     641             if ex.winerror in allowed_winerror:
642     642                 break
643     643                 raise
644         ↕
645         @@ -645,7 +646,7 @@ def _readlink_deep(path):
645     646                 break
646     647                 return path
647     648
648     -     648         def _getfinalpathname_nonstrict(path):
649     +     649         def _getfinalpathname_nonstrict(path, ignored_error=OSError):
650     650             # These error codes indicate that we should stop resolving the path
651     651             # and return the value we currently have.
652     652             # 1: ERROR_INVALID_FUNCTION
653         ↓
654         ↑
655         @@ -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     -     676         except OSError as ex:

```



```

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

```

Lib/posixpath.py

```

@@ -36,7 +36,7 @@
36 36         "samefile", "sameopenfile", "samestat",
37 37         "curdir", "pardir", "sep", "pathsep", "defpath", "altsep", "extsep",
38 38         "devnull", "realpath", "supports_unicode_filenames", "relpath",
39 -         "commonpath", "isjunction", "isdevdrive"]
39 +         "commonpath", "isjunction", "isdevdrive", "ALLOW_MISSING"]
40 40
41 41
42 42     def _get_sep(path):

```

```

↑
@@ -402,6 +402,15 @@ def realpath(filename, *, strict=False):
402 402     curdir = '.'
403 403     pardir = '..'
404 404     getcwd = os.getcwd
405 +     if strict is ALLOW_MISSING:
406 +         ignored_error = FileNotFoundError
407 +         strict = True
408 +     elif strict:
409 +         ignored_error = ()
410 +     else:
411 +         ignored_error = OSError
412 +
413 +     maxlinks = None
405 414
406 415     # The stack of unresolved path parts. When popped, a special value of None
407 416     # indicates that a symlink target has been resolved, and that the original
↓
@@ -462,25 +471,28 @@ def realpath(filename, *, strict=False):
↑
462 471         path = newpath
463 472         continue
464 473         target = os.readlink(newpath)
465 -     except OSError:
466 -         if strict:
467 -             raise
468 -         path = newpath
474 +     except ignored_error:
475 +         pass
476 +     else:
477 +         # Resolve the symbolic link
478 +         if target.startswith(sep):
479 +             # Symlink target is absolute; reset resolved path.
480 +             path = sep
481 +         if maxlinks is None:
482 +             # Mark this symlink as seen but not fully resolved.
483 +             seen[newpath] = None
484 +             # Push the symlink path onto the stack, and signal its
specialness
485 +             # by also pushing None. When these entries are popped, we'll
486 +             # record the fully-resolved symlink target in the 'seen'
mapping.

```

```

487 +         rest.append(newpath)
488 +         rest.append(None)
489 +         # Push the unresolved symlink target parts onto the stack.
490 +         target_parts = target.split(sep)[::-1]
491 +         rest.extend(target_parts)
492 +         part_count += len(target_parts)
493
494 +         continue
495
496 -         # Resolve the symbolic link
497 -         seen[newpath] = None # not resolved symlink
498 -         if target.startswith(sep):
499 -             # Symlink target is absolute; reset resolved path.
500 -             path = sep
501 -             # Push the symlink path onto the stack, and signal its specialness by
502 -             # also pushing None. When these entries are popped, we'll record the
503 -             # fully-resolved symlink target in the 'seen' mapping.
504 -             rest.append(newpath)
505 -             rest.append(None)
506 -             # Push the unresolved symlink target parts onto the stack.
507 -             target_parts = target.split(sep)[::-1]
508 -             rest.extend(target_parts)
509 -             part_count += len(target_parts)
510 +             # An error occurred and was ignored.
511 +             path = newpath
512
513 496 498
514 497 500
515 498 501
516 499 502
517 500 503
518 501 504
519 502 507
520 503 508
521 504 509
522 505 510
523 506 511
524 507 512
525 508 513
526 509 514
527 510 515
528 511 516
529 512 517
530 513 518
531 514 519
532 515 520
533 516 521
534 517 522
535 518 523
536 519 524
537 520 525
538 521 526
539 522 527
540 523 528
541 524 529
542 525 530
543 526 531
544 527 532
545 528 533
546 529 534
547 530 535
548 531 536
549 532 537
550 533 538
551 534 539
552 535 540
553 536 541
554 537 542
555 538 543
556 539 544
557 540 545
558 541 546
559 542 547
560 543 548
561 544 549
562 545 550
563 546 551
564 547 552
565 548 553
566 549 554
567 550 555
568 551 556
569 552 557
570 553 558
571 554 559
572 555 560
573 556 561
574 557 562
575 558 563
576 559 564
577 560 565
578 561 566
579 562 567
580 563 568
581 564 569
582 565 570
583 566 571
584 567 572
585 568 573
586 569 574
587 570 575
588 571 576
589 572 577
590 573 578
591 574 579
592 575 580
593 576 581
594 577 582
595 578 583
596 579 584
597 580 585
598 581 586
599 582 587
600 583 588
601 584 589
602 585 590
603 586 591
604 587 592
605 588 593
606 589 594
607 590 595
608 591 596
609 592 597
610 593 598
611 594 599
612 595 600
613 596 601
614 597 602
615 598 603
616 599 604
617 600 605
618 601 606
619 602 607
620 603 608
621 604 609
622 605 610
623 606 611
624 607 612
625 608 613
626 609 614
627 610 615
628 611 616
629 612 617
630 613 618
631 614 619
632 615 620
633 616 621
634 617 622
635 618 623
636 619 624
637 620 625
638 621 626
639 622 627
640 623 628
641 624 629
642 625 630
643 626 631
644 627 632
645 628 633
646 629 634
647 630 635
648 631 636
649 632 637
650 633 638
651 634 639
652 635 640
653 636 641
654 637 642
655 638 643
656 639 644
657 640 645
658 641 646
659 642 647
660 643 648
661 644 649
662 645 650
663 646 651
664 647 652
665 648 653
666 649 654
667 650 655
668 651 656
669 652 657
670 653 658
671 654 659
672 655 660
673 656 661
674 657 662
675 658 663
676 659 664
677 660 665
678 661 666
679 662 667
680 663 668
681 664 669
682 665 670
683 666 671
684 667 672
685 668 673
686 669 674
687 670 675
688 671 676
689 672 677
690 673 678
691 674 679
692 675 680
693 676 681
694 677 682
695 678 683
696 679 684
697 680 685
698 681 686
699 682 687
700 683 688
701 684 689
702 685 690
703 686 691
704 687 692
705 688 693
706 689 694
707 690 695
708 691 696
709 692 697
710 693 698
711 694 699
712 695 700
713 696 701
714 697 702
715 698 703
716 699 704
717 700 705
718 701 706
719 702 707
720 703 708
721 704 709
722 705 710
723 706 711
724 707 712
725 708 713
726 709 714
727 710 715
728 711 716
729 712 717
730 713 718
731 714 719
732 715 720
733 716 721
734 717 722
735 718 723
736 719 724
737 720 725
738 721 726
739 722 727
740 723 728
741 724 729
742 725 730
743 726 731
744 727 732
745 728 733
746 729 734
747 730 735
748 731 736
749 732 737
750 733 738
751 734 739
752 735 740
753 736 741
754 737 742
755 738 743
756 739 744
757 740 745
758 741 746
759 742 747
760 743 748
761 744 749
762 745 750
763 746 751
764 747 752
765 748 753
766 749 754
767 750 755
768 751 756
769 752 757
770 753 758
771 754 759
772 755 760
773 756 761
774 757 762
775 758 763
776 759 764
777 760 765
778 761 766
779 762 767
780 763 768
781 764 769
782 765 770
783 766 771
784 767 772
785 768 773
786 769 774
787 770 775
788 771 776
789 772 777
790 773 778
791 774 779
792 775 780
793 776 781
794 777 782
795 778 783
796 779 784
797 780 785
798 781 786
799 782 787
800 783 788
801 784 789
802 785 790
803 786 791
804 787 792
805 788 793
806 789 794
807 790 795
808 791 796
809 792 797
810 793 798
811 794 799
812 795 800
813 796 801
814 797 802
815 798 803
816 799 804
817 800 805
818 801 806
819 802 807
820 803 808
821 804 809
822 805 810
823 806 811
824 807 812
825 808 813
826 809 814
827 810 815
828 811 816
829 812 817
830 813 818
831 814 819
832 815 820
833 816 821
834 817 822
835 818 823
836 819 824
837 820 825
838 821 826
839 822 827
840 823 828
841 824 829
842 825 830
843 826 831
844 827 832
845 828 833
846 829 834
847 830 835
848 831 836
849 832 837
850 833 838
851 834 839
852 835 840
853 836 841
854 837 842
855 838 843
856 839 844
857 840 845
858 841 846
859 842 847
860 843 848
861 844 849
862 845 850
863 846 851
864 847 852
865 848 853
866 849 854
867 850 855
868 851 856
869 852 857
870 853 858
871 854 859
872 855 860
873 856 861
874 857 862
875 858 863
876 859 864
877 860 865
878 861 866
879 862 867
880 863 868
881 864 869
882 865 870
883 866 871
884 867 872
885 868 873
886 869 874
887 870 875
888 871 876
889 872 877
890 873 878
891 874 879
892 875 880
893 876 881
894 877 882
895 878 883
896 879 884
897 880 885
898 881 886
899 882 887
900 883 888
901 884 889
902 885 890
903 886 891
904 887 892
905 888 893
906 889 894
907 890 895
908 891 896
909 892 897
910 893 898
911 894 899
912 895 900
913 896 901
914 897 902
915 898 903
916 899 904
917 900 905
918 901 906
919 902 907
920 903 908
921 904 909
922 905 910
923 906 911
924 907 912
925 908 913
926 909 914
927 910 915
928 911 916
929 912 917
930 913 918
931 914 919
932 915 920
933 916 921
934 917 922
935 918 923
936 919 924
937 920 925
938 921 926
939 922 927
940 923 928
941 924 929
942 925 930
943 926 931
944 927 932
945 928 933
946 929 934
947 930 935
948 931 936
949 932 937
950 933 938
951 934 939
952 935 940
953 936 941
954 937 942
955 938 943
956 939 944
957 940 945
958 941 946
959 942 947
960 943 948
961 944 949
962 945 950
963 946 951
964 947 952
965 948 953
966 949 954
967 950 955
968 951 956
969 952 957
970 953 958
971 954 959
972 955 960
973 956 961
974 957 962
975 958 963
976 959 964
977 960 965
978 961 966
979 962 967
980 963 968
981 964 969
982 965 970
983 966 971
984 967 972
985 968 973
986 969 974
987 970 975
988 971 976
989 972 977
990 973 978
991 974 979
992 975 980
993 976 981
994 977 982
995 978 983
996 979 984
997 980 985
998 981 986
999 982 987
1000 983 988

```

Lib/tarfile.py

```

@@ -68,7 +68,7 @@
68 68         "DEFAULT_FORMAT", "open", "fully_trusted_filter", "data_filter",
69 69         "tar_filter", "FilterError", "AbsoluteLinkError",
70 70         "OutsideDestinationError", "SpecialFileError",
71 71         "AbsolutePathError",
72 72         "LinkOutsideDestinationError"]
73 73
74 74         #-----

```

	⬆️	@@ -755,10 +755,22 @@ def __init__(self, tarinfo, path):
755	755	super().__init__(f'{tarinfo.name!r} would link to {path!r}, '
756	756	+ 'which is outside the destination')
757	757	
758		+ class LinkFallbackError(FilterError):
759		+ def __init__(self, tarinfo, path):
760		+ self.tarinfo = tarinfo
761		+ self._path = path
762		+ super().__init__(f'link {tarinfo.name!r} would be extracted as a '
763		+ + f'copy of {path!r}, which was rejected')
764		+
765		+ # Errors caused by filters -- both "fatal" and "non-fatal" -- that
766		+ # we consider to be issues with the argument, rather than a bug in the
767		+ # filter function
768		+ _FILTER_ERRORS = (FilterError, OSError, ExtractError)
769		+
758	770	def _get_filtered_attrs(member, dest_path, for_data=True):
759	771	new_attrs = {}
760	772	name = member.name
761		- dest_path = os.path.realpath(dest_path)
773		+ dest_path = os.path.realpath(dest_path, strict=os.path.ALLOW_MISSING)
762	774	# Strip leading / (tar's directory separator) from filenames.
763	775	# Include os.sep (target OS directory separator) as well.
764	776	if name.startswith('/'): os.sep):
		@@ -768,7 +780,8 @@ def _get_filtered_attrs(member, dest_path,
		for_data=True):
768	780	# For example, 'C:/foo' on Windows.
769	781	raise AbsolutePathError(member)
770	782	# Ensure we stay in the destination
771		- target_path = os.path.realpath(os.path.join(dest_path, name))
783		+ target_path = os.path.realpath(os.path.join(dest_path, name),
784		+ strict=os.path.ALLOW_MISSING)
772	785	if os.path.commonpath([target_path, dest_path]) != dest_path:
773	786	raise OutsideDestinationError(member, target_path)
774	787	# Limit permissions (no high bits, and go-w)
		@@ -806,14 +819,18 @@ def _get_filtered_attrs(member, dest_path,
		for_data=True):
806	819	if member.islnk() or member.issym():
807	820	if os.path.isabs(member.linkname):
808	821	raise AbsoluteLinkError(member)

822	+	<code>normalized = os.path.normpath(member.linkname)</code>
823	+	<code>if normalized != member.linkname:</code>
824	+	<code>new_attrs['linkname'] = normalized</code>
809	825	<code>if member.issym():</code>
810	826	<code>target_path = os.path.join(dest_path,</code>
811	827	<code>os.path.dirname(name),</code>
812	828	<code>member.linkname)</code>
813	829	<code>else:</code>
814	830	<code>target_path = os.path.join(dest_path,</code>
815	831	<code>member.linkname)</code>
816	-	<code>target_path = os.path.realpath(target_path)</code>
832	+	<code>target_path = os.path.realpath(target_path,</code>
833	+	<code>strict=os.path.ALLOW_MISSING)</code>
817	834	<code>if os.path.commonpath([target_path, dest_path]) != dest_path:</code>
818	835	<code>raise LinkOutsideDestinationError(member, target_path)</code>
819	836	<code>return new_attrs</code>
⌵		<code>@@ -2323,30 +2340,58 @@ def extractall(self, path=".", members=None, *,</code>
⌶		<code>numeric_owner=False,</code>
2323	2340	<code>members = self</code>
2324	2341	
2325	2342	<code>for member in members:</code>
2326	-	<code>tarinfo = self._get_extract_tarinfo(member, filter_function,</code>
		<code>path)</code>
2343	+	<code>tarinfo, unfiltered = self._get_extract_tarinfo(</code>
2344	+	<code>member, filter_function, path)</code>
2327	2345	<code>if tarinfo is None:</code>
2328	2346	<code>continue</code>
2329	2347	<code>if tarinfo.isdir():</code>
2330	2348	<code># For directories, delay setting attributes until later,</code>
2331	2349	<code># since permissions can interfere with extraction and</code>
2332	2350	<code># extracting contents can reset mtime.</code>
2333	-	<code>directories.append(tarinfo)</code>
2351	+	<code>directories.append(unfiltered)</code>
2334	2352	<code>self._extract_one(tarinfo, path, set_attrs=not tarinfo.isdir(),</code>
2335	-	<code>numeric_owner=numeric_owner)</code>
2353	+	<code>numeric_owner=numeric_owner,</code>
2354	+	<code>filter_function=filter_function)</code>
2336	2355	
2337	2356	<code># Reverse sort directories.</code>
2338	2357	<code>directories.sort(key=lambda a: a.name, reverse=True)</code>

```

2339 2358
2359 +
2340 2360 # Set correct owner, mtime and filemode on directories.
2341 - for tarinfo in directories:
2342 -     dirpath = os.path.join(path, tarinfo.name)
2361 + for unfiltered in directories:
2343 2362     try:
2363 +         # Need to re-apply any filter, to take the *current*
filesystem
2364 +         # state into account.
2365 +         try:
2366 +             tarinfo = filter_function(unfiltered, path)
2367 +         except _FILTER_ERRORS as exc:
2368 +             self._log_no_directory_fixup(unfiltered, repr(exc))
2369 +             continue
2370 +         if tarinfo is None:
2371 +             self._log_no_directory_fixup(unfiltered,
2372 +                                         'excluded by filter')
2373 +             continue
2374 +         dirpath = os.path.join(path, tarinfo.name)
2375 +         try:
2376 +             lstat = os.lstat(dirpath)
2377 +         except FileNotFoundError:
2378 +             self._log_no_directory_fixup(tarinfo, 'missing')
2379 +             continue
2380 +         if not stat.S_ISDIR(lstat.st_mode):
2381 +             # This is no longer a directory; presumably a later
2382 +             # member overwrote the entry.
2383 +             self._log_no_directory_fixup(tarinfo, 'not a directory')
2384 +             continue
2344 2385         self.chown(tarinfo, dirpath, numeric_owner=numeric_owner)
2345 2386         self.utime(tarinfo, dirpath)
2346 2387         self.chmod(tarinfo, dirpath)
2347 2388     except ExtractError as e:
2348 2389         self._handle_nonfatal_error(e)
2349 2390
2391 + def _log_no_directory_fixup(self, member, reason):
2392 +     self._dbg(2, "tarfile: Not fixing up directory %r (%s)" %
2393 +               (member.name, reason))
2394 +

```

```

2350 2395         def extract(self, member, path="", set_attrs=True, *,
numeric_owner=False,
2351 2396             filter=None):
2352 2397         """Extract a member from the archive to the current working
directory,
@@ -2362,41 +2407,56 @@ def extract(self, member, path="",
set_attrs=True, *, numeric_owner=False,
2362 2407             String names of common filters are accepted.
2363 2408         """
2364 2409         filter_function = self._get_filter_function(filter)
2365 2410         - tarinfo = self._get_extract_tarinfo(member, filter_function, path)
2410 2411         + tarinfo, unfiltered = self._get_extract_tarinfo(
2411 2412             member, filter_function, path)
2366 2412         if tarinfo is not None:
2367 2413             self._extract_one(tarinfo, path, set_attrs, numeric_owner)
2368 2414
2369 2415         def _get_extract_tarinfo(self, member, filter_function, path):
2370 2416         - """Get filtered TarInfo (or None) from member, which might be a
str"""
2416 2417         + """Get (filtered, unfiltered) TarInfos from *member*
2417 2418         +
2418 2419         + *member* might be a string.
2419 2420         +
2420 2421         + Return (None, None) if not found.
2421 2422         + """
2371 2423         if isinstance(member, str):
2372 2424         - tarinfo = self.getmember(member)
2424 2425         + unfiltered = self.getmember(member)
2373 2425         else:
2374 2426         - tarinfo = member
2426 2427         + unfiltered = member
2375 2427
2376 2428         - unfiltered = tarinfo
2428 2429         + filtered = None
2377 2429         try:
2378 2430         - tarinfo = filter_function(tarinfo, path)
2430 2431         + filtered = filter_function(unfiltered, path)
2379 2431         except (OSError, UnicodeEncodeError, FilterError) as e:
2380 2432             self._handle_fatal_error(e)

```

```

2381 2433         except ExtractError as e:
2382 2434             self._handle_nonfatal_error(e)
2383 -         if tarinfo is None:
2435 +         if filtered is None:
2384 2436             self._dbg(2, "tarfile: Excluded %r" % unfiltered.name)
2385 -         return None
2437 +         return None, None
2438 +
2386 2439         # Prepare the link target for makelink().
2387 -         if tarinfo.islnk():
2388 -             tarinfo = copy.copy(tarinfo)
2389 -             tarinfo._link_target = os.path.join(path, tarinfo.linkname)
2390 -         return tarinfo
2440 +         if filtered.islnk():
2441 +             filtered = copy.copy(filtered)
2442 +             filtered._link_target = os.path.join(path, filtered.linkname)
2443 +         return filtered, unfiltered
2391 2444
2392 -         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner):
2393 -             """Extract from filtered tarinfo to disk"""
2445 +         def _extract_one(self, tarinfo, path, set_attrs, numeric_owner,
2446 +                         filter_function=None):
2447 +             """Extract from filtered tarinfo to disk.
2448 +
2449 +             filter_function is only used when extracting a *different*
2450 +             member (e.g. as fallback to creating a symlink)
2451 +             """
2394 2452         self._check("r")
2395 2453
2396 2454         try:
2397 2455             self._extract_member(tarinfo, os.path.join(path, tarinfo.name),
2398 2456                                 set_attrs=set_attrs,
2399 -                                 numeric_owner=numeric_owner)
2457 +                                 numeric_owner=numeric_owner,
2458 +                                 filter_function=filter_function,
2459 +                                 extraction_root=path)
2400 2460         except (OSError, UnicodeEncodeError) as e:
2401 2461             self._handle_fatal_error(e)
2402 2462         except ExtractError as e:

```



```
@@ -2454,9 +2514,13 @@ def extractfile(self, member):
```

		↑	
2454	2514		return None
2455	2515		
2456	2516		def _extract_member(self, tarinfo, targetpath, set_attrs=True,
2457	-		numeric_owner=False):
2458	-		"""Extract the TarInfo object tarinfo to a physical
	2517	+	numeric_owner=False, *, filter_function=None,
	2518	+	extraction_root=None):
	2519	+	"""Extract the filtered TarInfo object tarinfo to a physical
2459	2520		file called targetpath.
	2521	+	
	2522	+	filter_function is only used when extracting a *different*
	2523	+	member (e.g. as fallback to creating a symlink)
2460	2524		"""
2461	2525		# Fetch the TarInfo object for the given name
2462	2526		# and build the destination pathname, replacing
		↓	@@ -2485,7 +2549,10 @@ def _extract_member(self, tarinfo, targetpath,
		↑	set_attrs=True,
2485	2549		elif tarinfo.ischr() or tarinfo.isblk():
2486	2550		self.makedev(tarinfo, targetpath)
2487	2551		elif tarinfo.islnk() or tarinfo.issym():
2488	-		self.makelink(tarinfo, targetpath)
	2552	+	self.makelink_with_filter(
	2553	+	tarinfo, targetpath,
	2554	+	filter_function=filter_function,
	2555	+	extraction_root=extraction_root)
2489	2556		elif tarinfo.type not in SUPPORTED_TYPES:
2490	2557		self.makeunknown(tarinfo, targetpath)
2491	2558		else:
		↓	@@ -2568,29 +2635,57 @@ def makedev(self, tarinfo, targetpath):
		↑	
2568	2635		os.makedev(tarinfo.devmajor, tarinfo.devminor))
2569	2636		
2570	2637		def makelink(self, tarinfo, targetpath):
	2638	+	return self.makelink_with_filter(tarinfo, targetpath, None, None)
	2639	+	
	2640	+	def makelink_with_filter(self, tarinfo, targetpath,
	2641	+	filter_function, extraction_root):
2571	2642		"""Make a (symbolic) link called targetpath. If it cannot be created
2572	2643		(platform limitation), we try to make a copy of the referenced file

```
2573 2644         instead of a link.
2645 +
2646 +         filter_function is only used when extracting a *different*
2647 +         member (e.g. as fallback to creating a link).
2574 2648         """
2649 +         keyerror_to_extractorerror = False
2575 2650     try:
2576 2651         # For systems that support symbolic and hard links.
2577 2652         if tarinfo.issym():
2578 2653             if os.path.lexists(targetpath):
2579 2654                 # Avoid FileExistsError on following os.symlink.
2580 2655                 os.unlink(targetpath)
2581 2656                 os.symlink(tarinfo.linkname, targetpath)
2657 +                 return
2582 2658         else:
2583 2659             if os.path.exists(tarinfo._link_target):
2584 2660                 os.link(tarinfo._link_target, targetpath)
2585 -             else:
2586 -                 self._extract_member(self._find_link_target(tarinfo),
2587 -                                     targetpath)
2661 +                 return
2588 2662     except symlink_exception:
2663 +         keyerror_to_extractorerror = True
2664 +
2665 +     try:
2666 +         unfiltered = self._find_link_target(tarinfo)
2667 +     except KeyError:
2668 +         if keyerror_to_extractorerror:
2669 +             raise ExtractError(
2670 +                 "unable to resolve link inside archive") from None
2671 +         else:
2672 +             raise
2673 +
2674 +     if filter_function is None:
2675 +         filtered = unfiltered
2676 +     else:
2677 +         if extraction_root is None:
2678 +             raise ExtractError(
2679 +                 "makelink_with_filter: if filter_function is not None, "
2680 +                 + "extraction_root must also not be None")
```

```
2589 2681         try:
2590         -             self._extract_member(self._find_link_target(tarinfo),
2591         -                                 targetpath)
2592         -         except KeyError:
2593         -             raise ExtractError("unable to resolve link inside archive")
                from None
2682 +             filtered = filter_function(unfiltered, extraction_root)
2683 +         except _FILTER_ERRORS as cause:
2684 +             raise LinkFallbackError(tarinfo, unfiltered.name) from cause
2685 +         if filtered is not None:
2686 +             self._extract_member(filtered, targetpath,
2687 +                                 filter_function=filter_function,
2688 +                                 extraction_root=extraction_root)
2594 2689
2595 2690         def chown(self, tarinfo, targetpath, numeric_owner):
2596 2691             """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 @@
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