Bitmask Documentation

Release 0.10.0

LEAP Encryption Access Project

Oct 09, 2017

Contents

Release v0.10. (*Installation* and *Known Issues*)

CHAPTER 1

What is Bitmask?

Bitmask is the client for the services offered by the LEAP Platform. The services currently deployed are Encrypted Internet Proxy (VPN) and Encrypted Mail.

Bitmask offers a **command-line interface** and a **multiplatform desktop client**. It can be also used as a **set of libraries** to communicate with the different services from third party applications.

Bitmask is written in python using Twisted and licensed under the GPL3. The Graphical User Interface is written in html+js and uses PyQt5 for serving the application.

CHAPTER 2

Understood! Show me the docs!

These documents that you are reading are, mostly, a **guide for developers** that want to contribute to the development of Bitmask, and seek to understand better the code organization and the contribution process.

The **authoritative users guide** lives at bitmask.net.

Other important documents about the LEAP Project can be found at the Official LEAP documentation site. If you ever need an offline copy, you can clone the repo for the LEAP Docs site. That repo contains also the related LEAP Platform documentation and all the latest design documents. Enhancement contributions and new translations are always welcome! Just open a new merge request.

On the contrary, this developers documentation you are reading right now is maintained in the bitmask-dev git repo itself, and can also be checked online.

Building the docs

if you want to build these docs locally, you can do:

```
make doc
```

from the topmost folder in the bitmask-dev repo. Note that you need to have sphinx installed.

CHAPTER 3

Contents

Installation

Here you can find instructions for developers and advanced users. For **user instructions**, you should refer to the official Bitmask Install Guide. You should only need to read the following sections if:

- You plan to contribute code to bitmask core libraries.
- You intend to develop the Bitmask JS User Interface.
- You are a prospective maintainer for some platform yet unsupported.
- Your platform is unsupported by the official packages, and you want to install the python packages in your system.

If you want to contribute translations to some of these sections, please get in touch with us, it will be greatly appreciated to extend the community.

With Pip

If we still do not provide packages for your platform (debian/ubuntu only at the moment), and for some reason you cannot run the bundles we offer for download, you still should be able to run bitmask downloading the packages from pypi. First you will need some dependencies in your system, that very probably will be provided by your package manager:

```
lxpolkit openvpn gnupg1 python-pyqt5 python-dev libffi-dev
```

Now you can install the latest bitmask package from pypi:

```
pip install leap.bitmask[gui]
```

If you want also to use the pixelated MUA, you need to install an additional extra:

```
pip install leap.bitmask[pixelated]
```

From git

If you want to run latest code from git, you can refer to the setting up the development environment section to learn how to run Bitmask from the latest code in the master branch.

Building bundles

The standalone bundles are built with PyInstaller.

If you are inside a development virtualenv, you should be able to install it together with some extra development dependencies with:

```
pip install -r pkg/requirements-dev.pip
```

And then just do:

```
make bundle
```

To build a new bundle.

There's also a script that automates re-creating the virtualenv from which the packaging takes place:

```
pkg/build_bundle_with_venv.sh
```

To ensure a repeatable system-wide environment, you can build those bundles from within a docker container. First you need to create the container:

```
make docker_container
```

and then you can launch the above script inside that container:

```
make bundle_in_docker
```

That will build the latest from master. If you're interested in building some development branch, you can pass some extra variables:

```
make bundle_in_docker REPO=https://0xacab.org/kali/bitmask-dev BRANCH=somethingcool
```

A new bundle is created by the CI for every commit using this procedure involving docker, you can read more about the bundles in the *qa section*.

Debian & ubuntu

You only need to paste three lines to get bitmask installed under debian or ubuntu, and get updates as they are released, or as they are merged into master. Watch a screencast of the install process.

Refer to the section in the install guide about the different debian repositories that we provide to understand how the different packages are produced.

The debian packages are maintained in the `debian/` folder in the source code repo.

For a quick rebuild of local changes, you can do:

```
debuild -us -uc
```

Archlinux

Not officially supported, but DoctorJellyFace maintains a PKGBUILD that can be found in the AUR repo.

Testing & QA

Latest bundles for the next release cycle are automatically built by our Gitlab CI for every commit in master.

Beginning with 0.9.5, we have ported again the VPN service into the Bitmask client. Choose the following testing providers (beware that no guarantee about continuity of the accounts is made at this point):

- For Encrypted Email, test against `https://cdev.bitmask.net`
- For VPN, test against `https://demo.bitmask.net`

Reporting bugs

- Bug reports go into our Issue Tracker.
- Here is some very good read about what constitutes a good bug report.
- Have also a look at the *Known Issues* page.

Logs

Bitmask stores the logs in `$HOME/.config/leap/bitmaskd.log`. You might want to remove the whole `~/.config/leap` folder when trying a new account, or when you stop using Bitmask.

Tips for QA

If you want to give a hand testing the unreleased bundles, please follow the following tips:

- Focus all your efforts, if possible, on whatever is *the* golden distro at the time of the release. This currently is: Ubuntu 17.04.x (zesty), 64bits, with Unity as the default desktop environment. It's very important to have a reference environment as bug-free as possible, before trying to solve issues that are present in other distributions or window managers.
- Identify all issues that need help in the QA phase. You can do that going to the bug tracker, and filtering all the issues for a given release that are in the QA state.
- If the issue is solved in your tests for this alpha release, please add a comment to the issue stating the results of your tests, and the platform and desktop environment in which your tests took place. But please do not change the QA status on the issue. We generally leave this role to the author of the original issue, or to the person playing the role of the release QA master.
- Always test with a newly created account (specially relevant when testing email candidates)
- Always test with the reference Mail User Agent (currently, Thunderbird, in whatever version is present in the reference distribution).
- Remove also any thunderbird configuration, start a freshly configured account.
- If you find a new bug, please make sure that it hasn't already been reported in the issue tracker. If you are absolutely certain that you have found a new bug, please attach a log of a new bitmask session, which should contain *only* the behaviour needed to reproduce the bug you are reporting.

On a live image (xenial)

Pasting the following line in a terminal will help you testing the latest bundle from inside a virtual machine running a **live image for xenial** (note that this is **not** an installation method!):

```
curl https://0xacab.org/leap/bitmask-dev/raw/master/docs/testing/latest-bundle-xenial␣
↪| bash
```

Providers with invalid certificates

If you **really** need to test against a provider without a valid certificate, you can use the following flag. I assume you know what you are doing:

```
SKIP_TWISTED_SSL_CHECK=1 bitmask
```

Thunderbird integration

You can configure Thunderbird manually with the info that is shown in the "Configure a Mail Client" section, within the Mail pane in Bitmask. However, there is a Thunderbird extension that automates this configuration. Before configuring a Bitmask account in Thunderbird, make sure that you have Bitmask running and that you are logged in with an account that supports the mail service (mail.bitmask.net).

These are the steps:

1. Run Bitmask, login with an account that suports encrypted email.
2. From within Thunderbird, install the Bitmask Thunderbird Extension.
3. Enable the menu bar: Right Click in the top bar > Menu Bar
4. Create a Bitmask Account in Thunderbird: From the menubar, click on File > New > Bitmask Account
5. Fill in your name. This can be anything.
6. Fill in your username, in the form "username@provider"

Known Issues

VPN

- If there's no running polkit agent in the system, the UI hangs in the 'Connecting' State. (solution: install and run some agent, like `lxpolkit`).
- Some problems with resuming after hibernation (more debug info is welcome).

Wizard

- In the wizard log in / sign up page, the username field gets deselected.
- The list of providers should have icons, be sortable, filterable.
- The wizard should look more pretty.

Main window

- UI doesn't subscribe to events yet, won't get updated if user has logged out via the command line interface.
- Removing an account does not actually clean up all the files associated with that account (need backend code).
- #8840: No ability to recover if the key generation fails.

Pixelated Integration

The integration of the Pixelated webmail is a bit rough at the moment.

Particularly, simultaneous use of Pixelated and Thunderbird is likely to hit some usability issues:

- #8905: pixelated: if a message is open in Thunderbird, the unread count is not updated
- #8906: pixelated: can't see messages that were sent with Thunderbird
- Pixelated MUA is not authenticated.

Hacking

So you want to hack on Bitmask? Thanks, and welcome!

This document assumes a `Linux` environment.

There are also ongoing documents with notes for setting up an OSX and a Windows working environments too, contribution is very much welcome on those docs!

Running tests

Tox is all you need:

```
tox
```

Test when changes are made to common/soledad

If you are developing against a non-published branch of `leap.common` or `leap.soledad`, run instead:

```
tox -e py27-dev
```

This expects `leap_common` and `soledad` repos to be checked out in the parent folder.

Gitlab-runner

For debugging issues related to how tests are run by the gitlab CI, you need to install:

- docker_ce from docker's repositories.
- gitlab-runner from gitlab's repositories

You probably want to add *sleep 9000* to allow debuggin on the docker container. For convenience, you can execute the packaging step by doing:

```
make package_in_docker
```

Look at the `Makefile` to see the command that it's actually used.

Architecture

There is a small explanation of the bitmask components that might be helpful to get you started with the different moving parts of the Bitmask codebase.

User Interface

The Javascript User Interface has its own guide for developers.

Setup

Read how to setup your python development environment to start coding in no time.

Debug

A must-read for debugging the Bitmask Core daemon is the manhole HowTo.

Privileges

Bitmask VPN needs administrative privileges. For developing, you need to be sure that you have installed the privileged helpers and that you keep them udpated.

Contributing

There are some guidelines for contributing code that you might want to check if you are insterested in developing with Bitmask.

Release

We keep a checklist for the release process.

Ideas

Want to help, but you don't know where to start? Come talk to us on irc or the mailing list!

Some areas in which we always need contribution are:

- Localization of the client (talk to elijah).
- Multiplatform gitlab runners
- Windows and OSX packaging (talk to kali)
- Windows Firewall integration for VPN
- Migrating components to py3 (look for vshyba or kali).

- Minimal C++ Qt client (see kali's bitmaskqt5 repo)

Bitmask CLI

`bitmaskctl` is the command line interface.

It will try to launch the bitmask backend.

Creating an user

```
bitmaskctl user create user@example.org
```

If the provider needs invite codes to register new users, you can pass one:

```
bitmaskctl user create --invitecode xxxxxxx user@example.org
```

Authenticating

To authenticate, start a session and start configured services:

```
bitmaskctl user auth user@example.org
```

Uploading logs

(This needs `pastebinit` installed in your system)

```
bitmaskctl logs send
```

Bitmask VPN

The Bitmask VPN Module

Gateway Selection

By default, the Gateway Selector will apply a heuristic based on the configured timezone of the system. This will choose the closest gateway based on the timezones that the provider states in the `eip-config.json` file.

If the locations section is not properly set by the provider, or if the user wants to manually override the selection, the only way to do this for the `0.10` version of Bitmask is to add a section to the `bitmaskd.cfg` configuration file:

```
[vpn]
locations = ["rio__br"]
countries = ["BR", "AR", "UY"]
```

Take into account that the locations entry has precedence over the country codes enumeration.

Also, the normalization is done so that any non-alphabetic character is substituted by an underscore ('_).

You can list all the configured locations using the CLI:

```
% bitmaskctl vpn list
demo.bitmask.net       [DE] Frankfurt (UTC+1)
demo.bitmask.net       [US] Seattle, WA (UTC-7)
```

This manual override functionality will be exposed through the UI and the CLI in release `0.11`.

Bitmask Core

The bitmask core daemon can be launched like this:

```
bitmaskd
```

The command-line program `bitmaskctl` and the GUI will launch the daemon when needed.

Starting the REST service

If configured to do so, the bitmask core will expose all of the commands throught a REST API. In bitmaskd.cfg:

```
[services]
web = True
```

Global API Authentication

To avoid some kind of attacks, the Bitmask API is protected by a global authentication token.

The JS API receives this value when the initial entrypoint is loaded for the first time, in the anchor part of the url.

To authenticate any request to the API, the `X-Bitmask-Auth` header has to be added to it, set to the single value that is initialized during the bitmask deaemon startup:

```
curl -X POST http://localhost:7070/API/mail/status
unauthorized:bad auth token

curl -X POST http://localhost:7070/API/mail/status -H 'X-Bitmask-Auth:␣
↪fae20706aa4f4f98ac0e67996787a370'
{"result": {"status": "on", "childrenStatus": {"smtp": {"status": "on", "error": null}
↪, "imap": {"status": "on", "error": null}}, "error": null}, "error": null}
```

This token can be found in `.config/leap/authtoken`

API Authentication (this section not implemented yet)

By default, the resources in the API are protected by an authentication token.

The rationale is that, since the Bitmask core can be used simultaneously by several users, no single user should be able to interfere with others by querying for sensible information or disrupting other users' interaction with running services.

Therefore, there's a small white list of resources that do not need authentication, based on which is the subset of the API that needs to provide functionality for the creation of new accounts (the first-run wizard from the UI perspective).

The local authentication token is sent back in the response for an authentication call.

This local session token is different from the remote SRP token, although both are returned together. In the case that the remote SRP authentication with the provider fails (or with no network connectivity), the backend **should** signal the error but equally return a local authentication token (this is not implemented yet, but needs to be done to support an offline mode of operation).

To authenticate any request to the API, the `Authorization` header has to be added to it. You need to pass a `Token` field, with a value equal to the concatenation of the username and the local session token , base64-encoded:

```
$ curl -X POST localhost:7070/API/core/stop
$ Unauthorized

>>> import base64
>>> base64.b64encode('user@provider.org:52dac27fcf633b1dba58')
'dXNlckBwcm92aWRlci5vcmc6NTJkYWMyN2ZjZjYzM2IxZGJhNTg='

$ curl -X POST localhost:7070/API/core/stop -H 'Authorization: Token␣
↪dXNlckBwcm92aWRlci5vcmc6NTJkYWMyN2ZjZjYzM2IxZGJhNTg='
$ {'shutdown': 'ok'}
```

Resources

Following is a list of currently available resources and a brief description of each one. For details click on the resource name.

By default, all the resources need authentication. An asterisk next to it means that it does not need an authentication header.

Resource	Description
`POST` */core/version* *	Get Bitmask Core Version Info
`POST` */core/stats* *	Get Stats about Bitmask Usage
`POST` */core/status*	Get Bitmask Status
`POST` */core/stop*	Stop Bitmask Core
`POST` */bonafide/provider/list* *	List all providers
`POST` */bonafide/provider/create* *	Create a new provider
`POST` */bonafide/provider/read* *	Get info about a provider
`POST` cmd_prov_del	Delete a given provider
`POST` */bonafide/user/list*	List all users
`POST` */bonafide/user/active*	Get active user
`POST` */bonafide/user/create* *	Create a new user
`POST` */bonafide/user/update*	Change the user password
`POST` */bonafide/user/authenticate* *	Authenticate an user
`POST` */bonafide/user/logout*	End session for an user
`POST` */mail/status*	Get all known keys for an user
`POST` */keys/insert/*	Insert a new key
`POST` */keys/delete/*	Delete a given key
`POST` */keys/export/*	Export keys

Passing parameters

In all the cases that need data passed as parameter, those will be passed as JSON-encoded data to the POST.

/core/version

POST /core/version

Get Bitmask Core Version Info

Example request:

```
curl -X POST localhost:7070/API/core/version
```

Example response:

```
{
   "error": null,
   "result":
       {
          "version_core": "0.9.3+185.g59ee6c29.dirty"
       }
}
```

/core/stats

POST /core/stats

Get Stats about Bitmask Usage

/core/status

POST /core/status

Get Bitmask status

/core/stop

POST /core/stop

Stop Bitmask core (daemon shutdown).

/bonafide/provider/list

POST /bonafide/provider/list

List all known providers.

/bonafide/provider/create

POST /bonafide/provider

Create a new provider.

/bonafide/provider/read

POST /bonafide/provider/read

Get info about a given provider.

Example request:

```
curl -X POST localhost:7070/API/bonafide/provider/read -d '["dev.bitmask.net
↪"]'
```

Example response:

```
{
"error": null,
     "result": {
     "api_uri": "https://api.dev.bitmask.net:4430",
     "api_version": "1",
     "ca_cert_fingerprint": "SHA256:␣
↪0f17c033115f6b76ff67871872303ff65034efe7dd1b910062ca323eb4da5c7e",
     "ca_cert_uri": "https://dev.bitmask.net/ca.crt",
     "default_language": "es",
     "description": {
         "en": "Bitmask is a project of LEAP",
     },
     "domain": "dev.bitmask.net",
     "enrollment_policy": "open",
     "languages": [
         "es"
     ],
     "name": {
         "en": "Bitmask"
     },
     "service": {
         "allow_anonymous": false,
         "allow_free": true,
         "allow_limited_bandwidth": false,
         "allow_paid": false,
         "allow_registration": true,
         "allow_unlimited_bandwidth": true,
         "bandwidth_limit": 102400,
         "default_service_level": 1,
         "levels": {
             "1": {
                 "description": "Please donate.",
                 "name": "free"
             }
         }
     },
     "services": [
         "mx",
         "openvpn"
     ]
 }
}
```

Form parameters:

- `domain` *(required)* - domain to obtain the info for.

/bonafide/provider/delete

POST /bonafide/provider/delete

Delete a given provider.

/bonafide/user/list

POST /bonafide/user/list

List all the users known to the local backend.

Form parameters:

Status codes:

- `200` - no error

/bonafide/user/active

POST /bonafide/user/active

Get the active user.

/bonafide/user/create

POST /bonafide/user/create

Create a new user.

Form parameters:

- `username` *(required)* - in the form user@provider.
- `pass` *(required)* - the username passphrase
- `invitecode` *(optional)* - an optional invitecode, to be used if the provider requires it for creating a new account.
- `autoconf` *(optional)* - whether to autoconfigure the provider, if we have not seen it before.

Status codes:

- `200` - no error

/bonafide/user/update

POST /bonafide/user/update

Change the user password.

Form parameters:

- `username` *(required)* - in the form user@provider
- `oldpass` *(required)* - current password
- `newpass` *(required)* - new password

Status codes:

- `200` - no error

/bonafide/user/authenticate

POST /bonafide/user/authenticate

Authenticate an user.

Form parameters:

- `username` *(required)* - in the form user@provider
- `pass` *(required)* - passphrase
- `autoconf` *(optional)* - whether to autoconfigure the provider, if we don't have seen it before.

Status codes:

- `200` - no error

/bonafide/user/logout

POST /bonafide/user/logout

Logs out an user, and destroys its local session.

/mail/status

**POST /mail/status

Get the status of the mail service

Example request:

```
curl -X POST localhost:7070/API/mail/status -d '["foobar@mail.bitmask.net"]'
```

Example response:

```
{
  "result": {
    "status": "on",
    "keys": "found",
    "unread": 5,
    "childrenStatus": {
      "keymanager": {
        "status": "on",
        "keys": "found",
        "error": null
      },
      "smtp": {
        "status": "on",
        "error": null
      },
      "incoming": {
        "status": "on",
        "unread": 5,
        "error": null
      },
      "imap": {
        "status": "on",
        "error": null
      }
```

```
    },
    "error": null
  },
  "error": null
}
```

Form parameters:

- `address` *(required)* - the email address of the account.

Values:

The possible values for `status` are:

- `on`
- `off`
- `starting`
- `stopping`
- `failed`

The possible values for the `keys` field are:

- `null` - nothing is known about the private key status.
- `sync` - syncing soledad to see if there is a private key.
- `generating` - creating a new private key as none was found.
- `found` - there is a valid private key in KeyManager.

/keys/list

POST /keys/list

Get all keys for an user.

/keys/insert/

POST /keys/insert

Insert a new key for an user.

/keys/delete/

POST /keys/delete

Delete a key for an user.

/keys/export/

POST /keys/export

Export keys for an user.

Bonafide

blah blah

Using the library

Keymanager

Keymanager is the Bitmask component that does key management, including generation, discovery and validation. It is, esentially, a nicknym client that uses Soledad as its storage layer.

Keymanager handles the creation of a OpenPGP transparently in user's behalf. When bootstrapping a new account, keymanager will generate a new key pair. The key pair is stored encrypted inside soledad (and therefore able to be synced by other replicas). After generating it, the public key is sent to the provider, which will replace any prior keys for the same address in its database.

To discover keys for other users, the nicknym client in keymanager will query the nicknym server associated with user's provider, and will process the keys that the server returns. This query has the following form:

```
https://nicknym.test.bitmask.net:6425?address=user@example.com
```

It is up to the the provider's service to determine the sources for the keys.

Keymanager currently implements all the levels defined in the Transitional Key Validation spec, although the mechanisms for validation currently in place only reach level 2 of what's contemplated in the spec.

Sources of public keys

Currently Bitmask can discover new public keys from different sources:

- Keys attached to incoming emails. Simple *.asc* attachments with the keys will be taken into account, like the ones produced by enigmail.
- OpenPGP header in incoming emails. The only field taken into account is the *url* and it will only be used if it's an *https* address from the same domain as the senders email address.
- When sending emails, if the recipient key is not known, Bitmask will ask for the key to its nicknym provider. The nicknym provider will try to discover the key from other nicknym providers. If provider discovery does not bring any results, the key will be fetched from the sks key servers. Note that this *sks discovery mechanism is probably going to be removed at some time in the future as it provides too many unused keys*.

Other methods are planned to be added in the future, like discovery from signatures in emails, headers (autocrypt spec) or other kind of key servers.

Implementation: using Soledad Documents

KeyManager uses two types of Soledad Documents for the keyring:

- key document, that stores each gpg key.
- active document, that relates an address to its corresponding key.

Each key can have 0 or more active documents with a different email address each:

Fields in a key document:

- uids
- fingerprint
- key_data
- private. bool marking if the key is private or public
- length
- expiry_date
- refreshed_at
- version = 1
- type = "OpenPGPKey"
- tags = ["keymanager-key"]

Fields in an active document:

- address
- fingerprint
- private
- validation
- last_audited_at
- encr_used
- sign_used
- version = 1
- type = "OpenPGPKey-active"
- tags = ["keymanager-active"]

The meaning of validation, encr_used and sign_used is related to the Transitional Key Validation

Bitmask Mail

decentralized and secure mail delivery and synchronization

This is the documentation for the `leap.mail` module. It is a twisted module, hanging from the `leap.bitmask` namespace, 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 mail 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.

From release 0.10 on, the Bitmask Bundles will also ship a branded version of the Pixelated User Agent, that will be served locally. This will be one of the recommended ways of accessing the user emails. The other will be Thunderbird, by using the Bitmask Thunderbird Extension.

Note that this used to be a standalone python package, under the `leap.mail` namespace. It was merged into bitmask repo, so it now lives in the `leap.bitmask.mail` namespace. The legacy repo will no longer be updated.

How does Bitmask Mail 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, discovery, validation, and keyring management are handled by the `leap.bitmask.keymanager` module.

The life cycle of a LEAP Email

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

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.

Authentication

Currently, IMAP and SMTP are twisted services that are binded to `localhost`. These services be initialized by the bitmask.core daemon, but they are not tied to any user session. When an use attempts to log in to those services, a `twisted.cred` pluggable authentication plugin will try to lookup a `mail token` that is stored inside the soledad encrypted storage.

From within the cli, you can get the mail token once you are authenticated with:

```
bitmaskctl mail get_token
```

When launched, the `bitmaskd` daemon writes the tokens for each account to a file inside the folder named `/tmp/bitmask_tokens`, and this is where the Thunderbird Extension reads them from.

Pixelated user agent

From the 0.9.5 release, bundles are shipping the Pixelated User Agent. Until some merge requests (dealing with packaging of the js resources) are merged upstream, you will need to install the pixelated user agent from leap's repo:

```
pip install pixelated_www pixelated_user_agent --find-links https://downloads.leap.se/
↪libs/pixelated/
```

Pixelated also needs a couple of extra dependencies:

```
pip install whoosh chardet requests==2.11.1
```

Mail development resources

Some old notes that might help you while developing or debugging bitmask mail issues (a bit outdated).

Hacking on Bitmask Mail

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.

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").

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
```

Debugging IMAP

After IMAP service is running, 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
```

Use `ngrep` to obtain live logs of the commands and responses:

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

Thunderbird

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

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

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.

Changelog

0.10.2 - master

Note: This version is not yet released and is under active development.

Bugfixes

- #9099: properly check for openvpn binary path in bundles.

0.10.1

Bugfixes

- #9073: fix bootstrapping of providers like riseup.

- Use the right path for firewall in debian packages.

0.10.0 - la rosa de foc

Features

- Initial cli port of the legacy vpn code
- #8112: Check validity of key signature
- #8755: Add account based keymanagement API
- #8770: Simplify mail status in the cli
- #8769: Eliminate active user from bonafide
- #8771: Add json print to the cli
- #8765: Require a global authentication token for the api
- #8819: Send key to provider if a new priv key is putted in the keyring
- #8821: Add a 'fetch' flag to key export
- #8049: Restart the VPN automatically
- #8852: Stop the vpn (and all services) when application is shut down
- #8804: Automatic selection of gateways, based on user timezone
- #8855: Manual override for the vpn gateway selection
- #8835: On bitmaskclt vpn start use the last vpn if no provider is provided
- #9059: Automatic renewal of vpn certificate
- #8895: Check for running pkexec in the system
- #8977: Download config files if newer ones are found in the provider
- Add VPN API to bitmask.js
- Add vpn get_cert command
- Indicate a successful/failure OpenPGP header import
- Get more detailed status report for email
- VPN and Mail status displayed in the UI
- Port Pixelated UA integration from legacy bitmask
- Add Pixelated Button to the UI
- Add ability to ssh into the bitmask daemon for debug
- Add a call to inject messages into a mailbox using the cli.
- New `bitmask_chromium` gui: launches Bitmask UI as a standalone chromium app if chromium is installed in your system
- Add new debianization split, with separated bitmask components.
- #9029: add a package for the bitmask javascript UI.

Bugfixes

- #8783: use username instead of provider in the vpn calls
- #8868: can't upload generated key with bitmask
- #8832: don't allow putting non-private keys for the keyring address
- #8901: use gpg1 binary if present
- #8971: handle 502 replies from nicknym
- #8957: alot doesn't automatically decrypt messages sent from Bitmask
- Repeat decryption if signed with attached key
- Log error in case JSON parsing fails for decrypted doc

Misc

- Remove usage of soledad offline flag.
- Tests use soledad master instead of develop
- Build bundles with pixelated libraries

0.9.4 - works for you

Features

- #7550: Add ability to use invite codes during signup
- #7965: Add basic keymanagement to the cli.
- #8265: Add a REST API and bitmask.js library for it.
- #8400: Add manual provider registration.
- #8435: Write service tokens to a file for email clients to read.
- #8486: Fetch smtp cert automatically if missing.
- #8487: Add change password command.
- #8488: Add list users to bonafide.
- Use mail_auth token in the core instead of imap/smtp tokens.

Bugfixes

- #8498: In case of wrong url don't leave files in the config folder.

List of design docs

Here you can find a list of links to authoritative sources for the design documents

- (all the links to leap.se/docs)
- Paper on the LEAP architecture https://satsymposium.org/papers/leap.pdf

List of contributors

The following people have contributed in the history of Bitmask.

- drebs drebs at leap dot se
- Ruben Pollan meskio at sindominio dot net
- elijah elijah at riseup dot net
- Tomás Touceda chiiph at leap dot se
- Ivan Alejandro ivanalejandro0 at gmail dot com
- Kali Kaneko kali at leap dot se
- Micah Anderson micah at riseup dot net
- kwadronaut kwadronaut at leap dot se
- Bruno Wagner Goncalves bwagner at thoughtworks dot com
- Victor Shyba victor.shyba at gmail dot com
- Duda Dornelles ddornell at thoughtworks dot com
- Folker Bernitt fbernitt at thoughtworks dot com
- Parménides GV parmegv at sdf dot org
- Caio Carrara caiocarrara at gmail dot com
- Giovane Liberato giovaneliberato at gmail dot com
- Thais Siqueira thais dot siqueira at gmail dot com
- NavaL ayoyo at thoughtworks dot com
- Denis Costa deniscostadsc at gmail dot com
- Zara Gebru zgebru at thoughtworks dot com
- Tulio Casagrande tcasagra at thoughtworks dot com
- Sriram Viswanathan sriramv at thoughtworks dot com
- irregulator irregulator at riseup dot net
- Paixu Aabuizia PaixuAabuizia at users dot noreply at github dot com
- Christoph Kluenter ckluente at thoughtworks dot com
- Simon Fondrie-Teitler simonft at riseup dot net
- Neissi Lima neissi.lima at gmail dot com
- k clair kclair at riseup dot net
- antialias antialias at leap dot se
- Jaromil jaromil at dyne dot org

Updating the authors file

From the root of the `bitmask-dev` repo:

```
pkg/tools/get_authors.sh
```

However, beware that some of the codebase from the legacy bitmask_client repo has not been preserved with the original authorship. Do not remove people here just because you don't see the commits they made :)

- search