**How to Resolve Certificate Errors in a Node.js App with SSL Calls**

A practical guide to resolve SSL certificate errors

If you’ve worked on Node/Express App, you may already know that it is pretty straightforward to setup the app with HTTPS and a Server Certificate in non-production environment. In a production environment, NodeJs App typically sits behind a reverse proxy like Nginx which serves certificates. In this case, HTTPS is not required if the App does not need to call other external services.

In reality, your Node app will likely needs to call various back-end services protected by HTTPS. By default, NodeJs is built with [a bundle of commonly used CA root certificates](https://github.com/nodejs/node-v0.x-archive/blob/master/src/node_root_certs.h). Even with these certificates, we still may be bashed with HTTPS related errors if your back-end services are hosted with self-signed certificates (i.e. company-specific private CA).

Some of these errors include:

UNABLE\_TO\_GET\_ISSUER\_CERT\_LOCALLY UNABLE\_TO\_VERIFY\_LEAF\_SIGNATURE DEPTH\_ZERO\_SELF\_SIGNED\_CERT

We need to understand them before they can be resolved. Let’s go deep into these errors.

**SSL Hand-shake**

Diagram

Description automatically generated

These errors originate from the SSL hand-shake process. When a client begins to establish a connection to the server, a TLS handshake happens. It is a series of messages exchanged between the client and server. In these messages, they agree with the version of the TLS and cipher suites to use, verify the identity of the server and generate the session keys.  
At step 2, the server sends a message containing the server’s SSL certificate. Then the client verifies the certificate with the CA (Certificate authority) that issued it. This confirms that the server is who it says it is, and the client is interacting with the actual owner of the domain.

These errors are thrown because the client is not able to verify the self-signed server certificate sent in step 2.

*TLS handshake is a complex process, you may like to read the article below for more details.*

**[Deep Dive Into TLS Handshake](https://levelup.gitconnected.com/deep-dive-into-tls-handshake-e029e28e2eb3" \t "_blank)**

[TLS 1.2 Handshake process explained step by step](https://levelup.gitconnected.com/deep-dive-into-tls-handshake-e029e28e2eb3" \t "_blank)

[levelup.gitconnected.com](https://levelup.gitconnected.com/deep-dive-into-tls-handshake-e029e28e2eb3" \t "_blank)

**rejectUnauthorized**

The easiest solution to resolve these errors is to use the “rejectUnauthorized” option shown below.

https.request({   
 ....,  
 rejectUnauthorized: false,  
 },  
...)

or set it as environment variable

NODE\_TLS\_REJECT\_UNAUTHORIZED=0

However, this method is unsafe because it disables the server certificate verification, making the Node app open to [MITM](https://en.wikipedia.org/wiki/Man-in-the-middle_attack)attack. Thus, it can only be applied in a development environment and must not be used in production.

**CA Option**

A safer way is to specify the CA certificate that is expected from the server. In other words, the common name of the certificate needs to match with the server certificate.

request({   
 ca: [fs.readFileSync([certificate path])],  
 rejectUnauthorized: true,  
}

As you can see, the ca option is an array, and you can set multiple certificate files if required. The disadvantage is that the certificate has to be specified in the code. It could be a problem as we may need to manage multiple versions of certificates across multiple environments.

**NODE\_EXTRA\_CA\_CERTS**

From Node version 7.3.0, NODE\_EXTRA\_CA\_CERTS environment variable is introduced to pass in a CA certificate file. This allows the “root” CAs to be extended with the extra certificates in file. The file should consist of one or more trusted certificates in **PEM**format.

Please note that the extra certificates won’t be effective if the ca options property is explicitly specified for a HTTPS client or server.

**How to troubleshoot?**

So now you have to use either the ca option or the NODE\_EXTRA\_CA\_CERTS, but you may still receive the same errors. It is normally caused by the incorrect certificate being used. For the ca option or the extra certs to work, we need to get the full CA Chain or at least the Root CA certificate.

You can use [OpenSSL](https://www.openssl.org/) to retrieve the Full CA Chain as following:

openssl s\_client -connect ${REMHOST}:${REMPORT}

An example of the certificate chain is shown below.

Graphical user interface, text, application

Description automatically generated

Google CA Certificate Chain

*Please note that the*showcerts *command may not work if the command is executed behind a proxy or the remote server uses*[*SNI*](https://en.wikipedia.org/wiki/Server_Name_Indication)*.*

**Summary**

In this article, We discuss how to resolve the certificate errors in NodeJs App in details. If you still can’t fix the error after trying everything, then StackOverflow is your best friend :).