

 aio-lib/aiohttp Public

[Code](#) [Issues](#) 185 [Pull requests](#) 90 [Discussions](#) [Actions](#) [Security and](#)

Commit 8a74257



 **Dreamsorcerer** authored 3 weeks ago · ✓ 34 / 38 · Verified

Restrict multipart header sizes (#12208) (#12228)











(cherry picked from commit [5fe9dfb](#))

 master (#12228) ·  v3.13.5 v3.13.4

1 parent [53b35a2](#) commit 8a74257 

 **8 files changed** +109 -20 lines changed

[↑ Top](#) 

- ▼  aiohttp
 -  multipart.py
 -  streams.py
 -  test_utils.py
 -  web_protocol.py
 -  web_request.py
- ▼  tests
 -  test_multipart.py
 -  test_streams.py
 -  test_web_request.py

 **8 files changed** +109 -20 lines changed



▼ aiohttp/multipart.py ⋮

 @@ -41,6 +41,7 @@

41 41)

```

42 42     from .helpers import CHAR, TOKEN, parse_mimetype, reify
43 43     from .http import HeadersParser
44 44 +   from .http_exceptions import BadHttpMessage
44 45     from .log import internal_logger
45 46     from .payload import (
46 47         JsonPayload,
@@ -658,7 +659,14 @@ class MultipartReader:
658 659         #: Body part reader class for non multipart/* content types.
659 660         part_reader_cls = BodyPartReader
660 661
661 661 -     def __init__(self, headers: Mapping[str, str], content: StreamReader) ->
        None:
662 662 +     def __init__(
663 663 +         self,
664 664 +         headers: Mapping[str, str],
665 665 +         content: StreamReader,
666 666 +         *,
667 667 +         max_field_size: int = 8190,
668 668 +         max_headers: int = 128,
669 669 +     ) -> None:
662 670         self._mimetype = parse_mimetype(headers[CONTENT_TYPE])
663 671         assert self._mimetype.type == "multipart", "multipart/* content type
        expected"
664 672         if "boundary" not in self._mimetype.parameters:
@@ -669,8 +677,10 @@ def __init__(self, headers: Mapping[str, str], content:
        StreamReader) -> None:
669 677         self.headers = headers
670 678         self._boundary = ("--" + self._get_boundary()).encode()
671 679         self._content = content
672 672 -         self._default_charset: Optional[str] = None
673 673 -         self._last_part: Optional[Union["MultipartReader", BodyPartReader]] =
        None
680 680 +         self._default_charset: str | None = None
681 681 +         self._last_part: MultipartReader | BodyPartReader | None = None
682 682 +         self._max_field_size = max_field_size
683 683 +         self._max_headers = max_headers
674 684         self._at_eof = False
675 685         self._at_bof = True
676 686         self._unread: List[bytes] = []

```

```

@@ -770,7 +780,12 @@ def _get_part_reader(
770 780         if mimetype.type == "multipart":
771 781             if self.multipart_reader_cls is None:
772 782                 return type(self)(headers, self._content)
773 -                 return self.multipart_reader_cls(headers, self._content)
783 +                 return self.multipart_reader_cls(
784 +                     headers,
785 +                     self._content,
786 +                     max_field_size=self._max_field_size,
787 +                     max_headers=self._max_headers,
788 +                 )
774 789         else:
775 790             return self.part_reader_cls(
776 791                 self._boundary,
@@ -832,12 +847,14 @@ async def _read_boundary(self) -> None:
832 847         async def _read_headers(self) -> "CIMultiDictProxy[str]":
833 848             lines = []
834 849             while True:
835 -                 chunk = await self._content.readline()
850 +                 chunk = await
self._content.readline(max_line_length=self._max_field_size)
836 851                 chunk = chunk.rstrip(b"\r\n")
837 852                 lines.append(chunk)
838 853                 if not chunk:
839 854                     break
840 -                 parser = HeadersParser()
855 +                 if len(lines) > self._max_headers:
856 +                     raise BadHttpMessage("Too many headers received")
857 +                 parser = HeadersParser(max_field_size=self._max_field_size)
841 858                 headers, raw_headers = parser.parse_headers(lines)
842 859                 return headers
843 860

```

▼ aiohttp/streams.py

...

```

@@ -21,6 +21,7 @@
21 21         set_exception,
22 22         set_result,

```

```

23 23 )
24 + from .http_exceptions import LineTooLong
24 25 from .log import internal_logger
25 26
26 27 __all__ = (
@@ -372,10 +373,12 @@ async def _wait(self, func_name: str) -> None:
372 373         finally:
373 374             self._waiter = None
374 375
375 -     async def readline(self) -> bytes:
376 -         return await self.readuntil()
376 +     async def readline(self, *, max_line_length: Optional[int] = None) ->
bytes:
377 +         return await self.readuntil(max_size=max_line_length)
377 378
378 -     async def readuntil(self, separator: bytes = b"\n") -> bytes:
379 +     async def readuntil(
380 +         self, separator: bytes = b"\n", *, max_size: Optional[int] = None
381 +     ) -> bytes:
379 382         seplen = len(separator)
380 383         if seplen == 0:
381 384             raise ValueError("Separator should be at least one-byte string")
@@ -386,6 +389,7 @@ async def readuntil(self, separator: bytes = b"\n") ->
bytes:
386 389         chunk = b""
387 390         chunk_size = 0
388 391         not_enough = True
392 +         max_size = max_size or self._high_water
389 393
390 394         while not_enough:
391 395             while self._buffer and not_enough:
@@ -400,8 +404,8 @@ async def readuntil(self, separator: bytes = b"\n") ->
bytes:
400 404                 if ichar:
401 405                     not_enough = False
402 406
403 -                 if chunk_size > self._high_water:
404 -                     raise ValueError("Chunk too big")
407 +                 if chunk_size > max_size:

```

```

408 + raise LineTooLong(chunk[:100] + b"...", max_size)
405 409
406 410         if self._eof:
407 411             break
    ↓
    ↑
@@ -622,7 +626,7 @@ async def wait_eof(self) -> None:
622 626         def feed_data(self, data: bytes, n: int = 0) -> None:
623 627             pass
624 628
625 - async def readline(self) -> bytes:
629 + async def readline(self, *, max_line_length: Optional[int] = None) ->
    bytes:
626 630             return b""
627 631
628 632         async def read(self, n: int = -1) -> bytes:
    ↓

```

```

  aiohttp/test_utils.py
    ↑
@@ -729,6 +729,9 @@ def make_mocked_request(
729 729
730 730         if protocol is sentinel:
731 731             protocol = mock.Mock()
732 +         protocol.max_field_size = 8190
733 +         protocol.max_line_length = 8190
734 +         protocol.max_headers = 128
732 735         protocol.transport = transport
733 736         type(protocol).peername = mock.PropertyMock(
734 737             return_value=transport.get_extra_info("peername")
    ↓

```

```

  aiohttp/web_protocol.py
    ↑
@@ -142,6 +142,9 @@ class RequestHandler(BaseProtocol):
142 142         """
143 143
144 144         __slots__ = (
145 +         "max_field_size",
146 +         "max_headers",
147 +         "max_line_size",
145 148         "_request_count",

```

```

146 149     "_keepalive",
147 150     "_manager",
    ↓
@@ -205,6 +208,10 @@ def __init__(
    ↑
205 208         self._request_handler: Optional[_RequestHandler] =
            manager.request_handler
206 209         self._request_factory: Optional[_RequestFactory] =
            manager.request_factory
207 210
211 +         self.max_line_size = max_line_size
212 +         self.max_headers = max_headers
213 +         self.max_field_size = max_field_size
214 +
208 215         self._tcp_keepalive = tcp_keepalive
209 216         # placeholder to be replaced on keepalive timeout setup
210 217         self._next_keepalive_close_time = 0.0
    ↓

```

▼ aiohttp/web_request.py



```

    ↑
@@ -696,7 +696,12 @@ async def json(self, *, loads: JSONDecoder =
    DEFAULT_JSON_DECODER) -> Any:
696 696
697 697         async def multipart(self) -> MultipartReader:
698 698             """Return async iterator to process BODY as multipart."""
699 -         return MultipartReader(self._headers, self._payload)
700 +         return MultipartReader(
701 +             self._headers,
702 +             self._payload,
703 +             max_field_size=self._protocol.max_field_size,
704 +             max_headers=self._protocol.max_headers,
705 +         )
700 705
701 706         async def post(self) -> "MultiDictProxy[Union[str, bytes, FileField]]":
702 707             """Return POST parameters."""
    ↓

```

▼ tests/test_multipart.py



```

    ↑
@@ -3,6 +3,7 @@
3 3     import json

```

```

4     4     import pathlib
5     5     import sys
6     6     + from typing import Optional
7     7     from unittest import mock
8     8
9     9     import pytest
@@ -85,7 +86,7 @@
@@ -85,7 +86,7 @@ async def read(self, size=None):
85    86         def at_eof(self):
86    87             return self.content.tell() == len(self.content.getbuffer())
87    88
88    88         - async def readline(self):
89    89         + async def readline(self, *, max_line_length: Optional[int] = None) ->
90    90             bytes:
91    91             return self.content.readline()
92    92
93    93         def unread_data(self, data):
@@ -856,7 +857,7 @@
@@ -856,7 +857,7 @@ async def read(self, size=None) -> bytes:
856   857         def at_eof(self) -> bool:
857   858             return not self.content
858   859
859   859         - async def readline(self) -> bytes:
860   860         + async def readline(self, *, max_line_length: int | None = None) ->
861   861             bytes:
862   862                 line = b""
863   863                 while self.content and b"\n" not in line:
864   864                     line += self.content.pop(0)

```

tests/test_streams.py

...

```

@@ -12,6 +12,7 @@
12    12     from re_assert import Matches
13    13
14    14     from aiohttp import streams
15    15     + from aiohttp.http_exceptions import LineTooLong
16    16
17    17     DATA = b"line1\nline2\nline3\n"
18    18

```

⌵	@@ -325,7 +326,7 @@	async def test_readline_limit_with_existing_data(self) -
⌶		> None:
325	326	stream.feed_data(b"li")
326	327	stream.feed_data(b"ne1\nline2\n")
327	328	
328	-	with pytest.raises(ValueError):
329	+	with pytest.raises(LineTooLong):
329	330	await stream.readline()
330	331	# The buffer should contain the remaining data after exception
331	332	stream.feed_eof()
⌵	@@ -346,7 +347,7 @@	def cb():
346	347	
347	348	loop.call_soon(cb)
348	349	
349	-	with pytest.raises(ValueError):
350	+	with pytest.raises(LineTooLong):
350	351	await stream.readline()
351	352	data = await stream.read()
352	353	assert b"chunk3\n" == data
⌵	@@ -436,7 +437,7 @@	async def test_readuntil_limit_with_existing_data(self,
⌶		separator: bytes) -> Non
436	437	stream.feed_data(b"li")
437	438	stream.feed_data(b"ne1" + separator + b"line2" + separator)
438	439	
439	-	with pytest.raises(ValueError):
440	+	with pytest.raises(LineTooLong):
440	441	await stream.readuntil(separator)
441	442	# The buffer should contain the remaining data after exception
442	443	stream.feed_eof()
⌵	@@ -458,7 +459,7 @@	def cb():
458	459	
459	460	loop.call_soon(cb)
460	461	
461	-	with pytest.raises(ValueError, match="Chunk too big"):
462	+	with pytest.raises(LineTooLong):
462	463	await stream.readuntil(separator)
463	464	data = await stream.read()
464	465	assert b"chunk3#" == data
⌵		

```

tests/test_web_request.py
@@ -13,6 +13,7 @@
13 13
14 14     from aiohttp import HttpVersion
15 15     from aiohttp.base_protocol import BaseProtocol
16 + from aiohttp.http_exceptions import BadHttpRequest, LineTooLong
16 17     from aiohttp.http_parser import RawRequestMessage
17 18     from aiohttp.streams import StreamReader
18 19     from aiohttp.test_utils import make_mocked_request
@@ -896,7 +897,57 @@
896 897         result["a_file"].file.close()
897 898
898 899
899 - async def test_make_too_big_request_limit_None(protocol) -> None:
900 + async def test_multipart_formdata_headers_too_many(protocol: BaseProtocol) ->
    None:
901 +     many = b"".join(f"X-{{i}}: a\r\n".encode() for i in range(130))
902 +     body = (
903 +         b"--b\r\n"
904 +         b'Content-Disposition: form-data; name="a"\r\n' + many + b"\r\n1\r\n"
905 +         b"--b--\r\n"
906 +     )
907 +     content_type = "multipart/form-data; boundary=b"
908 +     payload = StreamReader(protocol, 2**16, loop=asyncio.get_running_loop())
909 +     payload.feed_data(body)
910 +     payload.feed_eof()
911 +     req = make_mocked_request(
912 +         "POST",
913 +         "/",
914 +         headers={"CONTENT-TYPE": content_type},
915 +         payload=payload,
916 +     )
917 +
918 +     with pytest.raises(BadHttpRequest, match="Too many headers received"):
919 +         await req.post()
920 +
921 +

```

```
922 + async def test_multipart_formdata_header_too_long(protocol: BaseProtocol) ->
    None:
923 +     k = b"t" * 4100
924 +     body = (
925 +         b"--b\r\n"
926 +         b'Content-Disposition: form-data; name="a"\r\n'
927 +         + k
928 +         + b": "
929 +         + k
930 +         + b"\r\n"
931 +         + b"\r\n1\r\n"
932 +         b"--b--\r\n"
933 +     )
934 +     content_type = "multipart/form-data; boundary=b"
935 +     payload = StreamReader(protocol, 2**16, loop=asyncio.get_running_loop())
936 +     payload.feed_data(body)
937 +     payload.feed_eof()
938 +     req = make_mocked_request(
939 +         "POST",
940 +         "/",
941 +         headers={"CONTENT-TYPE": content_type},
942 +         payload=payload,
943 +     )
944 +
945 +     match = "400, message:\n Got more than 8190 bytes when reading"
946 +     with pytest.raises(LineTooLong, match=match):
947 +         await req.post()
948 +
949 +
950 + async def test_make_too_big_request_limit_None(protocol: BaseProtocol) -> None:
900 951     payload = StreamReader(protocol, 2**16, loop=asyncio.get_event_loop())
901 952     large_file = 1024**2 * b"x"
902 953     too_large_file = large_file + b"x"
```



Comments 0



Please [sign in](#) to comment.