leap.mail Documentation

Release 0.4.0

Kali Kaneko

Sep 27, 2017

Contents

decentralized and secure mail delivery and synchronization

This is the documentation for the `leap.mail` module. It is a twisted package that allows to receive, process, send and access existing messages using the LEAP platform.

One way to use this library is to let it launch two standard mail services, `smtp` and `imap`, that run as local proxies and interact with a remote `LEAP` provider that offers *a soledad syncronization endpoint* and receives the outgoing email. This is what Bitmask client does.

From the release 0.4.0 on, it's also possible to use a protocol-agnostic email public API, so that third party mail clients can manipulate the data layer. This is what the awesome MUA in the Pixelated project is using.

CHAPTER 1

How does this all work?

All the underlying data storage and sync is handled by a library called soledad, which handles encryption, storage and sync. Based on u1db, documents are stored locally as local `sqlcipher` tables, and syncs against the soledad sync service in the provider.

OpenPGP key generation and keyring management are handled by another leap python library: keymanager.

See the life cycle of a leap email for an overview of the life cycle of an email through `LEAP` providers.

CHAPTER 2

Data model

The data model at the present moment consists of several *document types* that split email into different documents that are stored in `Soledad`. The idea behind this is to keep clear the separation between *mutable* and *inmutable* parts, and still being able to reconstruct arbitrarily nested email structures easily.

CHAPTER 3

Documentation index

Hacking

Some hints oriented to *leap.mail* hackers. These notes are mostly related to the imap server, although they probably will be useful for other pieces too.

Don't panic! Just manhole into it

If you want to inspect the objects living in your application memory, in realtime, you can manhole into it.

First of all, check that the modules `PyCrypto` and `pyasn1` are installed into your system, they are needed for it to work.

You just have to pass the `LEAP_MAIL_MANHOLE=1` enviroment variable while launching the client:

```
LEAP_MAIL_MANHOLE=1 bitmask --debug
```

And then you can ssh into your application! (password is "leap"):

```
ssh boss@localhost -p 2222
```

Did I mention how *awesome* twisted is?? :)

Profiling

If using `twistd` to launch the server, you can use twisted profiling capabities:

```
LEAP_MAIL_CONFIG=~/.leapmailrc twistd --profile=/tmp/mail-profiling -n -y imap-server.
↪tac
```

`--profiler` option allows you to select different profilers (default is "hotshot").

You can also do profiling when using the `bitmask` client. Enable the `LEAP_PROFILE_IMAPCMD` environment flag to get profiling of certain IMAP commands:

```
LEAP_PROFILE_IMAPCMD=1 bitmask --debug
```

Offline mode

The client has an `--offline` flag that will make the Mail services (imap, currently) not try to sync with remote replicas. Very useful during development, although you need to login with the remote server at least once before being able to use it.

Mutt config

You cannot live without mutt? You're lucky! Use the following minimal config with the imap service:

```
set folder="imap://user@provider@localhost:1984"
set spoolfile="imap://user@provider@localhost:1984/INBOX"
set ssl_starttls = no
set ssl_force_tls = no
set imap_pass=MAHSIKRET
```

Running the service with twistd

In order to run the mail service (currently, the imap server only), you will need a config with this info:

```
[leap_mail]
userid = "user@provider"
uuid = "deadbeefdeadabad"
passwd = "foobar" # Optional
```

In the `LEAP_MAIL_CONFIG` enviroment variable. If you do not specify a password parameter, you'll be prompted for it.

In order to get the user uid (uuid), look into the `~/.config/leap/leap-backend.conf` file after you have logged in into your provider at least once.

Run the twisted service:

```
LEAP_MAIL_CONFIG=~/.leapmailrc twistd -n -y imap-server.tac
```

Now you can telnet into your local IMAP server and read your mail like a real programmer™:

```
% telnet localhost 1984
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ IDLE NAMESPACE] Twisted IMAP4rev1 Ready
tag LOGIN me@myprovider.net mahsikret
tag OK LOGIN succeeded
tag SELECT Inbox
* 2 EXISTS
* 1 RECENT
* FLAGS (\Seen \Answered \Flagged \Deleted \Draft \Recent List)
* OK [UIDVALIDITY 1410453885932] UIDs valid
```

```
tag OK [READ-WRITE] SELECT successful
^]
telnet> Connection closed.
```

Although you probably prefer to use `offlineimap` for tests:

```
offlineimap -c LEAPofflineimapRC-tests
```

Minimal offlineimap configuration

You can use this as a sample offlineimap config file:

```
[general]
accounts = leap-local

[Account leap-local]
localrepository = LocalLeap
remoterepository = RemoteLeap

[Repository LocalLeap]
type = Maildir
localfolders = ~/LEAPMail/Mail

[Repository RemoteLeap]
type = IMAP
ssl = no
remotehost = localhost
remoteport = 1984
remoteuser = user
remotepass = pass
```

Testing utilities

There are a bunch of utilities to test IMAP delivery in `imap/tests` folder. If looking for a quick way of inspecting mailboxes, have a look at `getmail`:

```
./getmail me@testprovider.net mahsikret
1. Drafts
2. INBOX
3. Trash
Which mailbox? [1] 2
1 Subject: this is the time of the revolution
2 Subject: ignore me

Which message? [1] (Q quits) 1
1 X-Leap-Provenance: Thu, 11 Sep 2014 16:52:11 -0000; pubkey="C1F8DE10BD151F99"
Received: from mx1.testprovider.net(mx1.testprovider.net [198.197.196.195])
(using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/256 bits))
(Client CN "*.foobar.net", Issuer "Gandi Standard SSL CA" (not verified))
by blackhole (Postfix) with ESMTPS id DEADBEEF
for <me@testprovider.net>; Thu, 11 Sep 2014 16:52:10 +0000 (UTC)
Delivered-To: 926d4915cfd42b6d96d38660c04613af@testprovider.net
Message-Id: <20140911165205.GB8054@samsara>
From: Kali <kali@leap.se>
```

```
(snip)
```

IMAP Message Rendering Regressions

For testing the IMAP server implementation, there is a litte regressions script that needs some manual work from your side.

First of all, you need an already initialized account. Which for now basically means you have created a new account with a provider that offers the Encrypted Mail Service, using the Bitmask Client wizard. Then you need to log in with that account, and let it generate the secrets and sync with the remote for a first time. After this you can run the twistd server locally and offline.

From the `leap.mail.imap.tests` folder, and with an already initialized server running:

```
./regressions_mime_struct user@provider pass path_to_samples/
```

You can find several message samples in the `leap/mail/tests` folder.

Debugging IMAP commands

Use `ngrep` to obtain logs of the commands:

```
sudo ngrep -d lo -W byline port 1984
```

To get verbose output from thunderbird/icedove, set the following environment variable:

```
NSPR_LOG_MODULES="imap:5" icedove
```

API documentation

If you were looking for the documentation of the `leap.mail` module, you will find it here.

Of special interest is the public mail api, which should remain relatively stable across the next few releases.

leap.mail package

Subpackages

mail.adaptors package

Subpackages

mail.adaptors.tests package

Submodules

mail.adaptors.tests.test_models module

Submodules

leap.mail.smtp.gateway module

Submodules

leap.mail.constants module

leap.mail.decorators module

leap.mail.interfaces module

leap.mail.load_tests module

leap.mail.mail module

leap.mail.mailbox_indexer module

leap.mail.size module

leap.mail.sync_hooks module

leap.mail.utils module

leap.mail.walk module

CHAPTER 4

Indices and tables

- genindex
- modindex
- search