Table of Contents

About

Implementing a good certification validation isn't an easy task, but this addon aims even higher. It implements the same steps a browser does to ensure the communication over the negotiated connection is safe and secure. Additionally, provides a management window to easily manage trusted certificates, update and test them. This addon implements all certification verification steps a browser normally do and additionally provides a management window to easily manage trusted certificates, update and test them.

Take a step further to improve the security of all protocols connecting through a TLS connection with a one line setup. Works with **all** protocols Best HTTP/2 supports.

All source code included. Requires the latest version of Best HTTP/2.

Features

- Certificate Chain Verification as described in RFC 3280
- Revocation checking of leaf certificates using OCSP with optional soft and hard fail
- Caching OCSP responses
- Support for OCSP Must-Staple
- Trusted Root CA, Trusted Intermediate and Client Credentials management through an easy to use Certification Manager Window to
    1. Update all certificates from a trusted source
    2. Add custom certificates
    3. Delete non-needed certificates
- Domain Name Matching
- Client Authentication
- Wide variety of options to configure almost every bit of the addon

Setup Guide

This addon doesn't require any installation steps, but it's recommended to import after the Best HTTP/2 package. After importing the package just must call one function on application startup:

```
#if !UNITY_WEBGL || UNITY_EDITOR
using BestHTTP.Addons.TLSSecurity;

TLSSecurity.Setup();
#endif
```

Calling `TLSSecurity.Setup()` going to set up the addon and installs itself as sets its TlsClient factory as the default one. One part of the setup phase is to load the databases into memory and write it to the application's persistent data path (with default settings) and unload the resource. This step needed as the databases going to open those files and read into memory only the required chunks. While it has some disk overhead, more complexity and can work on platforms and devices where it can create and write into a new file, it greatly reduces runtime memory requirements. This step also done only once, when the files are there no resources going to be loaded and written. To make updating certification databases possible, a hash file is generated and compared in the Setup call, so even if the files are there but the hashes are different a new file going to be written.

Under WebGL BestHTTP/2 must use the underlying browser's XmlHTTPRequest implementation, all tls verification is done by the browser.

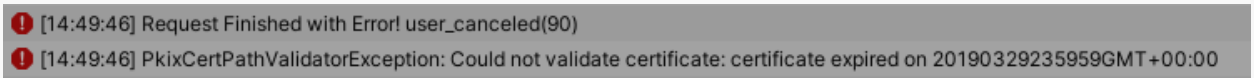
How To Detect TLS Certification Verification Failures

If certification verification fails, the connection going to be terminated the HTTPReqest's State going to be HTTPRequestStates.Error. The request's Exception property is a reference to an exception containing more information about the issue:

```
void OnRequestFinished(HTTPRequest req, HTTPResponse resp)
{
    switch (req.State)
    {
        // The request finished without any problem.
        case HTTPRequestStates.Finished:
            if (resp.IsSuccess)
            {
                Debug.Log("Done!");
            }
            else
            {
                Debug.LogWarning(string.Format("Request finished Successfully, but the server
sent an error. Status Code: {0}-{1} Message: {2}", resp.StatusCode, resp.Message,
resp.DataAsText));
            }
            break;

        // The request finished with an unexpected error. The request's Exception property may
contain more info about the error.
        case HTTPRequestStates.Error:
            Debug.LogError("Request Finished with Error! " + (req.Exception != null ?
(req.Exception.Message + "\n" + req.Exception.StackTrace) : "No Exception"));

            TlsFatalAlert tlsException = req.Exception as TlsFatalAlert;
            if (tlsException != null)
            {
                Debug.LogException(tlsException);
            }
            break;

        // The request aborted, initiated by the user.
        case HTTPRequestStates.Aborted:
            Debug.LogWarning("Request Aborted!");
            break;

        // Connecting to the server is timed out.
        case HTTPRequestStates.ConnectionTimedOut:
            Debug.LogError("Connection Timed Out!");
            break;

        // The request didn't finished in the given time.
        case HTTPRequestStates.TimedOut:
            Debug.LogError("Processing the request Timed Out!");
            break;
    }
```

```
}
```

For a failed TLS verification, it should produce two log lines, something like this:



Certification Manager Window

The *Window/Best HTTP/Addons/TLS Security/Certification Window* menu item (or CTRL+ALT+E shortcut) opens the addon's Certification Manager. Using this window certificates can be added, updated and deleted.

1

2

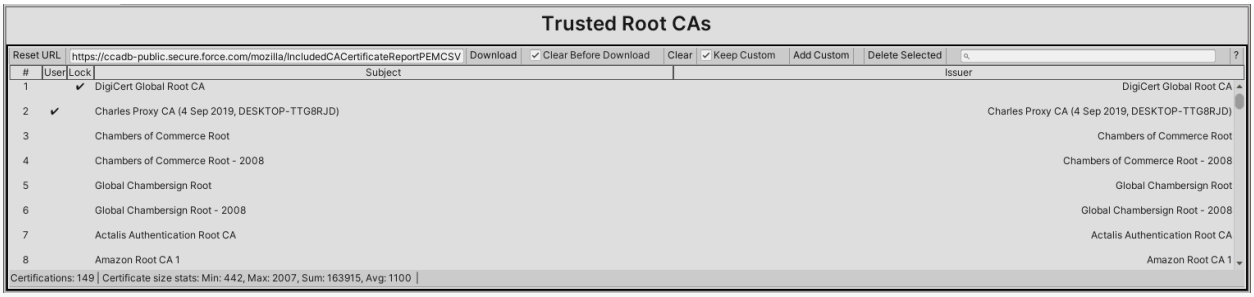
3

4

5

Trusted Root CAs

These are the basis of the trust chain, servers doesn't send root certificates the client must include the roots certificates of the accessed endpoints.



18

1. **Reset URL**: Reset the URL input back to its addon supplied url.
2. **URL Input**: The URL that the addon going to download the certifications. The addon expects CSV formatted data, but the URL can point to a local file using the file:// protocol. The default URLs are pointing to Mozilla repositories.
3. **Download**: Clicking on this button start the downloading, content parsing and loading process. Downloading the certificates already uses all verification implemented in the addon.
4. **Clear Before Download**: Check to remove all non-locked and non-user added (if `Keep Custom` is checked) certificates before download.
5. **Clear**: Remove all non-locked and non-user added (if `Keep Custom` is checked) certificates.
6. **Keep Custom**: If set Clear buttons doesn't remove user added certificates.
7. **Add Custom**: Add certificates from .cer, .pem and .p7b files.
8. **Delete Selected**: Delete selected certificates. Locked certificates can't be deleted!
9. **Search Input**: It can be used to search certificates by their `Subject` name. Minimum 3 characters needed.
10. **Help (?) Button**: Opens a browser window to this manual.
11. **# Column**: Index of the certificate.
12. **User Column**: It has a ✔, if it's a user-added certificate.
13. **Lock Column**: It has a ✔, if it's locked and can't be deleted. Currently only certificates needed to update from the default URL are locked.
14. **Subject Column**: Subject field of the certificate.
15. **Issuer Column**: Issuer field of the certificate.
16. **Certifications**: Number of certifications displayed.
17. **Certificate Size Stats**: Min, max, sum and average size of certificate data in bytes. This can help adjusting cache sizes.
18. **Status**: Status of the last operation.

Notice

Double clicking on a row or hitting Enter while at least one row is selected dumps out certification information to the console.

Trusted Intermediate Certificates

Because servers can choose to not send intermediate certificates it's a good practice to bundle them too.

1. **Reset URL**: Reset the URL input back to its addon supplied url.
2. **URL Input**: The URL that the addon going to download the certifications. The addon expects CSV formatted data, but the URL can point to a local file using the file:// protocol. The default URLs are pointing to Mozilla repositories.
3. **Download**: Clicking on this button start the downloading, content parsing and loading process. Downloading the certificates already uses all verification implemented in the addon.
4. **Clear Before Download**: Check to remove all non-locked and non-user added (if `Keep Custom` is checked) certificates before download.
5. **Clear**: Remove all non-locked and non-user added (if `Keep Custom` is checked) certificates.
6. **Keep Custom**: If set Clear buttons doesn't remove user added certificates.
7. **Add Custom**: Add certificates from .cer, .pem and .p7b files.
8. **Delete Selected**: Delete selected certificates. Locked certificates can't be deleted!
9. **Search Input**: It can be used to search certificates by their `Subject` name. Minimum 3 characters needed.
10. **Help (?) Button**: Opens a browser window to this manual.
11. **# Column**: Index of the certificate.
12. **User Column**: It has a ✔, if it's a user-added certificate.
13. **Lock Column**: It has a ✔, if it's locked and can't be deleted. Currently only certificates needed to update from the default URL are locked.
14. **Subject Column**: Subject field of the certificate.
15. **Issuer Column**: Issuer field of the certificate.
16. **Certifications**: Number of certifications displayed.
17. **Certificate Size Stats**: Min, max, sum and average size of certificate data in bytes. This can help adjusting cache sizes.
18. **Status**: Status of the last operation.

Notice

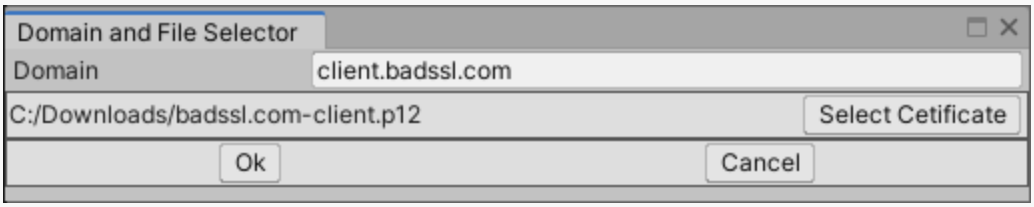
Double clicking on a row or hitting Enter while at least one row is selected dumps out certification information to the console.

Client Certificates

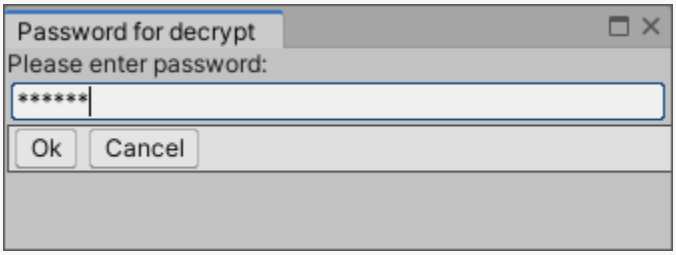
A client certificate can be associated with a domain. If the server asks for a client certificate during the TLS handshake, the client going to send it back.

1. **Add for domain**: Clicking on it a `Domain and File Selector` window is shown. If the domain is filled and the certification file is selected clicking on the *Ok* button going to add the certification for the domain.
2. **Delete Selected**: Delete selected domain-certificate associations.
3. **Help (?) Button**: Opens a browser window to this manual.
4. **# Column**: Index of the certificate
5. **Target Domain Column**: The certificate sent only if it's requested for the target domain.
6. **Authority Column**: *Common Name* or *Organizational Unit Name* from the certificate's Issuer field.
7. **Certifications**: Number of certifications displayed.
8. **Certificate Size Stats**: Min, max, sum and average size of certificate data in bytes. This can help adjusting cache sizes.

Clicking on the `Add for domain` button a new window opens to select the certification file and domain:

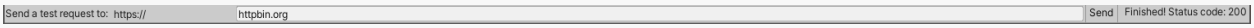


Then, clicking on the *Ok* button depending on the type of certificate file a window to input the file's password might open:



Testing HTTP Requests

A basic GET request can be sent out for the given domain to test the current setup.



1. Input field for the domain to test
2. Send button
3. Result of the request

Warning

Because of Connection Pooling a request that otherwise would fail can succeed if there's an already open connection to the domain!

Bottom Toolbar

1. Name and version number of this addon
2. Support e-mail

SecurityOptions

Options of the addon can be accessed through the static `SecurityOptions` class:

- **UseServerSentIntermediateCertificates**: If false, only certificates stored in the trusted intermediates database are used to reconstruct the certificate chain. When set to true (default), it improves compatibility but the addon going to use/accept certificates that not stored in its trusted database.
- **FolderAndFileOptions**: Folder, file and extension options.
- **OCSP**: OCSP and OCSP cache options.
- **TrustedRootsOptions**: Database options of the Trusted CAs database.
- **TrustedIntermediatesOptions**: Database options of the Trusted Intermediate Certifications database
- **Database options of the Client Credentials database**: Database options of the Client Credentials database

OCSP Options

- **ShortLifeSpanThreshold**: The addon not going to check revocation status for short lifespan certificates.
- **EnableOCSPQueries**: Enable or disable sending out OCSP requests for revocation checking.
- **FailHard**: Treat unknown revocation statuses (unknown OCSP status or unreachable servers) as revoked and abort the TLS negotiation.
- **FailOnMissingCertStatusWhenMustStaplePresent**: Treat the TLS connection failed if the leaf certificate has the must-staple flag, but the server doesn't send certificate status.
- **OCSPCache**: OCSP Cache Options as detailed below

OCSP Cache Options

OCSP request caching related options.

- **MaxWaitTime**: Maximum wait time to receive an OCSP response. Depending on the OCSP Options' `FailHard` value if no response is received in the given time the TLS negotiation might fail.
- **RetryUnknownAfter**: Wait time to retry to get a new OCSP response when the previous response's status is unknown.
- **FolderName**: OCSP cache's folder name.
- **DatabaseOptions**: Options for the OCSP cache database.
- **HTTPRequestOptions**: Customization options for the OCSP requests.

OCSP Cache's HTTPRequest Options

OCSP requests are plain old `HTTPRequest`s and every BestHTTP/2 global settings affecting them, but through this options OCSP requests can be further customized.

- **DataLengthThreshold**: A threshold in bytes to switch to a POST request instead of GET. Setting it to `0` all requests are sent as POST.
- **UseKeepAlive**: Whether to try to keep the connection alive to the OCSP server.
- **UseCache**: Whether to cache responses if possible.
- **ConnectTimeout**: Time limit to establish a connection to the server.
- **Timeout**: Time limit to send and receive an OCSP response from the server.

Database Options

- **Name**: Name of the database. This name is used to create the database files.
- **UseHashFile** Whether to calculate a hash from the database and write it to a file. It has a useage only if the file is created 'offline' and bundled with the application.
- **DiskManager**: Options for the database's DiskManager instance.

DiskManager Options

- **MaxCacheSizeInBytes**: This limits the maximum database rows kept in memory.
- **HashDigest**: Hash digest algorithm name to generate the database's hash.

Examples

```
#if !UNITY_WEBGL || UNITY_EDITOR
using BestHTTP.Addons.TLSSecurity;

// To disable the OCSP cache's memory cache:
SecurityOptions.OCSP.OCSPCache.DatabaseOptions.DiskManager.MaxCacheSizeInBytes = 0;

TLSSecurity.Setup();
#endif
```