LEEC: Let's Encrypt Erlang with Ceylan

Contact: about (dash) leec (at) esperide (dot) com

Creation date: Wednesday, November 11, 2020

Lastly updated: Sunday, January 10, 2021

Dedication: Users and maintainers of the `LEEC` library, version 0.6.

Abstract: The role of the `LEEC` library is to interact from Erlang/OTP with Let's Encrypt servers, mostly in order to generate X.509 certificates.

The latest version of this documentation is to be found at the official LEEC website (`http://leec.esperide.org`).

The documentation is also mirrored here.

Table of Contents

Overview

The online documentation for LEEC is currently available mostly here.

Design Notes

Multiple Domains Having Each Multiple Hostnames

At least the ACME servers from Let's Encrypt enforce various fairly low rate limits, which leads to preferring requesting certificates only on a per-domain basis (ex: for `foobar.org`) rather than on a per-hostname one (ex: for `baz.foobar.org`, `hurrican.foobar.org`, etc., these hosts being virtual ones or not), as such requests would quickly become too numerous to respect these rate thresholds.

A per-domain certificate should then include directly its various hostnames as *Subject Alternative Names* (SAN entries).

With the `http-01` challenge type, no wildcard for such SAN hosts (ex: `*.foobar.org`) cannot be specified), so all the wanted ones have to be explicitly listed[1].

So for example, with LEEC, the certificate for `foobar.org` should list following SAN entries: `baz.foobar.org`, `hurrican.foobar.org`, etc.

Concurrent Certificate Operations

LEEC implemented independent (`gen_statem`) FSMs to allow typically for concurrent certificate renewals to be triggered. A drawback of the aforementioned Let's Encrypt rate limits is that, while a given FSM is to remain below said thresholds, a set of parallel ones may not.

If a task ring may be used to avoid by design such FSMs to overlap, another option is to use a single FSM and to trigger certificate requests in turn.

Getting Information about the Generated Certificates

If using LEEC to generate a certificate for a `baz.foobar.org` host, the following three files shall be obtained from the Let's Encrypt ACME server:

- `baz.foobar.org.csr`: the PEM certificate request, sent to the ACME server (~980 bytes)
- `baz.foobar.org.key`: the TLS private key regular file, kept on the server (~1675 bytes)
- `baz.foobar.org.crt`: the PEM certificate itself of interest (~3450 bytes), to be used by the webserver

To get information about this certificate:

```
$ openssl x509 -text -noout -in baz.foobar.org.crt

Certificate:
   Data:
       Version: 3 (0x2)
```

[1] As a result, the certificate may disclose virtual hosts that would be otherwise invisible from the Internet (as not even declared in the DNS entries for that domain).

```
Serial Number:
    04:34:17:fd:ee:9b:bd:6b:c2:02:b1:c0:84:62:ed:a6:88:5c
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, O = Let's Encrypt, CN = R3
Validity
    Not Before: Dec 27 08:21:38 2020 GMT
    Not After : Mar 27 08:21:38 2021 GMT
Subject: CN = baz.foobar.org
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)

       Modulus:
            [...]
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 Basic Constraints: critical
        CA:FALSE
    X509v3 Subject Key Identifier:
        [...]
    X509v3 Authority Key Identifier:
        keyid:C0:CC:03:46:B9:58:20:CC:5C:72:70:F3:E1:2E:CB:20:B6:F5:

    Authority Information Access:
        OCSP - URI:http://ocsp.stg-int-x1.letsencrypt.org
        CA Issuers - URI:http://cert.stg-int-x1.letsencrypt.org/

    X509v3 Subject Alternative Name:
        DNS:hello.baz.foobar.org.crt, DNS:world.foobar.org.crt, DNS:
    X509v3 Certificate Policies:
        Policy: 2.23.140.1.2.1
        Policy: 1.3.6.1.4.1.44947.1.1.1
          CPS: http://cps.letsencrypt.org

    CT Precertificate SCTs:
        Signed Certificate Timestamp:
            Version   : v1 (0x0)
            Log ID    : [...]
            Timestamp : Jan  2 09:23:20.310 2021 GMT
            Extensions: none
            Signature : ecdsa-with-SHA256
        Signed Certificate Timestamp:
            Version   : v1 (0x0)
            Log ID    : [...]
            Timestamp : Jan  2 09:23:20.320 2021 GMT
            Extensions: none
```

```
                    Signature : ecdsa-with-SHA256
                                [...]
    Signature Algorithm: sha256WithRSAEncryption
    [...]
```

Troubleshooting HTTPS Certificate-related Issues

In order to understand why a given host (typically a webserver) does not seem to handle properly certificates, one may experiment with these commands from a client computer:

```
$ curl -vvv -I https://foobar.org
$ wget -v https://foobar.org -O -
$ openssl s_client -connect foobar.org:443
```

From the server itself:

```
$ iptables -nL
$ lsof -i:443
$ netstat -ltpn | grep ':443'
```

Using third-party solutions:

- test your server with SSL Labs

Dependency Issues between Webservers and HTTP(s) Clients

A potential dependency problem is that many Erlang-based webservers are powered by Cowboy (thus Cowlib) whereas LEEC used to rely necessarily on Shotgun, thus on Gun (and thus Cowlib) as well. Most of the time this implied different (potentially incompatible) versions of Cowlib, whereas only one should exist in the code path.

We prefer sticking to the Cowlib version induced by Cowboy. At the time of this writing, the latest Cowboy stable version (the one that webserver projects such as USWeb want) is 2.8.0 and relies on Cowlib 2.9.1, whereas the latest Shotgun stable version, 0.5.0, is lagging behind, relying on Gun 1.3.1, itself relying on Cowlib 2.6.0 (too old).

An attempt of solution was to remove the dependency of LEEC onto Shotgun (as it induced a dependency on an older Cowlib) but to use Gun instead, which is lower-level yet might be chosen in order to rely on the target Cowlib version. However we did not found a suitable Gun version for that (1.3 being too old, 2.0.* not ready).

So a last-resort solution has been to rely instead on the even lower-level Erlang-native httpc client module (involving `inets` and `ssl`). The result, although based only on HTTP/1.1 with no connection-reuse, proved satisfactory right from the start and thus is provided as an alternate way of using LEEC, without any extra dependency.

This allows embedding LEEC with only a dependency onto Myriad and a JSON parser (either JSX or Jiffy), and no other one (top-level or induced).

Support

Bugs, questions, remarks, patches, requests for enhancements, etc. are to be sent through the project interface, or directly at the email address mentioned at the beginning of this document.

Please React!

If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, follow the Support guidelines).

Ending Word

Have fun with LEEC!