python / **cpython** Public

Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

Commit d89a5f6

sethmlarson and **ZeroIntensity** authored on Jan 31, 2025 Verified

```
gh-105704: Disallow square brackets ([ and ]) in domain names for parsed URLs
(#129418)
* gh-105704: Disallow square brackets ( and ) in domain names for parsed URLs

* Use Sphinx references

Co-authored-by: Peter Bierma <zintensitydev@gmail.com>

* Add mismatched bracket test cases, fix news format

* Add more test coverage for ports

---------

Co-authored-by: Peter Bierma <zintensitydev@gmail.com>
```

main (#129418) · v3.15.0a8 ••• v3.14.0a5

1 parent 54f74b8 commit d89a5f6

3 files changed +58 -3 lines changed

Top

Filter files…

- Lib
  - test
    - test_urlparse.py
  - urllib
    - parse.py
- Misc/NEWS.d/next/Security
  - 2025-01-28-14-08-03.gh-issue-105704.EnhHxu.rst

3 files changed +58 -3 lines changed

Search within code

Lib/test/test_urlparse.py

```
@@ -1412,16 +1412,51 @@ def test_invalid_bracketed_hosts(self):
1412 1412          self.assertRaises(ValueError, urllib.parse.urlsplit,
'Scheme://user@[0439:23af::2309::fae7:1234]/Path?Query')
1413 1413          self.assertRaises(ValueError, urllib.parse.urlsplit,
'Scheme://user@[0439:23af:2309::fae7:1234:2342:438e:192.0.2.146]/Path?Query')
1414 1414          self.assertRaises(ValueError, urllib.parse.urlsplit,
'Scheme://user@]v6a.ip[/Path')
     1415 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[v6a.ip]')
     1416 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[v6a.ip].suffix')
     1417 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[v6a.ip]/')
     1418 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[v6a.ip].suffix/')
     1419 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[v6a.ip]?')
     1420 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[v6a.ip].suffix?')
     1421 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[::1]')
     1422 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[::1].suffix')
     1423 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[::1]/')
     1424 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[::1].suffix/')
     1425 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[::1]?')
     1426 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[::1].suffix?')
     1427 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://prefix.[::1]:a')
     1428 +        self.assertRaises(ValueError, urllib.parse.urlsplit,
'scheme://[::1].suffix:a')
```

```
1429 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix.[::1]:a1')
1430 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://[::1].suffix:a1')
1431 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix.[::1]:1a')
1432 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://[::1].suffix:1a')
1433 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix.[::1]:')
1434 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://[::1].suffix:/')
1435 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix.[::1]:?')
1436 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://user@prefix.[v6a.ip]')
1437 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://user@[v6a.ip].suffix')
1438 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://[v6a.ip')
1439 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://v6a.ip]')
1440 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://]v6a.ip[')
1441 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://]v6a.ip')
1442 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://v6a.ip[')
1443 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix.[v6a.ip')
1444 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://v6a.ip].suffix')
1445 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix]v6a.ip[suffix')
1446 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://prefix]v6a.ip')
1447 +         self.assertRaises(ValueError, urllib.parse.urlsplit,
     'scheme://v6a.ip[suffix')
1415 1448
1416 1449         def test_splitting_bracketed_hosts(self):
```

```
1417      -         p1 = urllib.parse.urlsplit('scheme://user@[v6a.ip]/path?query')
     1450 +         p1 = urllib.parse.urlsplit('scheme://user@[v6a.ip]:1234/path?query')
1418 1451           self.assertEqual(p1.hostname, 'v6a.ip')
1419 1452           self.assertEqual(p1.username, 'user')
1420 1453           self.assertEqual(p1.path, '/path')
     1454 +         self.assertEqual(p1.port, 1234)
1421 1455           p2 =
                 urllib.parse.urlsplit('scheme://user@[0439:23af:2309::fae7%test]/path?query')
1422 1456           self.assertEqual(p2.hostname, '0439:23af:2309::fae7%test')
1423 1457           self.assertEqual(p2.username, 'user')
1424 1458           self.assertEqual(p2.path, '/path')
     1459 +         self.assertIs(p2.port, None)
1425 1460           p3 =
                 urllib.parse.urlsplit('scheme://user@[0439:23af:2309::fae7:1234:192.0.2.146%t
                 est]/path?query')
1426 1461           self.assertEqual(p3.hostname,
                 '0439:23af:2309::fae7:1234:192.0.2.146%test')
1427 1462           self.assertEqual(p3.username, 'user')
```

Lib/urllib/parse.py

```
@@ -439,6 +439,23 @@ def _checknetloc(netloc):
439 439              raise ValueError("netloc '" + netloc + "' contains invalid " +
440 440                               "characters under NFKC normalization")
441 441
     442 + def _check_bracketed_netloc(netloc):
     443 +     # Note that this function must mirror the splitting
     444 +     # done in NetlocResultMixins._hostinfo().
     445 +     hostname_and_port = netloc.rpartition('@')[2]
     446 +     before_bracket, have_open_br, bracketed = hostname_and_port.partition('[')
     447 +     if have_open_br:
     448 +         # No data is allowed before a bracket.
     449 +         if before_bracket:
     450 +             raise ValueError("Invalid IPv6 URL")
     451 +         hostname, _, port = bracketed.partition(']')
     452 +         # No data is allowed after the bracket but before the port delimiter.
     453 +         if port and not port.startswith(":"):
     454 +             raise ValueError("Invalid IPv6 URL")
     455 +     else:
     456 +         hostname, _, port = hostname_and_port.partition(':')
```

```
      457 +     _check_bracketed_host(hostname)
      458 +
442   459     # Valid bracketed hosts are defined in
443   460     # https://www.rfc-editor.org/rfc/rfc3986#page-49 and
                https://url.spec.whatwg.org/
444   461     def _check_bracketed_host(hostname):
                @@ -505,8 +522,7 @@ def _urlsplit(url, scheme=None, allow_fragments=True):
505   522                     (']' in netloc and '[' not in netloc)):
506   523                 raise ValueError("Invalid IPv6 URL")
507   524             if '[' in netloc and ']' in netloc:
508         -               bracketed_host = netloc.partition('[')[2].partition(']')[0]
509         -               _check_bracketed_host(bracketed_host)
      525 +               _check_bracketed_netloc(netloc)
510   526         if allow_fragments and '#' in url:
511   527             url, fragment = url.split('#', 1)
512   528         if '?' in url:
```

...5-01-28-14-08-03.gh-issue-105704.EnhHxu.rst

```
      ...   @@ -0,0 +1,4 @@
      1   + When using :func:`urllib.parse.urlsplit` and :func:`urllib.parse.urlparse` host
      2   + parsing would not reject domain names containing square brackets (``[`` and
      3   + ``]``). Square brackets are only valid for IPv6 and IPvFuture hosts according to
      4   + `RFC 3986 Section 3.2.2 <https://www.rfc-editor.org/rfc/rfc3986#section-
            3.2.2>`__.
```

Comments 0