4 Undici does not use CONNECT or otherwise validate upstream HTTPS certificates when using a proxy

Share:

TIMELINE

jimterry submitted a report to Node.js. May 27th (6 months ago)

Summary: When using Undici with its ProxyAgent, it does not use CONNECT or correctly verify the upstream server's HTTPS certificate.

Description:

This affects both Undici itself and global fetch() in Node 18 when used with Undici's ProxyAgent. I've submitted this here for Node as it affects global fetch, and Undici isn't listed in the options (even though Undici's SECURITY.md says to report issues the same way as Node issues: https://github.com/nodejs/undici/blob/main/SECURITY.md)

Some context is required to explain the issue so this is quite long, sorry!

In general, given a setup like:

Undici client -> proxy.example -> remote-server.example

There's two possible ways to use the proxy server: you can send CONNECT to create a tunnel to the remote server, and then make a request within that tunnel (setting up TLS in the tunnel first, if making an HTTPS request) or you can make a request with an absolute URL like "GET http://remote-server.example/abc" to the proxy server, and expect that proxy to connect upstream and make the request for you.

The former CONNECT approach is far more common. Using the latter form is rare for plain HTTP and should *never* be used for HTTPS: it exposes all request & response data to the proxy, delegates all upstream certificate trust handling to the proxy, and additionally if the connection to the proxy is plain HTTP then it unwraps and exposes all HTTPS traffic on the network between the proxy & client, sending all HTTPS data in plain text.

common today, even in the modern HTTPS-only world, just providing dumb tunnels that are secured independently.

To make this work, what should normally happen when proxying HTTPS traffic to remote-server.example via proxy.example is:

- Undici connects to proxy.example (via HTTP or HTTPS - depends on the proxy setup, but either is basically fine)
- Undici sends "CONNECT remote-server.example:443" to the proxy server
- The proxy server connects to that address
- The proxy server responds to Undici with 200 OK, and then all future bytes are passed raw between Undici & the remote server
- Undici does normal TLS setup through this tunnel with the remote server, validating the certificate as normal.
- Once TLS is set up, Undici makes an HTTP request inside the TLS connection, inside the proxy tunnel.

This means that nobody on the network path between Undici and the remote server (including the proxy, and anybody else en route) can see or modify the HTTP request or response. https://en.wikipedia.org/wiki/HTTP_tunnel#HTTP_CONNECT_method has more details, as does the RFC: https://datatracker.ietf.org/doc/html/rfc7231#section-4.3.6.

That's how it should work. In practice, Undici's implementation (https://github.com/nodejs/undici/blob/main/lib/proxy-agent.js) appears to just send "GET https://remote-server.example" to the proxy (i.e. it just makes all URLs absolute, and then redirects the connection to the proxy instead of the real server).

This means Undici never verifies the remote server's certificate itself, and always exposes all request & response data to the proxy. This unexpectedly means that proxies can MitM all HTTPS traffic, and if the proxy's URL is HTTP (http://proxy.example) then it also means that nominally HTTPS requests are actually sent via in plain-text HTTP between Undici and the proxy server.

This also creates other major bugs too, for example when connecting to a proxy via HTTPS, it seems like the proxy's certificate is verified against the remote server's domain name, not the proxy's domain name, which makes HTTPS proxying unusable in most (all?) cases. This is a related bug which does not expose users to danger (it just breaks things) but I suspect this

Steps To Reproduce:

1. Use any proxy that supports HTTPS upstream connections and HTTP downstream connections. For a quick test, you can use https://hub.docker.com/r/vimagick/privoxy/ with Docker by running `docker run --rm -it -p 8118:8118 vimagick/privoxy:latest` to start an HTTP proxy on localhost:8118.
2. Then make a request to a HTTPS site with an invalid certificate (e.g. https://self-signed.badssl.com/) using Undici with this proxy , like so:

Code 199 Bytes Wrap lines Copy Download

```
1  const undici = require('undici')
2  const dispatcher = new undici.ProxyAgent({ uri: "http://localhost:8118" })
3  console.log((await undici.fetch("https://self-signed.badssl.com", { dispatcher })).st
```

3. The request should fail. The upstream certificate is self signed and completely invalid. Instead it succeeds and prints 200.

This works in Node 16.14.2 using Undici 5.3.0, and in Node 18.2.0 using Undici 5.3.0 or the built-in `fetch()` method. AFAICT this affects all versions of both. This works for all badssl.com test sites that should fail, including expired certificates, and certificates with the wrong hostname.

You can confirm that this should be rejected by removing the `{ dispatcher }` option. Sending the request directly without the proxy will correctly throw a `Error: self-signed certificate` error.

This is not really related to the proxy configuration. The proxy here could verify the upstream certificate and it doesn't, but in my quick bit of testing for this issue it appears that no proxies verify upstream certificates for you because nobody should ever be sending HTTPS traffic in plaintext through a proxy like this. Some proxies disallow non-CONNECT connections entirely, which avoids this issue, but that means they are totally unusable with Undici's ProxyAgent in all cases.

HTTPS clients using proxies should always open a direct tunnel to the remote server via CONNECT, and then verify an end-to-end TLS connection on top of that as normal.

The above reproduces the main "HTTPS via HTTP proxy is not secure" bug. To reproduce the related bug, where HTTPS certificates with HTTPS proxies is not validated correctly and so unusable:

3. Enter 'passphrase' as the passphrase and 'localhost' as the common name
4. Start an HTTPS proxy using this cert by running:

Code 244 Bytes Wrap lines Copy Download

```
const https = require('https');
const proxy = require('proxy');
const fs = require('fs');

proxy(https.createServer({
    key: fs.readFileSync('./key.pem'),
    passphrase: 'passphrase',
    cert: fs.readFileSync('./cert.pem')
})).listen(8443);
```

5. In a new terminal in the same directory, run `export NODE_EXTRA_CA_CERTS=$(pwd)/cert.pem` to trust the proxy's certificate.
6. In another node process in that terminal, use this proxy from Undici:

Code 219 Bytes Wrap lines Copy Download

```
const undici = require('undici')
const dispatcher = new undici.ProxyAgent({ uri: "https://localhost:443" }); // HTTPS
console.log((await undici.fetch("https://example.com", { dispatcher })).status);
```

7. This throws "Error [ERR_TLS_CERT_ALTNAME_INVALID]: Hostname/IP does not match certificate's altnames: Host: example.com. is not cert's CN: localhost".

This is incorrect validation, because the 'localhost' certificate is the certificate of the proxy, not the remote server. Since that certificate is trusted, it should be acceptable for the connection to the localhost proxy, and the server's certificate should be retrieved via a CONNECT tunnel and validated separately. All together, this makes HTTPS proxies unusable with Undici.

Impact

This very seriously affects all use of HTTPS via a HTTP proxy with Undici or Node's global fetch. In this case, it removes all HTTPS security from all requests sent using Undici's ProxyAgent, allowing trivial MitM attacks by anybody on the network path between the client and the target server (local network users, your ISP, the proxy, the target server's ISP, etc).

This less seriously affects HTTPS via HTTPS proxies, but it's still bad. When you send HTTPS via a proxy to a remote server, the proxy can freely view or modify all HTTPS traffic unexpectedly (but only the proxy - generally not anybody else on the network path). This is mitigated by this use case being entirely broken in Undici right now though AFAICT, since the proxy's HTTPS certificate is never validated correctly and so is always rejected. On the other hand, that does mean all proxy users must be using plain-text HTTP, which is seriously impacted by this issue.

mcollina Node.js staff posted a comment. May 27th (6 months ago)

Thanks for reporting. This is undici specific as Node.js does not ship with the ProxyAgent implementation (yet) and it's the right kind of feedback that we are aiming to receive. Keep it up! We'll investigate this shortly.

mcollina Node.js staff changed the status to ○ **Triaged**. Jun 5th (6 months ago)

We have a fix ready to be released. Can you confirm your github handle so we can add you to the reporrt?

pimterry posted a comment. Jun 7th (6 months ago)

Sorry, I've been away over the weekend. Back now - I've seen the PR & notification, I am indeed @pimterry on GitHub as you've guessed, and I'll review that PR today.

mcollina Node.js staff updated CVE reference to **CVE-2022-32210**. Jun 13th (6 months ago)

mcollina Node.js staff closed the report and changed the status to ○ **Resolved**. Jun 13th (6 months ago)

Fixed in v5.5.1

mcollina Node.js staff requested to disclose this report. Jun 13th (6 months ago)

mcollina Node.js staff posted a comment. Jun 13th (6 months ago)

You can request a bounty at https://hackerone.com/ibb. Feel free to include my contacts. From my point of view it should be eligible.

pimterry posted a comment. Jun 13th (6 months ago)

Great, will do! Thanks for the quick work on getting this fixed 👍

This report has been disclosed. Jul 13th (5 months ago)