LEEC

Technical Manual of LEEC: *Let's Encrypt Erlang with Ceylan*

Organisation: Copyright (C) 2020-2021 Olivier Boudeville

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

Creation date: Wednesday, November 11, 2020

Lastly updated: Sunday, February 21, 2021

Dedication: Users and maintainers of the `LEEC` library

Version: 0.6.1

Abstract: The role of the `LEEC` library is to interact from Erlang/OTP with servers implementing the ACME protocol - specifically *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 LEEC library is the Ceylan fork of the original and much appreciated letsencrypt-erlang, which is a Let's Encrypt client library for Erlang whose author is Guillaume Bour.

LEEC's purpose is to obtain proper X.509 security certificates from `Let's Encrypt` - or more generally ACME servers (version 2), typically in order to secure one's web-servers so that they can offer a solid HTTPS connectivity, which is pretty much standard nowadays.

LEEC is notably used in the context of US-Web.

Differences Introduced by this Fork

Compared to the original `letsencrypt-erlang` library, the main differences introduced by LEEC are:

- it is more specialised, in the sense that LEEC focuses on the "slave" use case (i.e. to be directly integrated within an Erlang webserver), as opposed to the "web-root" one (a third-party webserver is running separately with little possibilities of direct interactions) or the "standalone" one (where no specific prior webserver would be running, the certificate agent operating then its own one)
- more comments, more spell-checking, much clarification
- more typing, more runtime checking, extended traces supported
- security increased (notably using 4096-bit RSA keys)
- dependency onto Ceylan-Myriad added, to benefit from its facilities
- JSON parser can be JSX (the default), or Jiffy (refer to the `JSON parsers` section)
- HTTP client can be either `Shotgun` or the Erlang-native `httpc` client, to avoid any extra dependencies on `Gun` and `Cowlib` (whose versions could potentially clash with the ones required by any `Cowboy`-based integrating webserver)
- porting done from gen_fsm (soon to be deprecated) to the newer gen_statem
- minor API changes and additions, for a clearer and more flexible mode of operation
- fixed the compilation with Erlang version 23.0 and higher (ex: w.r.t. to `http_uri`/`uri_string`, to updated dependencies such as Jiffy, and newer Cowboy for the examples)
- allow for *concurrent* certificate requests (ex: if managing multiple domains with different keys, new certificates being requested for all of them at webserver start-up); so LEEC generates certificates in parallel and does not rely on a *registered* FSM (*Finite State Machine*) anymore
- global, ETS-based TCP connection pool replaced by an (optional) per-FSM internal cache (if relying on Shotgun)

- support for SAN (Subject Alternative Name) certificates, an extension to X.509 enabling a certificate to include a `subjectAltName` field to list, here, extra DNS names that are covered by this certificate
- basic support for the management of:
  - *Ephemeral Diffie-Helman* key, to ensure *Forward Secrecy* by relying on a set of keys that are never communicated
  - Intermediate Let's Encrypt Certificates

So, even if LEEC can be seen mostly as a "reckless" fork (in the sense that it became quickly obvious that retaining upstream compatibility could hardly be achieved) - with so many source-level differences (in terms of conventions, Myriad integration, whitespace cleanup) that a pull request can difficultly be considered - yet, in spite of the appearances, it remained quite close to the original (mainly differences of form) and followed the same structure.

By some ways, this LEEC fork is safer and more robust than the original, by others not (ex: test coverage, autonomous use, continuous integration). A key goal was to make it easier to understand and maintain.

Most of the elements of this pull request from Marc Worrell have also been integrated.

Prerequisites

Dependency Basics

The general dependencies are:

- `openssl`, version 1.1.1 or higher (required to generate RSA key and certificate request)
- `Erlang/OTP` (tested with 23.1 versions and upwards)

The LEEC-specific ones, which are automatically managed by rebar3 if opting for a rebar-based build, are:

- a JSON parser: either jsx (the default) or jiffy
- Ceylan-Myriad, for the various facilities on which LEEC relies
- optional: a more advanced HTTP client than the httpc Erlang-native one, namely Shotgun, which should be more efficient (TCP connection re-used, recent HTTP, etc.) at the cost of an extra dependency (which may clash with any your application may introduce, refer to the dependency issues section)

Switching JSON Parsers

If wanting to switch from the default jsx to jiffy, following files shall be updated:

- rebar.config (knowing it is generated from conf/leec.app.src)
- src/leec.app.src (knowing it is a mere symlink to `ebin/leec.app`, which is itself generated from conf/leec.app.src)

(none in Myriad)

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 up to one should exist in the code path at any time.

We prefer sticking to the Cowlib version that is induced by Cowboy. At the time of this writing, the latest Cowboy stable version (the one that webserver projects such as US-Web 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 involving any extra dependency.

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

Building

Two build procedures can be used, and are now mostly the same.

From the root of LEEC):

- either a rebar3-based one: then run `make all-rebar3`, simply corresponding to:

```
$ rebar3 upgrade
$ rebar3 compile
```

- or one relying on Ceylan's native build system, once the relevant prerequisites have been setup (selected, downloaded, built): run `make all`

This last procedure is the one that we prefer and use routinely (see the US-Web native deployment script as an example thereof).

Usage Example

The main example of LEEC in action can be found in link with US-Web, whose sources can be found here; see notably class_USCertificateManager.erl and us_web_letsencrypt_handler.erl.

This mode of operation, described in this section, is typical of the use case where an Erlang-based webserver (in this case based on Cowboy) has to renew certificates corresponding to the various virtual hosts (possibly dispatched under various domains) that it is making available.

A first part is to create as many LEEC FSMs as domains of interest, which will connect to the target ACME servers (most probably Let's Encrypt ones). Each FSM

is a LEEC agent that will generate its own (strong) RSA key, create automatically its throwaway ACME account on the server, secure properly the communication (with TLS signatures, nonces, etc.), and wait for further user request regarding its domain of interest (ex: `foobar.org`).

Such a request is bound to ask the ACME server to generate (as a Certificate Authority) a X.509 certificate covering, thanks to SAN, a set of subdomains (FQDN) to secure (ex: `hello.foobar.org`, `hurricane.foobar.org`) - knowing that no wildcard certificate can be obtained with the `http-01` challenge being used here. The ACME server will send challenges to LEEC so that it can prove that it controls indeed all these subdomains.

A second part of the LEEC action is to ensure that these answers are available indeed, as tokens. In practice the ACME server will attempt to read them at specific URLs (prefixed with `.well-known/acme-challenge/`) expected to be served for these subdomains (most probably thanks to virtual hosting). If the ACME server is able to query and read, directly from a domain, the right tokens corresponding to the challenges it sent for this domain, then the proof of actual control by the requester is established, and the ACME server can thus issue a corresponding certificate and transmit it appropriately to LEEC.

The overall webserver of the user shall thus track the transitions of these FSMs until (hopefully) they successfully complete their procedure and obtain from their ACME server the corresponding certificate. Then only the user webserver will be able to fire its https support with these brand new certificates[1].

Finally, a task scheduler may be used to trigger renewals on time (not too soon, not too late, as ACME rules apply and, of course, each FQDN shall be covered by a valid certificate at any time), and a task ring may be used to (paradoxically) ensure that the webserver as a whole does not interact too much in parallel (through its various LEEC FSMs) with the ACME server (despite hosting potentially a large number of FQDNs), knowing that severe rate limits (example in production) apply.

LEEC does its best to go through this procedure, validating as much as possible each of these steps for a better reliability/control, and reporting outcome for tracability and error management.

In practice, the user code is expected:

(A) to initialise first LEEC, with `leec:start/{1,2}` and proper options (see leec.erl); the PID of the corresponding LEEC FSM is then returned

(B) to request, thanks to this PID, a certificate to be generated for a domain, with `leec:obtain_certificate_for/{2,3}`

(C) to answer properly to the corresponding challenges for each (sub)domain, by delivering the right LEEC-computed tokens; see `leec:send_ongoing_challenges/2`

(D) to poll this FSM to establish if/when the targeted certificate is available; actually it is more convenient to define in (2) a callback to be triggered by LEEC when appropriate

For US-Web, (1), (2) and (4) are managed by class_USCertificateManager.erl (see respectively `init_leec/5`, `request_certificate/1` and the `onCertificateRequestOutcome/2` callback). (3) is taken in charge by us_web_letsencrypt_handler.erl (see `init/2`).

[1] Before, even if suitable certificates were pre-existing, at least the ACME URL prefix was to remain over http instead of being automatically promoted to https as all others.

Design Notes

Multiple Domains Having Each Multiple Hostnames

At least the ACME servers from Let's Encrypt enforce various rate limits (both in production environment and in staging one) that are fairly low, which leads to preferring requesting certificates only on a per-domain basis (ex: globally for `foobar.org`) rather than on a per-hostname host basis (ex: one for `baz.foobar.org`, another one for `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`) can be specified, so all the wanted ones have to be explicitly listed[2].

So for example, with LEEC, the certificate for `foobar.org` (that would be managed by a dedicated LEEC agent) 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 (thanks to autonomous LEEC agents, per-FSM connection pools, separate keys, etc.).

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.

Should this issue arise, an option is to use a single FSM and to trigger certificate requests in turn. Another one is to rely on a task ring in order to avoid by design that such FSMs overlap.

Let's Encrypt Accounts

Currently LEEC creates automatically throwaway ACME accounts, which is convenient yet may prevent the use if CAA (*Certificate Authority Authorization*).

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:

[2] 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 that would act as wildcard name resolvers).

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

Certificate:
   Data:
       Version: 3 (0x2)
       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
    [...]
```

Other Files of Interest

A `*.key` (ex: `my-foobar-leec-agent-private.key`) file is a (PEM, strong enough) RSA private key generated by LEEC so that its agent can safely authenticate to the ACME servers it is interacting with.

`lets-encrypt-r3-cross-signed.pem` is the (PEM) certificate associated to the *Certificate Authority* (Let's Encrypt here). It is automatically downloaded by LEEC if not already available.

The `dh-params.pem` file contains the parameters generated by LEEC in order to allow for safer *Ephemeral Diffie-Helman key exchanges* that is used to provide Forward Secrecy with TLS (see this article for further information). Its generation may take quite some time.

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'
```

Third-party solutions might also be used, like testing your server with SSL Labs; thanks to LEEC, US-Web can be ranked "grade A" there.

Licence

Ceylan-LEEC is distributed under the APACHE 2.0 licence, like the original work that it derives from.

Support

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

Possible Enhancements

- re-using ACME accounts: not creating throwaway, anonymous accounts but (possibly) reusing them by registering the ACME client with its email, etc.
- supporting certificate revocation
- supporting Elliptic Curve cryptography
- reintroducing elements brought by the upstream project yet not updated by the current fork: unit testing, standalone testing, hex package, various escripts and yml files involved
- besides the slave mode (main use case of interest with LEEC), better integrating/testing the other modes (webroot and standalone)
- supporting extra validation challenges, besides `http-01`, like `dns-01` (necessary to obtain wildcard certificates, i.e. applying to all subdomains of a given domain) and `proof-of-possession-01`
- supporting directly other ACME services besides `Let's Encrypt` (like `ZeroSSL`)

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! (not supposed to involve any memory leak)