20 **Built-in TLS module unexpectedly treats "rejectUnauthorized: undefined" as "rejectUnauthorized: false", disabling all certificate validation**

Share:

TIMELINE

pimterry submitted a report to Node.js. Jul 26th (about 1 year ago)

> NOTE! Thanks for submitting a report! Please replace *all* the [square] sections below with the pertinent details. Remember, the more detail you provide, the easier it is for us to triage and respond quickly, so be sure to take your time filling out the report!

Summary: "rejectUnauthorized: false" disables all TLS validation, and should not be set in almost all circumstances. The documentation says only the specific 'false' value will disable this validation, but in fact a 'undefined' value does also disables it, unexpectedly disabling TLS entirely.

Description:

The documentation for tls.connect (https://nodejs.org/api/tls.html#tls_tls_connect_options_callback) says:

> rejectUnauthorized `<boolean>` If not false, the server certificate is verified against the list of supplied CAs. An 'error' event is emitted if verification fails; err.code contains the OpenSSL error code. Default: true.

This is not true (see repro below) - in addition to `false`, an explicit `undefined` value does also disable server certificate verification.

This is very problematic, because it's reasonable to assume that `undefined` will be equivalent to setting the default, and it's also easy to accidentally produce undefined fields when dynamically building configuration. In any system that has done so, they are unknowingly silently not validating any TLS connections.

I've discovered this because I've found that https://www.npmjs.com/package/global-agent does exactly this (uses explicit undefineds when building options objects) and it is vulnerable because of this issue (i.e. all users of that package are by default unintentionally not validating TLS certificates for all connections).

This appears to affect all active node versions. Unless I'm missing something, this seems very bad.

Steps To Reproduce:

Repro code:

Code 259 Bytes Wrap lines Copy Download

```
1  const https = require('https');
2  const request = https.get('https://expired.badssl.com', { rejectUnauthorized: undefined });
3  request.on('error', (e) => console.log('Request failed:', e.message));
4  request.on('response', (e) => console.log('Request succeeded'));
```

1. Run the above
2. The request succeeds! It should not, because expired.badssl.com by design has an expired TLS certificate
3. Remove the { rejectUnauthorized: undefined } option, or change it to 'true'
4. It fails, as expected, due to an expired certificate.

Impact:

This breaks all TLS and HTTPS security for anybody who accidentally provides an undefined value, assuming it will be equivalent to providing no value at all.

Impact

Breaks all HTTPS protections, so complete disclosure or trivial manipulation of all HTTPS requests and responses by anybody capable of MITMing the TCP connection.

kumarak39 Node.js staff posted a comment. Updated Jul 27th (about 1 year ago)

Thanks, @pimterry for reporting! It seems `tls.connect` fails to assign the default value to `rejectUnauthorized` options property if it is explicitly set to `undefined`. This causes the request to succeed with authorization error `CERT_HAS_EXPIRED`.

kumarak39 Node.js staff posted a comment. Jul 27th (about 1 year ago)

I have limited experience with JS. Is it a good practice to explicitly assign `undefined` to the property? The assignment operation fails to mutate the property because it is explicitly assigned. https://github.com/nodejs/node/blob/master/lib/_tls_wrap.js#L1595

pimterry posted a comment. Jul 28th (about 1 year ago)

> I have limited experience with JS. Is it a good practice to explicitly assign undefined to the property?

It's certainly very common practice, I think it's the standard way to write code in most relevant cases. It's messy to build an object literal with properties that may or may not be included, so the normal approach is to instead include every property that's might be included, but set irrelevant values to undefined. I would expect almost all JS devs to assume that the Node APIs will treat undefined the same as providing no value at all. In this case, at least one million-downloads-a-week npm library is doing exactly that when building options for this specific API: https://github.com/gajus/global-agent/blob/3735fa1cac1a6680caa5afd853edba38674d8f27/src/classes/Agent.ts#L169

This is partly codified into JS itself too, e.g. JavaScript's built-in syntax for default parameters treats an explicit undefined the same as not providing a value, and uses the default instead of `undefined`. See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Default_parameters#passing_undefined_vs._other_falsy_values for example.

Also, every other Node.js APIs I've seen does follow this pattern correctly. For example, writable streams have an optional `emitClose` option which defaults to true. If explicitly set to undefined, `true` is correctly used just as if the option was not provided:

Code 87 Bytes Wrap lines Copy Download

```
1  new stream.Writable({ emitClose: undefined })._writableState.emitClose // Prints 'true'
```

just silently makes an insecure request instead.

targos Node.js staff posted a comment. Jul 28th (about 1 year ago)

There are multiple places where we explicitly check for `rejectUnauthorized !== false`.
Example: https://github.com/nodejs/node/blob/3f0b62375b3a4edc8365b803749e8dd5abc706b0/lib/_tls_wrap.js#L1192

Maybe there's one (or more) place where it's not done like this?

Refs: https://github.com/nodejs/node/commit/348cc80a3cbf0f4271ed30418c6ed661bdeede7b

pimterry posted a comment. Jul 28th (about 1 year ago)

The code linked there fixed this only for servers, for incoming client certificate validation. It looks like this was never fixed for outgoing TLS connection verification of server certificates.

I think this is the likely culprit: https://github.com/nodejs/node/blob/master/lib/_tls_wrap.js#L680

kumarak39 Node.js staff posted a comment. Jul 28th (about 1 year ago)

Thanks, @pimterry, and @targos for references.
I think there should be a similar check inside TLS connect function if the `rejectUnauthorized` property fails to initialize with default value because it is explicitly set to `undefined`.

https://github.com/nodejs/node/blob/master/lib/_tls_wrap.js#L1602

Code 66 Bytes Wrap lines Copy Download

```
options.rejectUnauthorized = options.rejectUnauthorized !== false;
```

This will avoid leaking the `undefined` value to the connection handler (`onSecureConnect`).

@pimterry, The authorization error (`CERT_HAS_EXPIRED`) shows up on enabling the debug logs for TLS module.

mcollina Node.js staff changed the status to **Triaged**. Aug 2nd (about 1 year ago)

Confirmed, this should be checked.

mcollina Node.js staff updated the severity from Critical (9.8) to Medium (5.9). Aug 4th (about 1 year ago)

mcollina Node.js staff updated the severity from Medium (5.9) to Low. Aug 4th (about 1 year ago)

mcollina Node.js staff posted a comment. Aug 4th (about 1 year ago)

I have updated the security classification to "low"

pimterry posted a comment. Aug 4th (about 1 year ago)

Can you explain why?

It seems very easy for applications & libraries to hit this issue accidentally (there are popular examples live on npm today, and the docs specifically say that the vulnerable code here is correct). For affected applications, this unexpectedly and silently disables all verification of server's HTTPS certificates, which is a very serious impact, and exploiting this in a vulnerable application is trivial (use any HTTPS certificate you like and it's always trusted).

Just a couple of months ago, the exact same issue (in an npm module, rather than node itself) was reported as 9.4 critical: https://nvd.nist.gov/vuln/detail/CVE-2021-31597.

Similar reports for other tools and vulnerabilities, in generally harder-to-exploit scenarios, all seem to be rated high or above: https://nvd.nist.gov/vuln/detail/CVE-2016-11086, https://nvd.nist.gov/vuln/detail/CVE-2020-11050, https://nvd.nist.gov/vuln/detail/CVE-2021-29504

mcollina Node.js staff posted a comment. Aug 4th (about 1 year ago)

From my point of view this requires quite significant pre-existing permissions to be able to exploit this deliberately. Therefore my CVE assessment is https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:L/AC:H/PR:H/UI:N/S:U/C:L/I:N/A:N&version=3.1.

pimterry posted a comment. Aug 4th (about 1 year ago)

> From my point of view this requires quite significant pre-existing permissions to be able to exploit this deliberately

Maybe there's a misunderstanding then? There are no permissions required to exploit this, as far as I'm aware. Anybody on the local network (same wifi) or remote network path (ISPs) of a vulnerable application can observe and change all HTTPS traffic from vulnerable applications, with no access to the application or device required.

More specifically, "privileges required: high"/PR:H from the CVSS definition (https://www.first.org/cvss/specification-document#2-1-3-Privileges-Required-PR) means "significant (e.g., administrative) control over the vulnerable component". That is *not* required for this vulnerability - you can attack a vulnerable application remotely with no control or access whatsoever to the device. Or is there something I'm missing, that means that that's not true?

Maybe we're conflating privileges and MitM access? That should be under attack complexity separately though. From the spec, a specific example of AC:H is "The attacker must inject themselves into the logical network path between the target and the resource requested by the victim in order to read and/or modify network communications".

Since an attack can arbitrarily change responses, not just observe them, integrity of the "trustworthiness and veracity of information" is very directly impacted too, that should not be 'None'. Availability is a bit less clear, but you can plausibly inject huge responses to clients who trust the server and aren't expecting lots of data to quickly take clients offline.

I guess at the end of the day it's up to you, but imo this seems like a very serious understatement of the significant risk to any impacted users here, and every other similar CVE in the past has considered complete & silent failure of TLS verification to be a high or critical issue.

mhdawson Node.js staff updated CVE reference to CVE-2021-22939. Aug 6th (about 1 year ago)

pimterry posted a comment. Aug 10th (about 1 year ago)

Tim Perry, from HTTP Toolkit. Thanks.

mhdawson Node.js staff posted a comment. Aug 11th (about 1 year ago)

Published: https://nodejs.org/en/blog/vulnerability/aug-2021-security-releases/

mhdawson Node.js staff closed the report and changed the status to **0 Resolved**. Aug 11th (about 1 year ago)

mhdawson Node.js staff requested to disclose this report. Aug 11th (about 1 year ago)

This report has been disclosed. Sep 10th (about 1 year ago)

The Internet Bug Bounty rewarded pimterry with a **$150** bounty. Sep 21st (about 1 year ago)