LemonLDAP::NG Documentation

Release 2.0

LemonLDAP::NG

Jun 20, 2021

CONTENTS

CHAPTER

ONE

DOCUMENTATION

1.1 Presentation

- *How it works*
- *Main features*
- *Quick start tutorial*

1.2 Workshops

- LDAPCon 2019: Connect LL::NG to OpenLDAP and use 2FA, configure SSO on Fusion Directory and Dokuwiki
- Pass the SALT 2019: Connect LL::NG to OpenLDAP and use 2FA, configure SSO on Fusion Directory

1.3 Installation and configuration

- Maintained versions:
  - Version 3.0 (dev)
  - Version 2.0 (stable)
  - Version 1.9 (oldstable)
- Archived versions (unmaintained by LLNG Team)
  - Version 1.4
  - Version 1.3
  - Version 1.2

- Version 1.1
- Version 1.0

1.3.1 Packaged versions

These versions are maintained under distribution umbrella following their policy.

Debian

Tip: Following Debian Policy, LLNG packages are never upgraded in published distributions. However, security patches are backported by maintenance teams *(except some inor ones)*.

Debian dist		LLNG version	Secured	Maintenance	LTS Limit	Extended LTS Limit
6	*Squeeze*	*0.9.4.1*	No known vulnerability	*None*	*February 2016*	*April 2019*
7	Wheezy	1.1.2	No known vulnerability	**None**[1]	May 2018	Probably 2021
8	Jessie	1.3.3	CVE-2019-19791 tagged as minor	**None**[1]	June 2020	Probably 2023
9	Stretch	1.9.7	CVE-2019-19791 tagged as minor	Debian LTS Team	June 2022	
	Stretch-backports	2.0.2	CVE-2019-12046, CVE-2019-13031, CVE-2019-15941	*None*	*June 2019*	
	Stretch-backports-sloppy	2.0.11		LLNG Team, "best effort"[3]	Until Debian 11 release[4]	
10	Buster	2.0.2	CVE-2019-19791 tagged as minor	Debian Security Team	Probably July 2024	
	Buster-backports	2.0.11		LLNG Team	Until Debian 11 release[4]	
	Bullseye	2.0.11		Debian Security Team	Probably July 2026	
Next	Testing	Latest[5]		LLNG Team		

See Debian Security Tracker and Debian Package Tracker for more.

[1] Possible Extended LTS
[3] updated by LLNG Team until dependencies are compatible
[4] around June 2021
[5] few days after release

Ubuntu

Attention: Ubuntu version are included in "universe" branch[8], so not really security maintained. Prefer to use our repositories or Debian ones

Ubuntu dist		LLNG version	Secured	Maintenance
12.04	Precise	1.1.2	No known vulnerability	None
14.04	Trusty	1.2.5	No known vulnerability	None
16.04	Xenial[9]	1.4.6	CVE-2019-12046, CVE-2019-13031	None
18.04	Bionic[9]	1.9.16	CVE-2019-12046, CVE-2019-13031, CVE-2020-24660	None
18.10	Cosmic	1.9.17	CVE-2019-12046, CVE-2019-13031, CVE-2020-24660	None
19.04	Disco	2.0.2	CVE-2019-12046, CVE-2019-13031, CVE-2019-15941, CVE-2020-24660	None
19.10	Eoan	2.0.5	CVE-2019-15941, CVE-2020-24660	None
20.04	Focal[9]	2.0.7	CVE-2020-24660	None
20.10	Groovy	2.0.8	CVE-2020-24660	None
20.10	Hirsute	2.0.11		None

1.4 Bug report

See *Reporting a bug*.

1.5 Development

- Bugtracker
- Source code

[8] Ubuntu universe/multiverse branches are community maintained *(so not maintained by Canonical)*, but in fact nobody considers LLNG security issues. See this issue for example

[9] LTS

- Nightly trunk builds *(for Debian or Ubuntu,***really unstable***)*
- Git access:

```
git clone https://gitlab.ow2.org/lemonldap-ng/lemonldap-ng.git
```

- CPAN test reports:
  - LemonLDAP::NG Common
  - LemonLDAP::NG Handler
  - LemonLDAP::NG Portal
  - LemonLDAP::NG Manager

1.6 Other

- Conferences
- References
- Press

CHAPTER

TWO

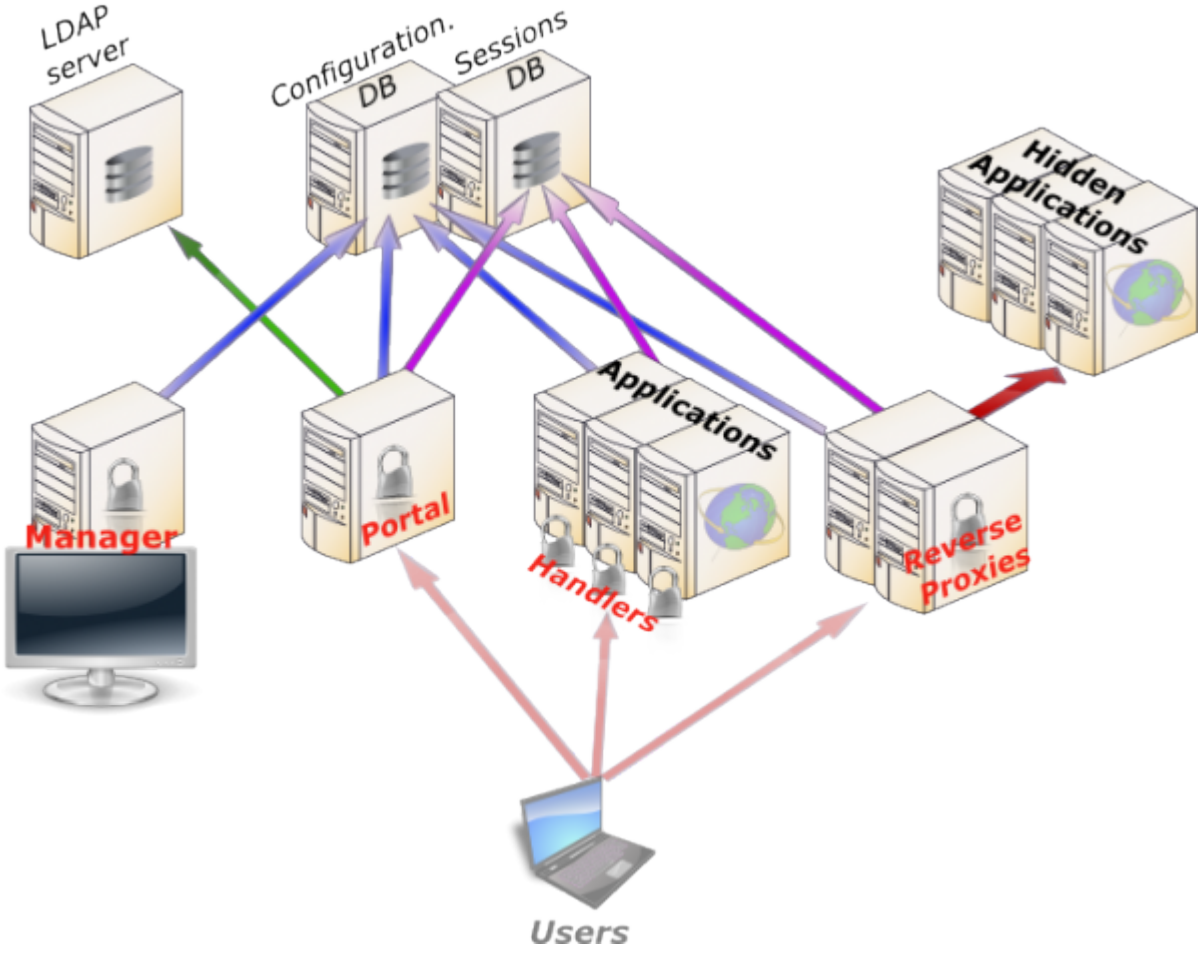
PRESENTATION

2.1 Presentation

LemonLDAP::NG is a modular WebSSO (Single Sign On) based on Apache::Session modules. It simplifies the build of a protected area with a few changes in the application.

It manages both authentication and authorization and provides headers for accounting. So you can have a full AAA protection for your web space as described below.

2.1.1 Architecture



Main components

- **Manager**: used to manage LemonLDAP::NG configuration and to explore sessions. Dedicated to administrators
- *Portal*: used to authenticate users, display applications list and provides identity provider service (SAML, OpenID, CAS). Futhermore, Portal affordes many other features (see *portal* for more)
- *Handler*: used to protect applications which can read HTTP headers or environment variables to get user information

Databases

Attention: We call "database" a backend where we can read or write a data. This can be a file, an LDAP directory, etc.

We split databases in two categories:

- **External databases**: not managed by LemonLDAP::NG, for example user database
- **Internal databases**: only used by LemonLDAP::NG

Main *external databases* are:

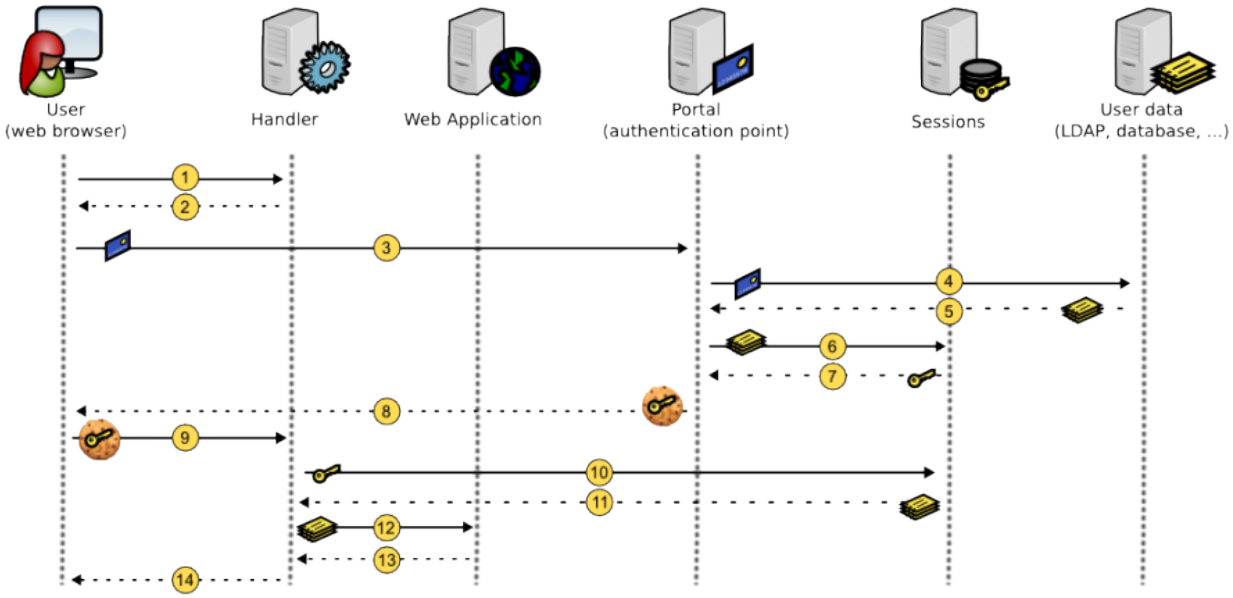
- **Authentication**: how authenticate users
- **User**: where collect user data
- **Password**: where change the password

Main internal databases are:

- *Configuration*: where configuration is stored. This does not include web server configuration which is not managed by LemonLDAP::NG
- *Sessions*: where sessions are stored.
- *Notifications*: messages displayed to connected users
- **Cache**: cache for configuration and sessions

2.1.2 Kinematics

Login



1. User tries to access protected application, his request is catched by Handler
2. *SSO cookies* is not detected, so Handler redirects user to Portal
3. User authenticates on Portal
4. Portal checks authentication
5. If authentication succeed, Portal collect user data
6. Portal creates a session to store user data
7. Portal gets the session key
8. Portal creates *SSO cookies* with session key as value
9. User is redirected on protected application, with his new cookie
10. Handler reads session key from cookie and retrieves user session datas
11. Handler stores user datas in its cache
12. Handler check access rule and send headers to protected application
13. Protected application sends response to Handler
14. Handler forwards the response to user

Then handler will check *SSO cookies* for each HTTP request.

Logout

Default use case:

1. User clicks on the logout link in Portal
2. Portal destroys session and redirects user on itself with an empty *SSO cookies*
3. User is redirected on portal and his *SSO cookies* is empty

LemonLDAP::NG is also able to *catch logout request* on protected applications, with different behavior:

- **SSO logout**: the request is not forwarded to application, only the SSO session is closed
- **Application logout**: the request is forwarded to application but SSO session is not closed
- **SSO and Application logout**: the request is forwarded to application and SSO session is closed

After logout process, the user is redirected on portal, or on a configured URL.

Session expiration

The session expires after 20 hours by default. This duration can be set in the manager's Configuration tab (General Parameters > Sessions > Sessions Timeout).

Attention:

- Handlers have a session cache, with a default lifetime of 10 minutes. So for Handlers located on different physical servers than the Portal, a user with an expired session can still be authorized until the cache expires.
- Sessions are deleted by a scheduled task. Don't forget to install cron files !

Cross Domain Authentication (CDA)

Note: For security reason, a cookie provided for a domain cannot be sent to another domain. To extend SSO on several domains, a cross-domain mechanism is implemented in LemonLDAP::NG.

1. User owns *SSO cookies* on the main domain (see *Login kinematics*)
2. User tries to access a protected application in a different domain
3. Handler does not see *SSO cookies* (because it is not in main domain) and redirects user on Portal
4. Portal recognizes the user with its *SSO cookies*, and see he is coming from a different domain
5. Portal redirects user on protected application with a token as URL parameter. The token is linked to a session which contains the real session ID
6. Handler detects URL parameter, gets the real session ID, delete the token session and creates a *SSO cookies* on its domain, with session ID as value

2.1.3 Authentication, Authorization and Accounting (AAA) mechanisms

Authentication

If a user is not authenticated and attempts to connect to an area protected by a LemonLDAP::NG compatible Handler, he is redirected to a portal.

Authentication process main steps are:

- **Control asked URL**: prevent XSS attacks and bad redirections
- **Control existing session**: detect SSO session, apply configured constraints (1 session per user, 1 session per IP, ...)
- **Extract form info**: get login/password, certificate, environment variable (depending on authentication module)
- **Get user info**: contact user database to collect attributes
- **Ask for second factor if required**: TOTP, U2F key, etc...
- **Set macros**: compute configured macros
- **Set groups**: request user database to find groups
- **Set local groups**: compute configured groups
- **Authenticate**: contact authentication database to check credentials
- **Grant session**: check rights to open SSO session
- **Store**: store user info in session database
- **Build cookie**: build *SSO cookies* with session ID
- **Redirect**: redirect user on protected application or on Portal (applications menu)

LemonLDAP::NG *SSO cookies* are generated by Apache::Session, they are as secure as a 128-bit random cookie. You may use the *securedCookie* options to avoid session hijacking. (since version 1.4.0 you can use SHA256 for generating safer cookies)

Authorization

Authorization is controlled only by Handlers. An authorization is defined by:

- An URL pattern (or `default` to match other URLs)
- An access rule

Note: Authorizations are defined inside a virtualhost and takes effect only on it. There are no *global* authorizations except the right to open a session in the portal.

Access rules values can be:

- `accept`: all authenticated users can pass
- `deny`: nobody is welcomed
- `skip`: all is open!
- `unprotect`: all is open, but authenticated users are seen as authenticated
- `logout_sso`, `logout_app`, `logout_app_sso`: catch logout request

- Perl expression: perl code snippet that returns 0 or 1

Some examples:

- Accept all authenticated users:
  - URL pattern: `default`
  - Access rule: `accept`
- Restrict /admin to administrators group
  - URL pattern: `^/admin/`
  - Access rule: `$groups =~ /\badministrators\b/`

Tip: \b means start or end of a word in PCRE (Perl Compatible Regular Expressions)

See *Writing rules and headers* chapter.

Accounting

Logging portal access

Portal produce a `notice` message in *Web server logs or syslog* when a user authenticates (or fails to authenticate) and logs out.

Logging application access

Handler informs Web server of connected user (parameter `whatToTrace`), so you can see user login in Web server access logs.

The real accounting has to be done by the application itself since SSO logs can not understand transactions.

LemonLDAP::NG can export *HTTP headers* either using a proxy or protecting directly the application.

An HTTP header is defined by:

- A name
- A value

Note: Headers are defined inside a virtualhost and take effect only on it. There are no *global* headers.

The header value is a Perl expression, returning a string.

Some examples:

- Send login in Auth-User:
  - Name: `Auth-User`
  - Value: `$uid`
- Send "Lastname, firstname" in Auth-Name:
  - Name: `Auth-Name`
  - Value: `$sn + ", " + $gn`

See *Writing rules and headers* for more.

2.2 Main features

2.2.1 Full access control

LL::NG is a web single-sign-on system, but unlike some systems it can manage rights on applications based on regular expressions on URL.

2.2.2 Easy to customize

LL::NG is designed using Model–View–Controller software architecture, so you just have to *change HTML/CSS files* to customize the portal.

2.2.3 Easy to integrate

Integrating applications in LL::NG is easy since its dialogue with applications is based on *customizable HTTP headers*.

Unifying authentications (Identity Federation)

LL::NG can easily exchange with other authentication systems by using SAML, OpenID or CAS protocoles. It may be the backbone of a heterogeneous architecture. LL:NG can be set as Identity provider, Service Provider or Protocol Proxy (*LL::NG as federation protocol proxy*).

Its SOAP API can also be used to dialogue directly with your custom applications.

2.2.4 Sessions

Session explorer

LL::NG Manager has a session explorer module that can be used to browse opened sessions:

- by users
- by IP *(IPv4 and IPv6)*
- by double IP (sessions opened by the same user from multiple computers)
- by date

It can be used to delete a session

Session restrictions

By default, a user can open several *sessions*. LL::NG can restrict the following:

- Allow only one session per user
- Allow only one IP address per user
- Allow only one user per IP address

Those capabilities can be used simultaneously or separately.

Double cookie

LL::NG can be configured to provides *2 cookies*:

- one secured (SSL only) for sensitive applications
- one unsecured for other applications

So that if the http cookie is stolen, sensitive applications remain secured.

2.2.5 Notifications

LL::NG can be used to notify users with a message when authenticating. This can be used to inform of a change in access rights, the publication of a new IT charter, etc. (See *notifications* for more details)

2.3 Quick start tutorial

Attention: This tutorial will guide you into a minimal installation and configuration procedure. You need some prerequisites:

- Root access to a Debian, Ubuntu, CentOS or RHEL test system
- A web browser
- A cup of coffee (or tea, we are open minded)

2.3.1 Installation

Debian / Ubuntu

```
apt install apt-transport-https
wget -O - https://lemonldap-ng.org/_media/rpm-gpg-key-ow2 | apt-key add -
echo "deb https://lemonldap-ng.org/deb stable main" > /etc/apt/sources.list.d/lemonldap-
↪ng.list
apt update
apt install lemonldap-ng
```

CentOS / RHEL

```
curl https://lemonldap-ng.org/_media/rpm-gpg-key-ow2 > /etc/pki/rpm-gpg/RPM-GPG-KEY-OW2
echo '[lemonldap-ng]
name=LemonLDAP::NG packages
baseurl=https://lemonldap-ng.org/redhat/stable/$releasever/noarch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-OW2' > /etc/yum.repos.d/lemonldap-ng.repo
yum update
yum install epel-release
yum install lemonldap-ng
# If you use SELinux
yum install lemonldap-ng-selinux
```

2.3.2 SSO domain configuration

LemonLDAP::NG needs all its components to be served on the same DNS domain.

If you can edit your */etc/hosts* file or have access to a DNS server, check *Using your own domain*, if you have no way to modify your DNS configuration, check *Using nip.io (or other DNS wildcard services)*.

Using your own domain

The defaut SSO domain is `example.com`. You can keep it for your tests or change it, for example for `mydomain.com`:

```
sed -i 's/example\.com/mydomain.com/g' \
   /etc/lemonldap-ng/* /var/lib/lemonldap-ng/conf/lmConf-1.json \
   /etc/nginx/conf.d/* \
   /etc/httpd/conf.d/* \
   /etc/apache2/sites-available/*
```

In order to be able to test, update your DNS or your local `hosts` file to map these names to the SSO server IP:

- auth.mydomain.com
- manager.mydomain.com
- test1.mydomain.com
- test2.mydomain.com

For example, you can enter the following command on your local computer: (adjust according to your server IP and test domain)

```
echo "192.168.1.30 auth.mydomain.com manager.mydomain.com test1.mydomain.com test2.
↪mydomain.com" >> /etc/hosts
```

Using nip.io (or other DNS wildcard services)

If you cannot edit /etc/hosts or your DNS zone, don't give up yet, you can use services such as nip.io, xip.io, sslip.io or others.

For example, if your server IP is 192.168.12.13, you can use 192-168-12-13.nip.io as your SSO domain:

```
sed -i 's/example\.com/192-168-12-13.nip.io/g' \
   /etc/lemonldap-ng/* /var/lib/lemonldap-ng/conf/lmConf-1.json \
   /etc/nginx/conf.d/* \
   /etc/httpd/conf.d/* \
   /etc/apache2/sites-available/*
```

Warning: nip.io, xip.io or any DNS wildcard services mentionned in this section are not affiliated with the LemonLDAP::NG project in any way. These services will receive DNS requests that will allow them to know your test server's IP address. If this is an issue for you, do not use these services.

2.3.3 Run

Starting services

Debian / Ubuntu

Enable the Nginx virtualhosts and restart the web server and LemonLDAP::NG server to apply the configuration changes

```
cd /etc/nginx/sites-enabled
ln -s ../sites-available/*nginx* .
systemctl restart lemonldap-ng-fastcgi-server
systemctl restart nginx
```

CentOS / RHEL

Enable and start httpd

```
systemctl enable httpd
systemctl start httpd
```

Open SSO session

Go on http://auth.mydomain.com and log with one of the demonstration account.

Login	Password	Role
rtyler	rtyler	user
msmith	msmith	user
dwho	dwho	administrator

Access protected application

Try http://test1.mydomain.com or http://test2.mydomain.com

Edit configuration

Log with the dwho account and go on http://manager.mydomain.com

2.4 Platforms overview

LLNG is able to use different web servers to provide its services. Here is a resume of all possibilities. We recommend:

- For installations subject to small/medium load: Nginx with our default FastCGI server, or Apache *(with mpm_prefork engine)*
- For heavily loaded installation: Nginx. The choice for FastCGI server engine depends on the behavior of your users

2.4.1 Portal/Manager installation

Since 2.0, both portal and manager are native FastCGI / PSGI Plack based applications. They can be powered by any FastCGI / PSGI compatible web servers. Some examples:

	Apache		Nginx	Plack servers family
Engines	mod_fcgid or mod_fastcgi		FastCGI/uWSGI server	Any Plack HTTP server (*see our doc*)
Link with web-server process	External processes managed by webserver *(default)*	External LLNG serve	External LLNG server	*Inside*

2.4.2 Application protection overview

Applications can be protected:

- by a LLNG handler
- by themselves if they can dial with a supported protocol (SAML, OpenID-Connect,...)

To protect applications with handler, LLNG can be used in two mode:

- Direct Application Mode : LLNG handler is an embedded application. Handler must be installed on application Web Server
- ReverseProxy Mode : applications are hidden behind a ReverseProxy which provides the required LLNG handler

Handler integration

Direct Application Mode

LLNG handlers can be installed on the following web servers:

	Apache	Nginx	Plack servers family	Node.js
Addon needed	Mod-Perl			Express
LLNG integration in webserver	*Inside*	Separate process: External LLNG FastCGI/uWSGI servers *(auth_request)*	*Inside*	Inside

ReverseProxy Mode

	Apache	Nginx
LLNG integration in ReverseProxy webserver	*Inside*	Separate process: External LLNG FastCGI/uWSGI servers

External servers for Nginx

Nginx supportes natively FastCGI and uWSGI protocoles.

Therefore, LLNG services can be provided by compatible external servers.

Tip: FastCGI or uWSGI server(s) can be installed on separate hosts. Also you can imagine a global cloud-FastCGI/uWSGI-service for all your Nginx servers. See more at *SSO as a service (SSOaaS)*.

FastCGI

By default, LLNG provides a Plack based FastCGI server able to afford all LLNG services using FCGI engine.

However, you can use some other FastCGI server engines:

- AnyEvent::FCGI
- FCGI::EV
- FCGI::Engine
- FCGI::Engine::ProcManager
- FCGI::Async
- LLNG FastCGI server for Node.js(*)

Danger: (*) LLNG Node.js handler can only be used as Nginx `auth_request` server, not to serve Portal or Manager

uWSGI

- uWSGI server (with uwsgi PSGI plugin, see *Advanced PSGI usage*)

CHAPTER

THREE

INSTALLATION

3.1 Before installation

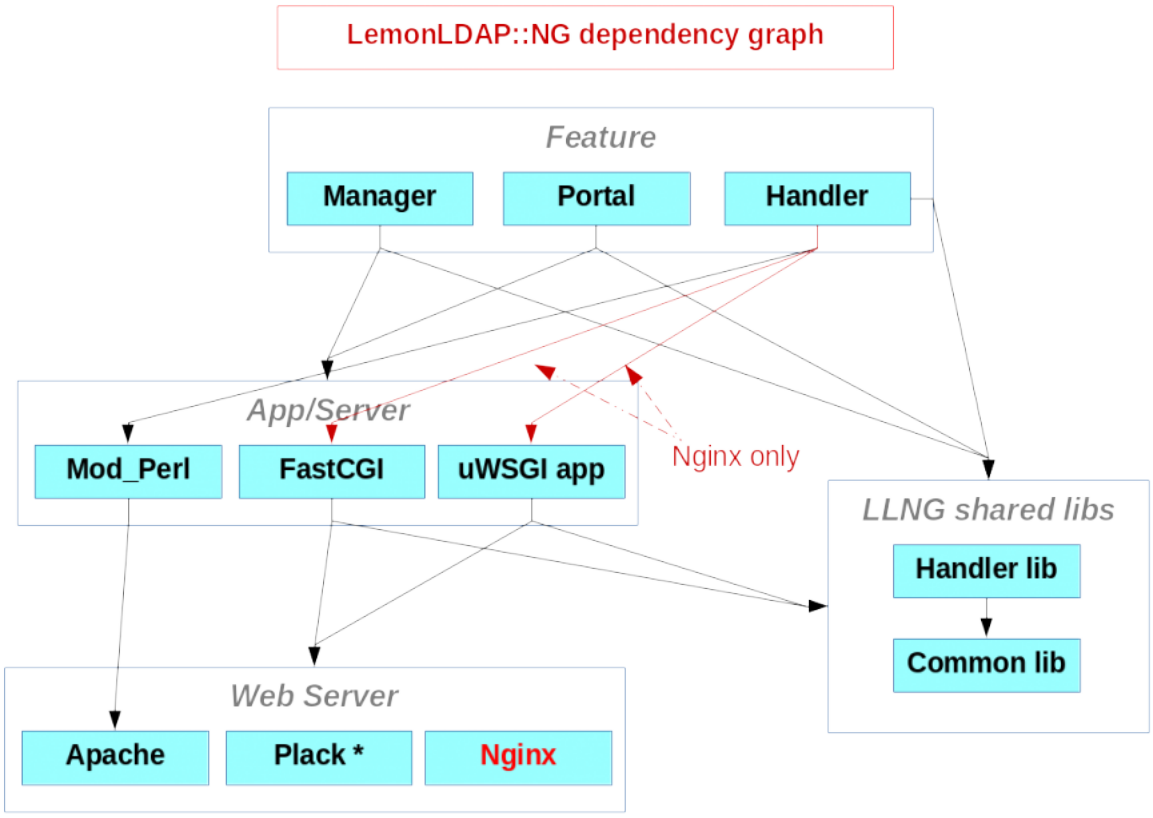
3.1.1 Prerequisites and dependencies

Web Server

To use LemonLDAP::NG, you have the choice of the Web Server :

- Nginx
- Apache 2
- Any FastCGI or uWSGI compatible Web Server (**Portal and manager only**)

For Apache2, you can use all workers mpm-worker, mpm-prefork and mpm-event. Mpm-worker works faster and LemonLDAP::NG use the thread system for best performance **but since Apache-2.4, mod_perl seems unstable in this configuration**. If you have to use mpm-prefork (for example if you use PHP), LemonLDAP::NG will work anyway.



Perl

Note: Here the list of Perl modules used in LemonLDAP::NG. Core modules must be installed on the system. Other modules are required only if you plan to use related features.

Core

- Apache::Session
- Cache::Cache
- Clone
- Config::IniFiles
- Convert::PEM
- Cookie::Baker::XS
- Crypt::OpenSSL::Bignum
- Crypt::OpenSSL::RSA
- Crypt::OpenSSL::X509
- Crypt::Rijndael

- Crypt::URandom
- DBI
- Digest::HMAC_SHA1
- Digest::MD5
- Digest::SHA
- Email::Sender
- GD::SecurityImage
- HTML::Template
- HTTP::Headers
- HTTP::Request
- IO::String
- JSON
- LWP::UserAgent
- LWP::Protocol::https
- MIME::Base64
- MIME::Entity
- Mouse
- Net::LDAP
- Plack
- Regexp::Assemble
- Regexp::Common
- SOAP::Lite *(optional)*
- String::Random
- Text::Unidecode *(Since LemonLDAP::NG 2.0.5)*
- Unicode::String
- URI
- URI::Escape

Deprecated features

- Old notifications format:
  - XML::LibXML
  - XML::LibXSLT
- OpenID 2.0:
  - Net::OpenID::Server
  - Net::OpenID::Consumer

SAML2

- Lasso
- GLib
- XML::Simple

Second factor

- Crypt::U2F::Server::Simple (U2F keys)
- Convert::Base32 (TOTP)

Specific authentication backends

- Facebook:
  - Net::Facebook::Oauth2
- Kerberos:
  - GSSAPI
- PAM:
  - Authen::PAM
- Radius:
  - Authen::Radius
- Twitter:
  - Net::OAuth
- WebID:
  - Web::ID

SMTP & Reset password/certificate by mail

- Email::Sender
- String::Random
- Net::SMTP
- Net::SSLeay
- DateTime::Format::RFC3339

Unit tests

- Authen::U2F::Tester
- Crypt::U2F::Server
- Test::MockObject
- Test::Output
- Test::POD
- Time::Fake
- YAML

Other

- Jquery (javascript framework) is included in tarball and RPMs, but is a dependency on Debian official releases
- Cache::Memcached : used by SecureToken handler

Install dependencies on your system

Danger: You don't need to install them if you use LL::NG packages. With `apt` or `yum`, dependencies will be automatically installed.

APT

Perl dependencies:

```
apt install libapache-session-perl libcache-cache-perl libclone-perl libconfig-inifiles-
↪perl libconvert-pem-perl libcrypt-openssl-bignum-perl libcrypt-openssl-rsa-perl␣
↪libcrypt-openssl-x509-perl libcrypt-rijndael-perl libdbi-perl libdigest-hmac-perl␣
↪libemail-sender-perl libgd-securityimage-perl libhtml-template-perl libio-string-perl␣
↪libjson-perl libmime-tools-perl libmouse-perl libnet-ldap-perl libplack-perl libregexp-
↪assemble-perl libregexp-common-perl libsoap-lite-perl libstring-random-perl libunicode-
↪string-perl liburi-perl libwww-perl libxml-simple-perl libxml-libxslt-perl libcrypt-
↪urandom-perl libtext-unidecode-perl libcookie-baker-xs-perl
```

For Apache:

```
apt install apache2 libapache2-mod-fcgid libapache2-mod-perl2
```

For Nginx:

```
apt install nginx nginx-extras
```

YUM

Tip: You need EPEL repository. See below how to enable this repository: http://fedoraproject.org/wiki/EPEL/FAQ#howtouse

Perl dependencies:

```
yum install perl-Apache-Session perl-Cache-Cache perl-Clone perl-Config-IniFiles perl-
↪Convert-PEM perl-Crypt-OpenSSL-RSA perl-Crypt-OpenSSL-X509 perl-Crypt-Rijndael perl-
↪Digest-HMAC perl-Digest-SHA perl-GD-SecurityImage perl-HTML-Template perl-IO-String␣
↪perl-JSON perl-LDAP perl-Mouse perl-Plack perl-Regexp-Assemble perl-Regexp-Common perl-
↪SOAP-Lite perl-String-Random perl-Unicode-String perl-version perl-XML-Simple perl-
↪Crypt-URandom perl-Email-Sender
```

For Apache:

```
yum install httpd mod_fcgid mod_perl
```

For Nginx:

```
yum install nginx
```

Attention: As you need a recent version of Nginx, the best is to install Nginx official packages.

3.1.2 Download

Release notes

Release notes for latest version: https://projects.ow2.org/view/lemonldap-ng/lemonldap-ng-2-0-9-is-out

Go on https://projects.ow2.org/bin/view/lemonldap-ng/ for older versions.

See also *upgrade notes*.

Packages and archives

Stable version (2.0.9)

Tarball

- Tarball

RPM

Tip: You can: - Use *our own YUM repository*. - Download them here and *install pre-required packages*.

RHEL/CentOS 7

- RPM bundle
- Source RPM

RHEL/CentOS 8

- RPM bundle
- Source RPM

Debian

Tip: You can:

- Use *packages provided by Debian*.
- Use *our own Debian repository*.
- Download them here and *install pre-required packages*.

- DEB bundle

Docker

See https://hub.docker.com/r/coudot/lemonldap-ng/

```
docker pull coudot/lemonldap-ng
```

Nightly builds from master branch

Debian repository of master branch, rebuilt every night: http://lemonldap-ng.ow2.io/lemonldap-ng/

Older versions

You can find all versions on OW2 releases.

Contributions

See https://github.com/LemonLDAPNG

Git repository

See https://gitlab.ow2.org/lemonldap-ng/lemonldap-ng

```
git clone git@gitlab.ow2.org:lemonldap-ng/lemonldap-ng.git
```

3.1.3 Upgrade from 2.0.x to 2.0.y

Please apply general caution as you would with any software: have backups and a rollback plan ready!

Known issues

Upgrading from 2.0.0 or 2.0.1 to later versions

If you have *installed LemonLDAP::NG from official RPMs*, you may run into bug #1757 and lose your Apache configuration files while updating from LemonLDAP::NG 2.0.0 or 2.0.1 to later versions. Please backup your `/etc/httpd/conf.d/z-lemonldap-ng-*.conf` files before the update.

Known regressions in the latest released version

None

2.0.12

Client Credential sessions missing expiration time

If you started using Client Credential grants in 2.0.11, you may have encountered issue 2481.

Because of this bug, the created sessions may never be purged by the *purgeCentralCache* script.

In order to detect these sessions, you can run the following command:

:: lemonldap-ng-sessions search –where _session_kind=SSO –select _session_id –select _utime | jq -r '. | map(select(._utime == null)) | map(._session_id) | join ("n")'

This will output a list of SSO sessions with no expiration time.

Review them manually using

```
lemonldap-ng-sessions get <session_id>
```

You can then remove them with

```
lemonldap-ng-sessions delete <session_id> <session_id> <etc.>
```

Brute-force protection plugin may cause duplicate persistent sessions

Because of bug #2482 , some users may notice that the persistent session database is filling with duplicate sessions. Some examples include:

- An uppercase version of the regular persistent session (dwho vs DWHO)
- An unqualified version (dwho vs dwho@idp.com)

This bug was fixed in 2.0.12, but administrators are advised to clean up their persistent session database to remove any duplicate persistent sessions remaining after the upgrade.

2.0.11

Portal templates changes

If you created your own skin and modified some template files, you may need to update them. No change is required if you are using the default `bootstrap` theme.

A new plugin has been introduced, in beta version: *FindUser*. It requires a modification of `login.tpl` to include `finduser.tpl`.

2.0.10

Security

A vulnerability affecting LemonLDAP::NG installations has been found out when ALL following criteria apply:

- Your handler server uses Nginx
- Your virtual host configuration contains per-URL `skip` or `unprotect` access rule

In this situation, you have to update your LUA configuration file like `/etc/nginx/nginx-lua-headers.conf`. See also issue 2434.

Other minor security fixes:

- It is now possible to hide sessions identifier in Manager (parameter `displaySessionId`). See also issue 2350.
- Second factor management by end user now requires safer conditions. See also issue 2332, issue 2337 and issue 2338.

Main changes

- New dependency: IO::Socket::Timeout
- TOTP check tolerates forward AND backward clock drift (totp2fRange)
- Avoid assignment in expressions option is disabled by default
- RHEL/CentOS SELinux users should install the new `lemonldap-ng-selinux` package to fix an issue with the new default cache directory
- If you use *Mattermost Team Edition* with OpenID Connect, you need to set the `id` claim type to *Integer*

- BruteForceProtection plugin now prevents authentication on backend if an account is locked
- In the Manager API, postLogoutRedirectUri is now returned and consumed as an array
- We fixed a bug that caused SAML sessions to be created and never deleted, you should check your session databases for sessions that have `"_session_kind": "ISAML"` but no `_utime`. You can safely delete SAML sessions with no `_utime` during the upgrade.

Portal templates changes

If you created your own skin and modified some template files, you may need to update them, see below if they have been modified.

No change is required if you are using the default `bootstrap` theme.

2FA manager

In `2fregisters.tpl` you need to add the `remove2f` class to the button that triggers second factor removal:

```
- <span device='<TMPL_VAR NAME="type">' epoch='<TMPL_VAR NAME="epoch">' class="btn btn-
↪danger" role="button">
+ <span device='<TMPL_VAR NAME="type">' epoch='<TMPL_VAR NAME="epoch">' class="btn btn-
↪danger remove2f" role="button">
```

Or, better yet, integrate the changes in `2fregisters.tpl` and `skin.min.js` into your custom theme to benefit from the new 2F removal confirmation dialog

Checkboxes

A CSS change has been done in `styles.css` to avoid checkbox labels overflow. See issue 2301.

The `form-check-input` class is missing in `register.tpl` and `notifinclude.tpl`. See issue 2374.

Password checker

Input id values have been modified in `mail.tpl` to work with password checker. See issue 2355.

Tables caption

Tables captions have been added in `sessionArray.tpl`. See issue 2356.

Stay connected

A small change is required in `checklogins.tpl` for issue 2365.

Other changes needed in `2fchoice.tpl`, `ext2check.tpl`, `totp2fcheck.tpl`, `u2fcheck.tpl` and `utotp2fcheck.tpl` for issue 2366.

Mails

The HTML `alt` attribute has been added on `img` in all `mail_*.tpl`. See issue 2422.

2.0.9

- Bad default value to display OIDC Consents tab has been fixed. The default value is now: `$_oidcConsents && $_oidcConsents =~ /\w+/`
- Some user log messages have been modified, check *logs documentation* (see also #2244)
- SAML SOAP calls are now using `text/xml` instead of `application/xml` as the MIME Content Type, as required by the SOAP standard
- Incremental lock times values can now be set in BruteForceProtection plugin through Manager. It MUST be a list of comma separated values. Default values are `5, 15, 60, 300, 600`
- This version is not compatible with *Mattermost Team Edition*

Cookie issues with Chrome

This release fixes several issues related to the change in SameSite cookie policy for Google Chrome users. The new default value of the SameSite configuration parameter will set SameSite to `Lax` unless you are using SAML, in which case it will be set to `None`.

This means that from now on, any LemonLDAP::NG installation using SAML must be served over HTTPS, as SameSite `None` value requires the `Secure` flag in cookie.

Change in default cache directory

The default config/session cache directory has been moved from `/tmp` to `/var/cache/lemonldap-ng` in order to avoid issues with cache purges when using Systemd. This change is only applied to new installations. If your installation is experiencing cache purge issues, you need to manually change your existing `localSessionStorageOptions/cache_root` parameter from `/tmp` to `/var/cache/lemonldap-ng`. Be sure to create this directory on your file system before modifying your configuration.

If you are using SELinux, you also need to run the following commands

```
semanage fcontext --add -t httpd_cache_t -f a '/var/cache/lemonldap-ng(/.*)?'
restorecon -R /var/cache/lemonldap-ng/
```

Required changes in NGINX handler rules (CVE-2020-24660)

We discovered a vulnerability that affects LemonLDAP::NG installations when ALL of the following criteria apply:

- You are using the *LemonLDAP::NG Handler* to protect applications
- Your handler server uses Nginx
- Your virtual host configuration contains per-URL access rules based on regular expressions in addition to the built-in *default* access rule.

Note: You are safe from this vulnerability if your virtualhost only uses a regexp-based rule to trigger logout

If you are in this situation, you need to modify *all* your handler-protected virtualhosts by making the following change:

- Replace `fastcgi_param X_ORIGINAL_URI $request_uri` by `fastcgi_param X_ORIGINAL_URI $original_uri` if you are using FastCGI
- Replace `uwsgi_param X_ORIGINAL_URI $request_uri` by `uwsgi_param X_ORIGINAL_URI $original_uri` if you are using uWSGI
- Right after `auth_request /lmauth;`, add the following line

```
set $original_uri $uri$is_args$args;
```

You can check the *Manage virtual hosts* page for more information

LDAP certificate validation (CVE-2020-16093)

LDAP server certificates were previously not verified by default when using secure transports (LDAPS or TLS), see CVE-2020-16093. Starting from this release, certificate validation is now enabled by default, including on existing installations.

If you have configured your CA certificates incorrectly, LemonLDAP::NG will now start complaining about invalid certificates. You may temporarily disable it again with the following command

```
/your/path/to/lemonldap-ng-cli set ldapVerify none
```

If you use LDAP as a configuration storage, and want to temporarily disable certificate validation, you must make the following addition to */etc/lemonldap-ng/lemonldap-ng.ini*

```
[configuration]
...
ldapVerify = none
```

If you use LDAP as a session backend, you are strongly encouraged to also upgrade corresponding `Apache::Session` modules (`Apache::Session::LDAP` or `Apache::Session::Browseable`). After this upgrade, if you want to temporarily disable certificate validation, you can add the following parameter to the list of Apache::Session module options:

- key: `ldapVerify`
- value: `none`

Please note that it is HIGHLY recommended to set certificate validation to *require* when contacting LDAP servers over a secure transport to avoid man-in-the-middle attacks.

2.0.8

- New dependency: Perl module Time::Fake is now required to run unit test and build packages, but should not be mandatory to run the software.
- Nginx configuration: some changes are required to allow IPv6, see #2152
- Option `singleSessionUserByIP` was removed, see #2159
- A memory leak was found in perl-fcgi with Perl < 5.18, a workaround is possible with Apache and llng-fastcgi-server, see #1314
  - With Apache: set `FcgidMaxRequestsPerProcess 500` in portal virtual host
  - With llng-fastcgi-server: set `PM_MAX_REQUESTS=500` in llng-fastcgi-server service configuration
- Cookie `SameSite` value: to avoid problems with recent browsers, SAML POST binding, LLNG cookies are now tagged as "**SameSite=None**". You can change this value using manager, "**SameSite=Lax**" is best for installations without federations. **Important note**: if you're using an unsecured connection *(http:// instead of https://)*, "SameSite=None" will be ignored by browsers and users that already have a valid session might be prompted to login again.
- OAuth2.0 Handler: a VHost protected by the OAuth2.0 handler will now return a 401 when called without an Access Token, instead of redirecting to the portal, as specified by RFC6750
- If you encounter the following issue:

```
AH01630: client denied by server configuration: /usr/share/lemonldap-ng/manager/api/api.
↪fcgi
```

when trying to access the portal. It probably comes from incorrect Apache configuration. Remove the (optional and disabled by default) manager API config:

```
rm /etc/httpd/conf.d/z-lemonldap-ng-api.conf && systemctl reload httpd
```

2.0.7

- Security:
  - #2040: Configuration of a redirection URI for an OpenID Connect Relying Party is now mandatory, as defined in the specifications. If you save your configuration, you will have an error if some of your RP don't have a redirect URI configured.
  - #1943 / CVE-2019-19791: along with the patch provided in 2.0.7 in `Lemonldap/NG/Common/PSGI/Request.pm`, Apache rewrite rule must be updated to avoid an unprotected access to REST services:

```
portal-apache2.conf
```

```
RewriteCond "%{REQUEST_URI}" "!^/(?:(?:static|javascript|favicon).*|.*\.fcgi(?:/.*)?)$"
RewriteRule "^/(.+)$" "/index.fcgi/$1" [PT]
```

```
manager-apache2.conf
```

```
RewriteCond "%{REQUEST_URI}" "!^/(?:static|doc|lib|javascript|favicon).*"
RewriteRule "^/(.+)$" "/manager.fcgi/$1" [PT]
```

- Other:

  - Option `checkTime` was enabled by default in `lemonldap-ng.ini`, this let the portal check the configuration immediately instead of waiting for configuration cache expiration. You can keep this option enabled unless you need strong *performances*.

- Removed parameters:
  - `samlIdPResolveCookie`

2.0.6

- Option was added to display generate password box in *password reset by mail plugin*. If you use this feature, you must enable this option, which is disabled by default.
- If you use the default _whatToTrace macro and a case insensitive authentication backend, then a user can generate several persistent sessions for the same login (see issue 1869). This can lead to a security bug if you enabled 2FA, which rely on data stored in the persistent session. To fix this, either choose a unique attribute for _whatToTrace, either force lower case in your macro:

```
$_auth eq 'SAML' ? lc($_user.'@'.$_idpConfKey) : $_auth eq 'OpenIDConnect' ? lc($_user.'@
↪'.$_oidc_OP) : lc($_user)
```

- On CentOS 7 / RHEL 7, a system upgrade breaks ImageMagick, which is used to display captchas (see #1951). To fix this, you can run the following commands:

```
yum install -y urw-base35-fonts-legacy
sed 's,/usr/share/fonts/default/Type1/,/usr/share/X11/fonts/urw-fonts/,g' -i /etc/
↪ImageMagick/type-ghostscript.xml
```

2.0.5

- The Text::Unidecode perl module becomes a requirement *(it will be automatically installed if you upgrade from from the deb or RPM repositories)*
- CAS logout starts validating the service= parameter, but only if you use the CAS Access control policy. The URL sent in the service= parameter will be checked against *known CAS applications*, Virtual Hosts, and *trusted domains*. Add your target domain to trusted domains if you suddenly start having "Invalid URL" messages on logout
- Improvements in cryptographic functions: to take advantage of them, **you must change the encryption key** of LemonLDAP::NG (see *CLI example*).
- Debian packaging: FastCGI / uWsgi servers require llng-lmlog.conf and llng-lua-headers.conf. Those configuration files are now provided by lemonldap-ng-handler package and installed in /etc/nginx/snippets directory.

3.1.4 Upgrade from 1.9 to 2.0

> **Attention:** 2.0 is a major release, lot of things have been changed. You must read this document before upgrade.

Upgrade order from 1.9.*

As usual, if you use more than 1 server and don't want to stop SSO service AND IF YOU HAVE NO INCOMPATIBILITY MENTIONED IN THIS DOCUMENT, upgrade must be done in the following order:

1. servers with handlers only;
2. portal servers *(all together if your load balancer is stateless (user or client IP) and if users use the menu)*;
3. manager server

Attention: You must revalidate your configuration using the manager.

Installation

Attention: French documentation is no more available. Only English version of this documentation is maintained now.

This release of LL::NG requires these minimal versions of GNU/Linux distributions:

- Debian 9 (stretch)
- Ubuntu 16.04 LTS
- CentOS 7
- RHEL 7

For SAML features, we require at least Lasso 2.5 and we recommend Lasso 2.6.

Configuration

- **lemonldap-ng.ini** requires some new fields in portal section. Update yours using the one given installed by default. New requires fields are:
  - **staticPrefix** *(manager and portal)*: the path to static content
  - **templateDir** *(manager and portal)*: the path to templates directory
  - **languages** *(manager and portal)*: accepted languages
- Portal skins are now in `/usr/share/lemonldap-ng/portal/templates`. See *skin customization* to adapt your templates.
- User module in authentication parameters now provides a "Same as authentication" value. You must revalidate it in the manager since all special values must be replaced by this *(Multi, Choice, Proxy, Slave, SAML, OpenID,...)**
- **"Multi" doesn't exist anymore**: it is replaced by *Combination*, a more powerful module.
- Apache and Nginx configurations must be updated to use FastCGI portal
- URLs for mail reset and register pages have changed, you must update configuration parameters. For example:

```
mailUrl => 'http://auth.example.com/resetpwd',
registerUrl => 'http://auth.example.com/register',
```

- Option `trustedProxies` was removed, you must now configure your Web Server to manage `X-Forwarded-For` header, see *how to run LL::NG behind a reverse proxy*.

Attention: Apache mod_perl has got lot of troubleshooting problems since 2.4 version (many segfaults,...), especially when using MPM worker or MPM event. That's why LL::NG doesn't use anymore ModPerl::Registry: all is now handled by FastCGI (portal and manager), except for Apache2 Handler.

For Handlers, it is now recommended to migrate to Nginx, but Apache 2.4 is still supported with MPM prefork.

Configuration refresh

Now portal has the same behavior than handlers: it looks to configuration stored in local cache every 10 minutes. So it has to be reload like every handler.

Attention: If you want to use reload mechanism on a portal only host, you must install a handler in Portal host to be able to refresh local cache. Include `handler-nginx.conf` or `handler-apache2.conf` for example

LDAP connection

Now LDAP connections are kept open to improve performances. To allow that, LL::NG requires an anonymous access to LDAP RootDSE entry to check connection.

Kerberos or SSL usage

- A new *Kerberos* authentication backend has been added since 2.0. This module solves many Kerberos integration problems *(usage in conjunction with other backends, better error display,...)*. However, you can retain the old integration manner (using *Apache authentication module*).
- For *SSL*, a new *Ajax option* can be used in the same idea: so SSL can be used in conjunction with other backends.

Logs

- **Syslog**: logs are now configured in `lemonldap-ng.ini` file only. If you use Syslog, you must reconfigure it. See *logs* for more.
- **Apache2**: Portal doesn't use anymore Apache2 logger. Logs are always written to Apache error.log but Apache "LogLevel" parameter has no more effect on it. Portal is now a FastCGI application and doesn't use anymore ModPerl. See *logs* for more.
- If you are running behind a proxy, make sure LemonLDAP::NG can *see the original IP address* of incoming HTTP connections

Security

LLNG portal now embeds the following features:

- CSRF protection *(Cross-Site Request Forgery)*: a token is build for each form. To disable it, set requireToken to 0 *(portal security parameters in the manager)*
- Content-Security-Policy header: portal build dynamically this header. You can modify default values in the manager *(Général parameters » Advanced parameters » Security » Content-Security-Policy)*

Handlers

- **Apache only**:
  - **Apache handler** is now Lemonldap::NG::Handler::ApacheMP2 and Menu is now Lemonldap::NG::Handler::ApacheMP2::Menu
  - because of an Apache behaviour change, PerlHeaderParserHandler must no more be used with "reload" URLs *(replaced by PerlResponseHandler)*. Any "reload url" that are inside a protected vhost must be unprotected in vhost rules *(protection has to be done by web server configuration)*.
- *CDA*, *ZimbraPreAuth*, *SecureToken* and *AuthBasic* are now *Handler Types*. So there is no more special file to load: you just have to choose "VirtualHost type" in the manager/VirtualHosts.
- *SSOCookie*: Since Firefox 60 and Chrome 68, "+2d, +5M, 12h and so on…" cookie expiration time notation is no more supported. CookieExpiration value is a number of seconds until the cookie expires. A zero or negative number will expire the cookie immediately.

Rules and headers

- hostname() and remote_ip() are no more provided to avoid some name conflicts *replaced by `$ENV{}`)*
- *$ENV{<cgi_variable>}* is now available everywhere: see *Writing rules and headers*
- some variable names have changed. See *Variables* document

Opening conditions

- Rule and message fields have been swaped. You have to modifiy and validate again your access rules.

Supported servers

- Apache-1.3 files are not provided now. You can build them yourself by looking at Apache-2 configuration files

Ajax requests

Before 2.0, an Ajax query launched after session timeout received a 302 code. Now a 401 HTTP code is returned. `WWW-Authenticate` header contains: `SSO <portal-URL>`

SOAP/REST services

- SOAP server activation is now split in 2 parameters (configuration/sessions). You must set them else SOAP service will be disabled
- Notifications are now REST/JSON by default. You can force old format in the manager. Note that SOAP proxy has changed: http://portal/notifications now.
- If you use "adminSessions" endpoint with "singleSession*" features, you must upgrade all portals simultaneously
- SOAP services can be replaced by new REST services

Attention: *AuthBasic Handler* uses now REST services instead of SOAP.

CAS

CAS authentication module no more use perl CAS client, but our own code. You can now define several CAS servers in a specific branch in Manager, like you can define several SAML or OpenID Connect providers.

CAS issuer module has also been improved, you must modify the configuration of CAS clients to move them from virtual host branch to CAS client branch.

Developer corner

APIs

Portal has now many REST features and includes an API plugin. See Portal manpages to learn how to write auth modules, issuers or other features.

Portal overview

Portal is no more a single CGI object. Since 2.0, It is based on Plack/PSGI and Mouse modules. Little resume

```
Portal object
  |
  +-> auth module
  |
  +-> userDB module
  |
  +-> issuer modules
  |
  +-> other plugins (notification,...)
```

Requests are independent objects based on Lemonldap::NG::Portal::Main::Request which inherits from Lemonldap::NG::Common::PSGI::Request which inherits from Plack::Request. See manpages for more.

Handler

Handler libraries have been totally rewritten. If you've made custom handlers, they must be rewritten, see *customhandlers*.

If you used self protected CGI, you also need to rewrite them, see *documentation*.

3.2 Main installation

3.2.1 Installation from the tarball

Get the tarball

Get the tarball from *download page*. You can also find on this page the SVN tarball if you want to test latest features.

Attention: The content of the SVN tarball is not the same as the official tarball. Please see the next chapter to learn how build an official tarball from SVN files.

Build the tarball from SVN

Either checkout or export the SVN repository, or extract the SVN tarball to get the SVN files on your disk.

Then go to trunk directory:

```
cd trunk
```

And run the "dist" target:

```
make dist
```

The generated tarball is in the current directory.

Extraction

Just run the tar command:

```
tar zxvf lemonldap-ng-*.tar.gz
```

Installation

First check and install the *prerequisites*.

For full install:

```
cd lemonldap-ng-*
make configure
make
make test
sudo make install PROD=yes
```

Note: `PROD=yes` makes web interface use minified versions of CSS and JS files.

You can modify location of default storage configuration file in configure target:

```
make configure STORAGECONFFILE=/etc/lemonldap-ng/lemonldap-ng.ini
```

You can choose other Makefile targets:

- Perl libraries install :
  - install_libs (all Perl libraries)
  - install_portal_libs
  - install_manager_libs
  - install_handler_libs
- Binaries install :
  - install_bin (/usr/local/lemonldap-ng/bin)
- FastCGI server install (required for Nginx)
  - install_fastcgi_server (/usr/local/lemonldap-ng/sbin)

- Web sites install :
  - install_site (all sites including install_doc_site)
  - install_portal_site (/usr/local/lemonldap-ng/htdocs/portal)
  - install_manager_site (/usr/local/lemonldap-ng/htdocs/manager)
  - install_handler_site (/usr/local/lemonldap-ng/handler)
- Documentation install :
  - install_doc_site (/usr/local/lemonldap-ng/htdocs/doc)
  - install_examples_site (/usr/local/lemonldap-ng/examples)

You can also pass parameters to the make install command, with this syntax:

```
sudo make install PARAM=VALUE PARAM=VALUE ...
```

Available parameters are:

- **ERASECONFIG**: set to 0 if you want to keep your configuration files (default: 1)
- **DESTDIR**: only for packaging, install the product in a jailroot (default: "")
- **PREFIX**: installation directory (default: /usr/local)
- **CRONDIR**: Cronfile directory (default: $PREFIX/etc/lemonldap-ng/cron.d)
- **APACHEUSER**: user running Apache
- **APACHEGROUP**: group running Apache
- **DNSDOMAIN**: Main DNS domain (default: example.com)
- **APACHEVERSION**: Apache major version (default: 2)
- **VHOSTLISTEN**: how listen parameter is configured for virtual hosts in Apache (default: *:80)
- **PROD**: use minified JS and CSS files
- **USEDEBIANLIBS**: use Debian packaged JS and CSS files (**Note that this options isn't yet usable** since Debian provides a too old AngularJS for now: LLNG manager needs at least version 1.4.0)
- **USEEXTERNALLIBS**: use files from public CDN
- **STORAGECONFFILE**: *make configure* target only. Location of default storage configuration file (default: /usr/local/lemonldap-ng/etc/lemonldap-ng.ini)

Tip: For Debian/Ubuntu with Apache2, you can use:

```
make debian-install-for-apache
make ubuntu-install-for-apache
```

And with Nginx:

```
make debian-install-for-nginx
make ubuntu-install-for-nginx
```

See also *Debian/Ubuntu installation documentation*.

Install cron jobs

LL::NG use cron jobs (or systemd timers) to:

- purge old sessions
- clean Handler cache

To install them on system:

```
sudo ln -s /usr/local/lemonldap-ng/etc/cron.d/* /etc/cron.d/
```

or install .timers files in systemd directory (/lib/systemd/system)

DNS

Configure your DNS server to resolve names with your server IP:

- auth.<your domain>: main portal, must be public
- manager.<your domain>: manager, only for adminsitrators
- test1.<your domain>, test2.<your domain>: sample applications

Follow the *next steps*.

3.2.2 Installation on Debian/Ubuntu with packages

Organization

LemonLDAP::NG provides these packages:

- lemonldap-ng: metapackage, contains no file but dependencies on other packages
- lemonldap-ng-doc: contains HTML documentation and project docs (README, etc.)
- lemonldap-ng-fastcgi-server: LL::NG FastCGI server (for Nginx)
- lemonldap-ng-handler: Handler files
- liblemonldap-ng-common-perl: configuration and common files
- liblemonldap-ng-handler-perl: Handler common libraries
- liblemonldap-ng-manager-perl: Manager files
- liblemonldap-ng-portal-perl: Portal files

Get the packages

Official repository

If you run Debian stable, testing or unstable, the packages are directly installable:

```
apt-get install lemonldap-ng
```

Tip: Packages from Debian repository may not be up to date but are **security-maintained** by Debian Security Team for "stable" release and LTS team for "oldstable" release. Then if you don't need some new features or aren't concerned by a bug fixed earlier, **this is a good choice**. You can also use Debian backports or "testing"/"unstable" packages, team maintained. Here is the list of Debian versions.

Danger: LLNG Ubuntu packages are not in the "universe" but in the "multiverse". This means they are not security-maintained. If you use them, you should follow our security advisories on lemonldap-ng-users@ow2.org.

LL::NG repository

You can add this repository to have recent packages.

First, make sure your system can install packages from HTTPS repositories:

```
apt install apt-transport-https
```

You will need to trust the following GPG key :

```
wget -O - https://lemonldap-ng.org/_media/rpm-gpg-key-ow2 | apt-key add -
```

Then, add the official LL::NG repository

```
vi /etc/apt/sources.list.d/lemonldap-ng.list
```

```
# LemonLDAP::NG repository
deb     https://lemonldap-ng.org/deb stable main
deb-src https://lemonldap-ng.org/deb stable main
```

Tip:

- Use the `oldstable` repository to get packages from previous major version
- Use the `testing` repository to get packages from next major version
- Use the `2.0` repository to avoid upgrade to next major version

Finally update your APT cache:

```
apt update
```

Manual download

Packages are available on the *Download page*.

Install packages

Attention: By default packages will require Nginx. If you want to use Apache2, install it first with mod_perl:

```
apt install apache2 libapache2-mod-perl2 libapache2-mod-fcgid
```

With apt

```
apt install lemonldap-ng
```

With dpkg

Before installing the packages, install *dependencies*.

Then:

```
dpkg -i liblemonldap-ng-* lemonldap-ng*
```

First configuration steps

Change default DNS domain

By default, DNS domain is `example.com`. You can change it quick with a sed command. For example, we change it to `ow2.org`:

```
sed -i 's/example\.com/ow2.org/g' /etc/lemonldap-ng/* /var/lib/lemonldap-ng/conf/lmConf-
↪1.json
```

Upgrade

If you upgraded LL::NG, check all *upgrade notes*.

DNS

Configure your DNS server to resolve names with your server IP:

- auth.<your domain>: main portal, must be public
- manager.<your domain>: manager, only for adminsitrators
- test1.<your domain>, test2.<your domain>: sample applications

Follow the *next steps*

File location

- Configuration is in /etc/lemonldap-ng
- LemonLDAP::NG configuration (edited by the Manager) is in /var/lib/lemonldap-ng/conf/
- All Perl modules are in the VENDOR perl directory (/usr/share/perl5/)
- All Perl scripts/pages are in /var/lib/lemonldap-ng/
- All lemonldap-ng tools are in /usr/share/lemonldap-ng/bin/
- All static content (examples, CSS, images, etc.) is in /usr/share/lemonldap-ng/
- Apache configuration files are in /etc/lemonldap-ng and linked in /etc/apache2/sites-available and /etc/nginx/sites-available

Build your packages

You can also get the *LemonLDAP::NG archive* and make the package yourself:

```
tar xzf lemonldap-ng-*.tar.gz
cd lemonldap-ng-*
make debian-packages
```

3.2.3 Installation on Red Hat/CentOS

Attention: LL::NG requires at least Red Hat/CentOS 7

Organization

LemonLDAP::NG provides packages for Red Hat/Centos 7:

- lemonldap-ng: metapackage, contains no file but dependencies on other packages
- lemonldap-ng-doc: contains HTML documentation and project docs (README, etc.)
- lemonldap-ng-conf: contains default configuration (DNS domain: example.com)
- lemonldap-ng-test: contains sample CGI test page
- lemonldap-ng-handler: contains Apache Handler implementation (agent)
- lemonldap-ng-manager: contains administration interface and session explorer
- lemonldap-ng-portal: contains authentication portal and menu

- lemonldap-ng-fastcgi-server: FastCGI server needed to use Nginx
- lemonldap-ng-nginx: contains Nginx configuration and dependencies
- lemonldap-ng-uwsgi-app: contains Uwsgi application
- lemonldap-ng-selinux: contains the SELinux policy for httpd
- perl-Lemonldap-NG-Common: CPAN - Shared modules
- perl-Lemonldap-NG-Handler: CPAN - Handler modules
- perl-Lemonldap-NG-Manager: CPAN - Manager modules
- perl-Lemonldap-NG-Portal: CPAN - Portal modules

Danger: The package lemonldap-ng-nginx requires the nginx community package. If you use openrestry or Nginx plus, you must ignore this dependency. To do this, download the package and install it with:

```
rpm --nodeps -i lemonldap-ng-nginx*.rpm
```

Get the packages

YUM repository

You can add this YUM repository to get recent packages:

```
vi /etc/yum.repos.d/lemonldap-ng.repo
```

```
[lemonldap-ng]
name=LemonLDAP::NG packages
baseurl=https://lemonldap-ng.org/redhat/stable/$releasever/noarch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-OW2
```

Tip: Replace `stable` by `2.0` to avoid upgrade to next major version

You may also need some extras packages, available here:

```
[lemonldap-ng-extras]
name=LemonLDAP::NG extra packages
baseurl=https://lemonldap-ng.org/redhat/extras/$releasever
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-OW2
```

Run this to update packages cache:

```
yum update
```

Danger: You must also install the EPEL repository for non-core dependencies. See *prerequisites and dependencies* chapter for more.

Manual download

RPMs are available on the *Download page*.

Package GPG signature

The GPG key can be downloaded here:

Install it to trust RPMs:

```
curl https://lemonldap-ng.org/_media/rpm-gpg-key-ow2 > /etc/pki/rpm-gpg/RPM-GPG-KEY-OW2
```

Install packages

With YUM

If the packages are stored in a yum repository:

```
yum install lemonldap-ng

# If you use SELinux
yum install lemonldap-ng-selinux
```

You can also use yum on local RPMs file:

```
yum localinstall lemonldap-ng-* perl-Lemonldap-NG-*
```

With RPM

Before installing the packages, install all *dependencies*.

You have then to install all the downloaded packages:

```
rpm -Uvh lemonldap-ng-* perl-Lemonldap-NG-*
```

Tip: You can choose to install only one component by choosing the package `lemonldap-ng-portal`, `lemonldap-ng-handler` or `lemonldap-ng-manager`.

Install the package `lemonldap-ng-conf` on all server which contains one of those packages.

First configuration steps

Change default DNS domain

By default, DNS domain is `example.com`. You can change it quick with a sed command. For example, we change it to `ow2.org`:

```
sed -i 's/example\.com/ow2.org/g' /etc/lemonldap-ng/* /var/lib/lemonldap-ng/conf/lmConf-
↪1.json /etc/nginx/conf.d/* /etc/httpd/conf.d/*
```

Upgrade

If you upgraded LL::NG, check all *upgrade notes*.

DNS

Configure your DNS server to resolve names with your server IP:

- auth.<your domain>: main portal, must be public
- manager.<your domain>: manager, only for adminsitrators
- test1.<your domain>, test2.<your domain>: sample applications

Follow the *next steps*

File location

- Configuration is in /etc/lemonldap-ng
- LemonLDAP::NG configuration (edited by the Manager) is in /var/lib/lemonldap-ng/conf/
- All Perl modules are in the VENDOR perl directory
- All Perl scripts/pages are in /var/lib/lemonldap-ng/
- All static content (examples, CSS, images, etc.) is in /usr/share/lemonldap-ng/

Build your packages

If you need it, you can rebuild RPMs:

- Install rpm-build package
- Install all build dependencies (see BuildRequires in lemonldap-ng.spec)
- Put LemonLDAP::NG tarball in %_topdir/SOURCES
- Edit ~/.rpmmacros and set your build parameters:

```
%_topdir /home/user/build
%dist .el7
%rhel 7
```

- Go to %_topdir
- Build:

```
rpmbuild -ta SOURCES/lemonldap-ng-VERSION.tar.gz
```

3.2.4 Installation on Suse Linux

Attention: LL::NG requires at least SLES 12 SP1 or equivalent

Organization

LemonLDAP::NG provides packages for SLES:

- lemonldap-ng: metapackage, contains no file but dependencies on other packages
- lemonldap-ng-doc: contains HTML documentation and project docs (README, etc.)
- lemonldap-ng-conf: contains default configuration (DNS domain: example.com)
- lemonldap-ng-test: contains sample CGI test page
- lemonldap-ng-handler: contains Apache Handler implementation (agent)
- lemonldap-ng-manager: contains administration interface and session explorer
- lemonldap-ng-portal: contains authentication portal and menu
- lemonldap-ng-fastcgi-server: FastCGI server needed to use Nginx
- perl-Lemonldap-NG-Common: CPAN - Shared modules
- perl-Lemonldap-NG-Handler: CPAN - Handler modules
- perl-Lemonldap-NG-Manager: CPAN - Manager modules
- perl-Lemonldap-NG-Portal: CPAN - Portal modules

Get the packages

Repositories

This manual only refers to SLES 12 SP1. Installation may work on other platforms, with no guarantee.

Different repositories are necessary for LemonLDAP::NG dependencies:

- Suse official repositories
- 2 repositories on openSUSE Build Service
- Additional packages available on repository.linagora.org or lemonldap-ng.org
- Suse SDK repository is advised for building packages (yast2 -> Software -> Software Repositories -> Add –> Extensions and modules from Registration Server)

First, make sure the exploitation system is up to date:

```
zypper update
```

You can add the openSUSE Build Service repositories with the following commands:

```
zypper addrepo http://download.opensuse.org/distribution/leap/42.1/repo/oss/suse/ leap42
zypper addrepo http://download.opensuse.org/repositories/devel:languages:perl/SLE_12/
↪devel:languages:perl.repo
zypper refresh
```

Accept both signing keys each time.

You can add the additional dependency repository *and* the LemonLDAP::NG repository with either commands:

```
zypper addrepo http://lemonldap-ng.org/sles12 lemonldap-sles12-repository
zypper refresh
```

or

```
zypper addrepo http://repository.linagora.org/lemonldap-sles12-repository lemonldap-
↪sles12-repository
zypper refresh
```

Tip: Only packages on SLES 12 SP1 are tested for now.

Manual download

RPMs are available on the *Download page*.

Package GPG signature

The GPG key can be downloaded here:

Install it to trust RPMs:

```
rpm --import rpm-gpg-key-ow2
```

Install packages

With ZYPPER

If the packages are stored in a repository:

```
zypper install lemonldap-ng
```

```
59 new packages to install.
Total download size: 13.5 MiB. Already cached : 0 B. After operation, 30.7 MiB of␣
↪supplementary disk space will be used.
Continue ? [y/n/? print all options] (y):
```

You can also use zypper on local RPMs file:

```
zypper install lemonldap-ng-* perl-Lemonldap-NG-*
```

With RPM

Before installing the packages, install all dependencies: (you need to get dependencies from previous repositories)

```
zypper install apache2 apache2-mod_perl apache2-mod_fcgid perl-ldap perl-XML-SAX perl-
↪XML-NamespaceSupport perl-XML-Simple perl-XML-LibXML perl-Config-IniFiles perl-Digest-
↪HMAC perl-Crypt-OpenSSL-RSA perl-Authen-SASL perl-Unicode-String gd perl-Regexp-
↪Assemble perl-Authen-Captcha perl-Cache-Cache perl-Apache-Session perl-CGI-Session␣
↪perl-IO-String perl-MIME-Lite perl-SOAP-Lite perl-XML-LibXSLT perl-String-Random perl-
↪Email-Date-Format perl-Crypt-Rijndael perl-HTML-Template perl-JSON perl-Crypt-OpenSSL-
↪X509 perl-Crypt-DES perl-Class-Inspector perl-Test-MockObject perl-Clone perl-Net-CIDR-
↪Lite perl-ExtUtils-MakeMaker perl-CGI perl-CGI-Session perl-HTML-Template perl-SOAP-
↪Lite perl-IPC-ShareLite perl-Error perl-HTML-Parser perl-libwww-perl perl-DBI perl-
↪Cache-Memcached perl-Class-ErrorHandler perl-Convert-PEM perl-Crypt-DES_EDE3 perl-
↪Digest-SHA perl-Env perl-Mouse perl-String-CRC32 perl-Plack perl-Regexp-Common perl-
↪Crypt-OpenSSL-Bignum perl-FCGI-ProcManager
```

You have then to install all the downloaded packages:

```
rpm -Uvh lemonldap-ng-* perl-Lemonldap-NG-*
```

Tip: You can choose to install only one component by choosing the package `lemonldap-ng-portal`, `lemonldap-ng-handler` or `lemonldap-ng-manager`.

Install the package `lemonldap-ng-conf` on all server which contains one of those packages.

First configuration steps

Enable Apache extensions

These extensions are activated by default on Apache at LemonLDAP install:

```
a2enmod perl
a2enmod headers
a2enmod mod_fcgid
a2enmod ssl
a2enmod rewrite
a2enmod proxy
a2enmod proxy_http
```

If you decide to use SSL, you should also activate the appopriate flag:

```
sed -i 's/^APACHE_SERVER_FLAGS=.*/APACHE_SERVER_FLAGS="SSL"/' /etc/sysconfig/apache2
```

Change default DNS domain

By default, DNS domain is `example.com`. You can change it quick with a sed command. For example, we change it to `ow2.org`:

```
sed -i 's/example\.com/ow2.org/g' /etc/lemonldap-ng/{*.conf,*.ini,for_etc_hosts} /var/
↪lib/lemonldap-ng/conf/lmConf-1
```

Check Apache configuration and restart:

```
apachectl configtest
apachectl restart
```

DNS

Configure your DNS server to resolve names with your server IP:

- auth.<your domain>: main portal, must be public
- manager.<your domain>: manager, only for adminsitrators
- test1.<your domain>, test2.<your domain>: sample applications

Follow the *next steps*

File location

- Configuration is in /etc/lemonldap-ng
- LemonLDAP::NG configuration (edited by the Manager) is in /var/lib/lemonldap-ng/conf/
- All Perl modules are in the VENDOR perl directory
- All Perl scripts/pages are in /var/lib/lemonldap-ng/
- All static content (examples, CSS, images, etc.) is in /usr/share/lemonldap-ng/

Build your packages

If you need it, you can rebuild RPMs:

- Install rpm-build package
- Get the lemonldap source package from repository:

```
zypper source-install lemonldap-ng
cd /usr/src/packages/
ls SPECS/ SOURCES/
```

- Install all build dependencies (see BuildRequires in lemonldap-ng.spec)
- Build:

```
rpmbuild -ba SPECS/lemonldap-ng.spec
```

Alternatively, you can use the automatic script "create-lemonldap-packages.sh", available in rpm-sles directory in the *lemonldap svn repository*. The automatic script can also generate intermediate dependencies. See README file in the same directory for more information.

3.2.5 LemonLDAP::NG in Docker

Presentation

Docker allows do run application into containers.

You can find a Docker image for LemonLDAP::NG in this repository: https://hub.docker.com/r/coudot/lemonldap-ng/

See also this github project: https://github.com/LemonLDAPNG/lemonldap-ng-docker

Usage

Prerequisites:

- Add auth.example.com/manager.example.com/test1.example.com/test2.example.com to /etc/hosts on the host

```
echo "127.0.0.1 auth.example.com manager.example.com test1.example.com test2.example.com
↪" | sudo tee -a /etc/hosts
```

- Map the container port 80 to host port 80 (option -p)

```
docker run -d -p 80:80 coudot/lemonldap-ng
```

Then connect to http://auth.example.com with your browser and log in with dwho/dwho.

3.2.6 Node.js handler

Since version 2.0, a beta Node.js handler is available on GitHub and NPMJS.

Up-to-date documentation is available on GitHub.

Examples

Important things:

- Rules and headers must be written in javascript for these hosts (example `$uid eq "dwho"` becomes `$uid === "dwho"`)
- Multi-lines are not supported in lemonldap-ng.ini
- Virtualhosts handled by node-lemonldap-ng-handler must be explicitly declared in your `lemonldap-ng.ini` file in `[node-handler]` section (**NB**: section `[handler]` isn't used by node handler):

```
[node-handler]

nodeVhosts = test.example.com, test2.example.com
```

Use it as FastCGI server (application protection only)

FastCGI server

```
var handler = require('lemonldap-ng-handler');

handler.init({
  configStorage: {
    "confFile": "/path/to/lemonldap-ng.ini"
  }
});

handler.nginxServer({
  "mode": "fcgi",   // or "http", default: fcgi
  "port": 9090,     // default value
  "ip": 'localhost' // default value
});
```

Nginx configuration

```
server {
  #...
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass localhost:9090;

    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";

    # Keep original hostname
    fastcgi_param HOST $http_host;
```

(continues on next page)

(continued from previous page)

```
    # Keep original request (LLNG server will receive /lmauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    include conf/nginx-lua-headers.conf;
  }
}
```

Use it to protect an express app

```
// Variables
var express = require('express');
var app = express();
var handler = require('lemonldap-ng-handler');

// initialize handler (optional args)
handler.init({
  configStorage: {
    "confFile":"test/lemonldap-ng.ini"
  }
});

// and load it
app.use(handler.run);

// Then simply use your express app
app.get('/', function(req, res) {
  return res.send('Hello ' + req.headers['Auth-User'] + ' !');
});
app.listen(3000, function() {
  return console.log('Example app listening on port 3000!');
});
```

3.3 After installation

3.3.1 Deploy Nginx configuration

FastCGI server

To use Nginx, you must install LemonLDAP::NG FastCGI server or use `llngapp.psgi` *(provided in examples)* with a PSGI server. See *Advanced PSGI usage*.

Debian/Ubuntu

```
apt install lemonldap-ng-fastcgi-server
```

Enable and start the service :

```
systemctl enable llng-fastcgi-server
systemctl start llng-fastcgi-server
```

Red Hat/CentOS

```
yum install lemonldap-ng-nginx lemonldap-ng-fastcgi-server
```

Enable and start the service :

```
systemctl enable llng-fastcgi-server
systemctl start llng-fastcgi-server
```

Files

With tarball installation, Nginx configuration files will be installed in `/usr/local/lemonldap-ng/etc/`, else they are directly in web server configuration.

Debian/Ubuntu

- Install log format *(automatically loaded when linked in this place)*

```
ln -s /etc/lemonldap-ng/nginx-lmlog.conf /etc/nginx/conf.d/llng-lmlog.conf
```

- Install snippet for vhost configuration files:

```
ln -s /etc/lemonldap-ng/nginx-lua-headers.conf /etc/nginx/snippets/llng-lua-headers.conf
```

- Enable sites:

```
ln -s /etc/nginx/sites-available/handler-nginx.conf /etc/nginx/sites-enabled/
ln -s /etc/nginx/sites-available/manager-nginx.conf /etc/nginx/sites-enabled/
ln -s /etc/nginx/sites-available/portal-nginx.conf /etc/nginx/sites-enabled/
ln -s /etc/nginx/sites-available/test-nginx.conf /etc/nginx/sites-enabled/
```

3.3.2 Deploy Apache configuration

Note: This step should have been already done if you installed LL::NG with packages.

Files

Attention: Apache Mod Perl has many issues since 2.4 version with MPM worker and MPM event. No problem for portal and manager since they are now handled by FastCGI. If you want to use Apache for Handler, please switch to MPM prefork, else use Nginx.

With tarball installation, Apache configuration files will be installed in `/usr/local/lemonldap-ng/etc/`, else they are in `/etc/lemonldap-ng`.

You have to include them in Apache main configuration, for example:

```
include /usr/local/lemonldap-ng/etc/portal-apache2.conf
include /usr/local/lemonldap-ng/etc/handler-apache2.conf
include /usr/local/lemonldap-ng/etc/manager-apache2.conf
include /usr/local/lemonldap-ng/etc/test-apache2.conf
```

Tip:

- You can also use symbolic links in `conf.d` or `sites-available` Apache directory.
- If you have run the Debian/Ubuntu install command, just use:

```
a2ensite manager-apache2.conf
a2ensite portal-apache2.conf
a2ensite handler-apache2.conf
a2ensite test-apache2.conf
```

Modules

You will also need to load some Apache modules:

- mod_rewrite
- mod_perl
- mod_alias
- mod_fcgid
- mod_headers

Tip: With Debian/Ubuntu:

```
a2enmod fcgid perl alias rewrite headers
```

3.3.3 Deploy LemonLDAP::NG on a Plack server

Plack is a powerful engine that powers many very fast servers. LLNG uses some Plack libraries to run as FastCGI server. So, It can be easily run on these servers. See also *Advanced PSGI usage* if you want to replace LLNG FastCGI server.

Complete example

```
#!/usr/bin/perl

use Data::Dumper;
use Plack::Builder;

# Basic test app
my $testApp = sub {
    my ($env) = @_;
    return [
        200,
        [ 'Content-Type' => 'text/plain' ],
        [ "Hello LLNG world\n\n" . Dumper($env) ],
    ];
};

# Build protected app
my $test = builder {
    enable "Auth::LemonldapNG";
    $testApp;
};

# Build portal app
use Lemonldap::NG::Portal::Main;
my $portal = builder {
    enable "Plack::Middleware::Static",
      path => '^/static/',
      root => '/path/to/portal/htdocs/';
    Lemonldap::NG::Portal::Main->run( {} );
};

# Build manager app
use Lemonldap::NG::Manager;
my $manager = builder {
    enable "Plack::Middleware::Static",
      path => '^/static/',
      root => '/path/to/manager/htdocs/';
    enable "Plack::Middleware::Static",
      path => '^/doc/',
      root => '/path/to/dir/that/contains/"doc"';
    enable "Plack::Middleware::Static",
      path => '^/lib/',
      root => '/path/to/doc/pages/documentation/current/';
    Lemonldap::NG::Manager->run( {} );
};
```

(continues on next page)

(continued from previous page)

```
# Global app
builder {
    mount 'http://test1.example.com/'   => $test;
    mount 'http://auth.example.com/'    => $portal;
    mount 'http://manager.example.com/' => $manager;
};
```

Launch it with Starman for example:

```
$ starman --port 80 --workers 32 llapp.psgi
```

CHAPTER

FOUR

CONFIGURATION FIRST STEPS

4.1 Configuration overview

4.1.1 Backends

LemonLDAP::NG configuration is stored in a backend that allows all modules to access it.

Attention: Note that all LL::NG components must have access:

- to the configuration backend
- to the sessions storage backend

Detailed configuration backends documentation is available *here*.

By default, configuration is stored in *files*, so access through network is not possible. To allow this, use *SOAP* for configuration access, or use a network service like *SQL database* or *LDAP directory*.

Configuration backend can be set in the *local configuration file*, in `configuration` section.

For example, to configure the `File` configuration backend:

```
[configuration]
type=File
dirName = /usr/local/lemonldap-ng/data/conf
```

Tip: See *How to change configuration backend* to known how to change this.

4.1.2 Manager

Most of configuration can be done through LemonLDAP::NG Manager (by default http://manager.example.com).

By default, Manager is protected to allow only the demonstration user "dwho".

Attention: This user will not be available anymore if you configure a new authentication backend! Remember to change the access rule in Manager virtual host to allow new administrators.

If you can not access the Manager anymore, you can unprotect it by editing `lemonldap-ng.ini` and changing the `protection` parameter:

```
[manager]

# Manager protection: by default, the manager is protected by a demo account.
# You can protect it :
# * by Apache itself,
# * by the parameter 'protection' which can take one of the following
# values :
#   * authenticate : all authenticated users can access
#   * manager      : manager is protected like other virtual hosts: you
#                    have to set rules in the corresponding virtual host
#   * rule: <rule> : you can set here directly the rule to apply
#   * none         : no protection
```

Tip: See *Manager protection documentation* to know how to use Apache modules or LL::NG to manage access to Manager.

The Manager displays main branches:

- **General Parameters**: Authentication modules, portal, etc.
- **Variables**: User information, macros and groups used to fill SSO session
- **Virtual Hosts**: Access rules, headers, etc.
- **SAML 2 Service**: SAML metadata administration
- **SAML identity providers**: Registered IDP
- **SAML service providers**: Registered SP
- **OpenID Connect Service**: OpenID Connect service configuration
- **OpenID Connect Providers**: Registered OP
- **OpenID Connect Relying Parties**: Registered RP

LemonLDAP::NG configuration is mainly a key/value structure, so Manager will present all keys into a structured tree. A click on a key will display the associated value.

When all modifications are done, click on `Save` to store configuration.

Danger: LemonLDAP::NG will do some checks on configuration and display errors and warnings if any. Configuration **is not saved** if errors occur.

Tip:

- *Configuration viewer* allow some users to edit WebSSO configuration in Read Only mode.
- You can set and display instance name in Manager menu by editing `lemonldap-ng.ini` in [manager] section:

```
[manager]
instanceName = LLNG_Demo
```

4.1.3 Manager API

Since 2.0.8, a Manager API is available for:

- Second factors management for users
- OpenID Connect RP management
- SAML SP management

See Manager API documentation.

Attention: To access Manager API, enable the `manager-api` virtual host and change the access rule. You can protect the API through Basic authentication, IP white list or any other condition.

4.1.4 Configuration text editor

LemonLDAP::NG provide a script that allows one to edit configuration without graphical interface, this script is called `lmConfigEditor` and is stored in the LemonLDAP::NG bin/ directory, for example /usr/share/lemonldap-ng/bin:

- On Debian:

```
/usr/share/lemonldap-ng/bin/lmConfigEditor
```

- On CentOS:

```
/usr/libexec/lemonldap-ng/bin/lmConfigEditor
```

Tip: This script must be run as root, it will then use the Apache user and group to access configuration.

The script uses the `editor` system command, that links to your favorite editor. To change it:

```
update-alternatives --config editor
```

The configuration is displayed as a big Perl Hash, that you can edit:

```
$VAR1 = {
          'ldapAuthnLevel' => '2',
          'notificationWildcard' => 'allusers',
          'loginHistoryEnabled' => '1',
          'key' => 'q`e)kJE%<&wm>uaA',
          'samlIDPSSODescriptorSingleSignOnServiceHTTPPost' =>
↪'urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST;#PORTAL#/saml/singleSignOn;',
          'portalSkin' => 'pastel',
          'failedLoginNumber' => '5',
          ...
          };
```

If a modification is done, the configuration is saved with a new configuration number. Else, current configuration is kept.

4.1.5 Command Line Interface (CLI)

LemonLDAP::NG provide a script that allows one to edit configuration items in non interactive mode. This script is called `lemonldap-ng-cli` and is stored in the LemonLDAP::NG bin/ directory, for example /usr/share/lemonldap-ng/bin:

- On Debian:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli
```

- On CentOS:

```
/usr/libexec/lemonldap-ng/bin/lemonldap-ng-cli
```

Tip: This script must be run as root, it will then use the Apache user and group to access configuration.

To see available actions, do:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli help
```

You can force an update of configuration cache with:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli update-cache
```

To get information about current configuration:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli info
```

To view a configuration parameter, for example portal URL:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli get portal
```

To set a parameter, for example domain:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli set domain example.org
```

To delete a parameter, for example portalSkinBackground:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli del portalSkinBackground
```

Tip: Use addKey and delKey actions to manage values of hash configuration parameters

You can use accessors (options) to change the behavior:

- -sep: separator of hierarchical values (by default: /).
- -iniFile: the lemonldap-ng.ini file to use if not default value.
- -yes: do not prompt for confirmation before saving new configuration.
- -cfgNum: the configuration number. If not set, it will use the latest configuration.
- -force: set it to 1 to save a configuration earlier than latest.

Some examples:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -cfgNum 10 get exportedHeaders/test1.
↪example.com
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 set notification 1
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -sep ',' get macros,_whatToTrace
```

Tip: See *other examples*.

4.1.6 Apache

Attention: LemonLDAP::NG does not manage Apache configuration

LemonLDAP::NG ships 3 Apache configuration files:

- **portal-apache2.conf**: Portal virtual host, with SOAP/REST end points
- **manager-apache2.conf**: Manager virtual host
- **handler-apache2.conf** : Handler declaration, reload virtual hosts
- **test-apache2.conf** : Example protected virtual hosts

See *how to deploy them*.

Portal

After enabling any REST/SOAP endpoints in the Manager, you also need to configure some for of authentication on the corresponding URLs in the **portal-apache2.conf** configuration file.

By default, access to those URLs is denied:

```
# REST/SOAP functions for sessions management (disabled by default)
<Location /index.fcgi/adminSessions>
    Order deny,allow
    Deny from all
</Location>
```

Allowing configuration reload

In order to allow configuration reload from a different server (if your manager is on a different server or if you are using load-balancing), you need to edit the access rule in **handler-apache2.conf**

```
<Location /reload>
    #CHANGE THIS######
    Require ip 127 ::1
    ###########^^^^^^^
    SetHandler perl-script
    PerlResponseHandler Lemonldap::NG::Handler::ApacheMP2->reload
</Location>
```

Handler

In order to protect your application VHosts with the LemonLDAP::NG handler, you need to add these directives:

- Load Handler in Apache memory:

(in a global configuration file)

```
PerlOptions +GlobalRequest
PerlModule Lemonldap::NG::Handler::ApacheMP2
```

- Catch error pages:

```
ErrorDocument 403 http://auth.example.com/lmerror/403
ErrorDocument 404 http://auth.example.com/lmerror/404
ErrorDocument 500 http://auth.example.com/lmerror/500
ErrorDocument 502 http://auth.example.com/lmerror/502
ErrorDocument 503 http://auth.example.com/lmerror/503
```

Then, to protect a standard virtual host, the only configuration line to add is:

```
PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2
```

See **test-apache2.conf** for a complete example of a protected application

4.1.7 Nginx

> **Attention:** LemonLDAP::NG does not manage Nginx configuration

LemonLDAP::NG ships 3 Nginx configuration files:

- **portal-nginx.conf**: Portal virtual host, with REST/SOAP end points
- **manager-nginx.conf**: Manager virtual host
- **handler-nginx.conf** : Handler reload virtual hosts
- **test-nginx.conf** : Example protected application

See *how to deploy them*.

> **Danger:** *LL::NG FastCGI* server must be enabled and started separately.

Portal

After enabling any REST/SOAP endpoints in the Manager, you also need to configure some for of authentication on the corresponding URLs in the **portal-nginx.conf** configuration file.

By default, access to those URLs is denied:

```
location ~ ^/index.psgi/adminSessions {
  fastcgi_pass llng_portal_upstream;
  deny all;
}
```

Allowing configuration reload

In order to allow configuration reload from a different server (if your manager is on a different server or if you are using load-balancing), you need to edit the access rule in **handler-nginx.conf**

```
location = /reload {

  ## CHANGE THIS #
  allow 127.0.0.1;
  ######^^^^^^^^^^#

  deny all;

  # FastCGI configuration
  include /etc/nginx/fastcgi_params;
  fastcgi_pass unix:__FASTCGISOCKDIR__/llng-fastcgi.sock;
  fastcgi_param LLTYPE reload;
}
```

Handler

Nginx handler is provided by the *LemonLDAP::NG FastCGI server*.

- Handle errors:

```
error_page 403 http://auth.example.com/lmerror/403;
error_page 404 http://auth.example.com/lmerror/404;
error_page 500 http://auth.example.com/lmerror/500;
error_page 502 http://auth.example.com/lmerror/502;
error_page 503 http://auth.example.com/lmerror/503;
```

To protect a standard virtual host, you must insert this (or create an included file):

```
# Insert $_user in logs
include /etc/lemonldap-ng/nginx-lmlog.conf;
access_log /var/log/nginx/access.log lm_combined;

# Internal call to FastCGI server
location = /lmauth {
  internal;
  include /etc/nginx/fastcgi_params;
  fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
  fastcgi_pass_request_body  off;
  fastcgi_param CONTENT_LENGTH "";
  fastcgi_param HOST $http_host;
  fastcgi_param X_ORIGINAL_URI  $original_uri;
}

# Client requests
location / {
  auth_request /lmauth;
  set $original_uri $uri$is_args$args;
  auth_request_set $lmremote_user $upstream_http_lm_remote_user;
```

(continues on next page)

(continued from previous page)

```
auth_request_set $lmlocation $upstream_http_location;
error_page 401 $lmlocation;
try_files $uri $uri/ =404;

# Set REMOTE_USER (for FastCGI apps only)
#fastcgi_param REMOTE_USER $lmremote_user

##################################
# PASSING HEADERS TO APPLICATION #
##################################

# IF LUA IS SUPPORTED
#include /path/to/nginx-lua-headers.conf

# ELSE
# Set manually your headers
#auth_request_set $authuser $upstream_http_auth_user;
#proxy_set_header Auth-User $authuser;
# OR
#fastcgi_param HTTP_AUTH_USER $authuser;

# Then (if LUA not supported), change cookie header to hide LLNG cookie
#auth_request_set $lmcookie $upstream_http_cookie;
#proxy_set_header Cookie: $lmcookie;
# OR
#fastcgi_param HTTP_COOKIE $lmcookie;

# Insert then your configuration (fastcgi_* or proxy_*)
```

4.1.8 Configuration reload

Note: As Handlers keep configuration in cache, when configuration change, it should be updated in Handlers. An Apache restart will work, but LemonLDAP::NG offers the mean to reload them through an HTTP request. Configuration reload will then be effective in less than 10 minutes. If you want to change this timeout, set `checkTime = 240` in your lemonldap-ng.ini file *(values in seconds)*

After configuration is saved by Manager, LemonLDAP::NG will try to reload configuration on distant Handlers by sending an HTTP request to the servers. The servers and URLs can be configured in Manager, `General Parameters > reload configuration URLs`: keys are server names or IP the requests will be sent to, and values are the requested URLs.

You also have a parameter to adjust the timeout used to request reload URLs, it is be default set to 5 seconds.

Attention: If "Compact configuration file" option is enabled, all useless parameters are removed to limit file size. Typically, if SAMLv2 service is disabled, all relative parameters will be erased. To avoid useless parameters to be purged, you can disable this option.

These parameters can be overwritten in LemonLDAP::NG ini file, in the section `apply`.

Tip: You only need a reload URL per physical servers, as Handlers share the same configuration cache on each physical server.

The `reload` target is managed in Apache or Nginx configuration, inside a virtual host protected by LemonLDAP::NG Handler (see below examples in Apache->handler or Nginx->Handler).

Attention: You must allow access to declared URLs to your Manager IP.

Attention: If reload URL is served in HTTPS, to avoid "Error 500 (certificate verify failed)", Go to :

`General Parameters > Advanced Parameters > Security > SSL options for server requests`

and set :

verify_hostname => 0

SSL_verify_mode => 0

Attention: If you want to use reload mechanism on a portal only host, you must install a handler in Portal host to be able to refresh local cache. Include `handler-nginx.conf` or `handler-apache2.conf` for example

Practical use case: configure reload in a LL::NG cluster. In this case you will have two servers (with IP 1.1.1.1 and 1.1.1.2), but you can keep only one reload URL (reload.example.com):

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 addKey \
reloadUrls '1.1.1.1' 'http://reload.example.com/reload' \
reloadUrls '1.1.1.2' 'http://reload.example.com/reload'
```

You also need to adjust the protection of the reload vhost, for example:

```
<Location /reload>
    Require ip 127 ::1 1.1.1.1 1.1.1.2
    SetHandler perl-script
    PerlResponseHandler Lemonldap::NG::Handler::ApacheMP2->reload
</Location>
```

4.1.9 Local file

LemonLDAP::NG configuration can be managed in a local file with INI format. This file is called `lemonldap-ng.ini` and has the following sections:

- **configuration**: where configuration is stored
- **apply**: reload URL for distant Hanlders
- **all**: parameters for all modules
- **portal**: parameters only for Portal
- **manager**: parameters only for Manager

- **handler**: parameters only for Handler

When you set a parameter in `lemonldap-ng.ini`, it will override the parameter from the global configuration.

For example, to override configured skin for portal:

```
[portal]
portalSkin = dark
```

Tip: You need to know the technical name of configuration parameter to do this. You can refer to *parameter list* to find it.

4.2 Single Sign On cookie, domain and portal URL

4.2.1 SSO cookie

The SSO cookie is built by the portal (as described in the *login kinematic*), or by the Handler for cross domain authentication (see *CDA kinematic*).

To edit SSO cookie parameters, go in Manager, `General Parameters` > `Cookies`:

- **Cookie name**: name of the cookie, can be changed to avoid conflicts with other LemonLDAP::NG installations
- **Domain**: validity domain for the cookie (the cookie will not be sent on other domains)
- **Multiple domains**: enable *cross domain mechanism* (without this, you cannot extend SSO to other domains)
- **Secured cookie**: 4 options:
  - **Non secured cookie**: the cookie can be sent over HTTP and HTTPS connections
  - **Secured cookie**: the cookie can only be sent over HTTPS
  - **Double cookie**: two cookies are delivered, one for HTTP and HTTPS connections, the other for HTTPS only
  - **Double cookie for single session**: same as double cookie but only one session is created in session database
- **Javascript protection**: set httpOnly flag, to prevent cookie from being leaked by malicious javascript code
- **Cookie expiration time**: by default, SSO cookie is a session cookie, which means it will be destroyed when browser is closed. You can change this behavior by setting a cookie expiration time. It must be an integer. **Cookie Expiration Time** value is a number of seconds until the cookie expires. Set a zero value to disable expiration time and use a session cookie.
- **Cookie SameSite value**: the value of the SameSite cookie attribute. By default, LemonLDAP::NG will set it to "Lax" in most cases, and "None" if you use federated authentiication like SAML or OIdC. Using "None" requires Secured Cookies, and accessing applications over HTTPS on most web browsers.

Danger: When you change cookie expiration time, it is written on the user hard disk unlike session cookie

Attention: Changing the domain value will not update other configuration parameters, like virtual host names, portal URL, etc. You have to update them by yourself.

4.2.2 Portal URL

Portal URL is the address used to redirect users on the authentication portal by:

- **Handler**: user is redirected if he has no SSO cookie (or in *CDA* mode)
- **Portal**: the portal redirect on itself in many cases (credentials POST, SAML, etc.)

Danger: The portal URL **must** be inside SSO domain. If secured cookie is enabled, the portal URL **must** be HTTPS.

4.3 Redirections

4.3.1 Handler Redirections

Note: When a user access a Handler without a cookie, he is redirected on portal, and the target URL is encoded in redirection URL (to redirect user after authentication process).

Protocol and port

To encode the redirection URL, the handler will use some Apache environment variables and also configuration settings:

- **HTTPS**: use https as protocol
- **Port**: port of the application (by default, 80 for http, 443 for https)

These parameters can be configured in Manager, in `General Parameters` > `Advanced parameters` > `Handler redirections`.

Tip: These settings can be overridden per virtual host, see *virtual host management*.

Forbidden and Server error

Handler use the default Apache error code for the following cases:

- User has no access authorization: FORBIDDEN (403)
- An error occurs on server side: SERVER_ERROR (500)
- The application is in maintenance: HTTP_SERVICE_UNAVAILABLE (503)

These errors can be catch through Apache `ErrorDocument` directive or Nginx `error_page` directive, to redirect user on a specific page:

```
# Apache: Common error page and security parameters
ErrorDocument 403 http://auth.example.com/?lmError=403
ErrorDocument 500 http://auth.example.com/?lmError=500
ErrorDocument 503 http://auth.example.com/?lmError=503
```

```
# Nginx: Common error page and security parameters
error_page 403 http://auth.example.com/?lmError=403;
error_page 500 http://auth.example.com/?lmError=500;
error_page 503 http://auth.example.com/?lmError=503;
```

It is also possible to redirect the user without using `ErrorDocument`: the Handler will not return 403, 500, 503 code, but code 302 (REDIRECT).

The user will be redirected on portal URL with error in the `lmError` URL parameter.

These parameters can be configured in Manager, in `General Parameters` > `Advanced parameters` > `Handler redirections`:

- **Redirect on forbidden**: use 302 instead 403
- **Redirect on error**: use 302 instead 500 or 503

4.3.2 Portal Redirections

Note: If a user is redirected from handler to portal for authentication and once he is authenticated, portal redirects him to the redirection URL.

- **Redirection message**: The redirection from portal can be done either with code 303 (See Other), or with a JavaScript redirection. Often the redirection takes some time because it is user's first access to the protected app, so a new app session has to be created : JavaScript redirection improves user experience by informing that authentication is performed, and by preventing from clicking again on the button because it is too slow.
- **Keep redirections for Ajax**: By default, when an Ajax request is done on the portal for an unauthenticated user (after a redirection done by the handler), a 401 code will be sentwith a `WWW-Authenticate` header containing "SSO <portal-URL>". Set this option to 1 to keep the old behavior (return of HTML code).
- **Skip re-auth confirmation**: by default, when re-authentication is needed, a confirmation screen is displayed to let user accept the re-authentication. If you enable this option, user will be directly redirected to login page.

4.4 Exported variables

4.4.1 Presentation

Exported variables are the variables available to *write rules and headers*. They are extracted from the users database by the *users module*.

To create a variable, you've just to map a user attributes in LL::NG using `Variables` » `Exported variables`. For each variable, the first field is the name which will be used in rules, macros or headers and the second field is the name of the user database field.

Examples for *LDAP*:

Variable name	LDAP attribute
uid	uid
number	employeeNumber
name	sn

You can define exported variables for each module in the module configuration itself. Variables defined in the main `Exported variables` will be used for each backend. Variables defined in the exported variables node of the module will be used only for that module.

Tip: You can define environment variables in `Exported variables`, this allows one to populate user session with some environment values. Environment variables will not be queried in users database.

4.4.2 Extend variables using macros and groups

Macros and groups are calculated during authentication process by the portal:

- macros are used to extend (or rewrite) *exported variables*. A macro is stored as attributes: it can contain boolean results or any string
- macros can also be used to import environment variables *(these variables are in CGI format)*. Example: `$ENV{HTTP_COOKIE}`
- groups are stored as a string with values separated by ‘’; ‘’ (default values separator) in the special attribute `groups`: it contains the names of groups whose rules were returned true for the current user. For example:

```
$groups = group3; admin
```

- You can also get groups in `$hGroups` which is a Hash Reference of this form:

```
$hGroups = {
          'group3' => {
                        'description' => [
                                           'Service 3',
                                           'Service 3 TEST'
                                         ],
                        'cn' => [
                                  'group3'
                                ],
                        'name' => 'group3'
                      },
          'admin' => {
                       'name' => 'admin'
                     }
        }
```

Example for macros:

```
# boolean macro
isAdmin -> $uid eq 'foo' or $uid eq 'bar'
# other macro
```

(continues on next page)

(continued from previous page)

```
displayName -> $givenName." ".$surName

# Use a boolean macro in a rule
^/admin -> $isAdmin
# Use a string macro in a HTTP header
Display-Name -> $displayName
```

Defining a group for admins

```
# group
admin -> $uid eq 'foo' or $uid eq 'bar'
```

Using groups in a rule

```
^/admin -> $groups =~ /\badmin\b/

# Or with hGroups
^/admin -> defined $hGroups->{'admin'}

# Since 2.0.8
^/admin -> inGroup('admin')
```

Note: Groups are computed after macros, so a group rule may involve a macro value.

Warning: Macros and groups are computed in alphanumeric order, that is, in the order they are displayed in the manager. For example, macro "macro1" will be computed before macro "macro2": so, expression of macro2 may involve value of macro1. As same for groups: a group rule may involve another, previously computed group.

4.5 Manage virtual hosts

LemonLDAP::NG configuration is build around Apache or Nginx virtual hosts. Each virtual host is a protected resource, with access rules, headers, POST data and options.

4.5.1 Apache configuration

To protect a virtual host in Apache, the LemonLDAP::NG Handler must be activated (see *Apache global configuration*).

Then you can take any virtual host, and simply add this line to protect it:

```
PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2
```

Hosted application

Example of a protected virtual host for a local application:

```
<VirtualHost *:80>
        ServerName localsite.example.com

        PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2

        DocumentRoot /var/www/localsite

        ErrorLog /var/log/apache2/localsite_error.log
        CustomLog /var/log/apache2/localsite_access.log combined

</VirtualHost>
```

Reverse proxy

Example of a protected virtual host with LemonLDAP::NG as reverse proxy:

```
<VirtualHost *:80>
        ServerName application.example.com

        PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2

        # Reverse-Proxy
        ProxyPass / http://private-name/
        # Change "Location" header in redirections
        ProxyPassReverse / http://private-name/
        # Change domain cookies
        ProxyPassReverseCookieDomain private-name application.example.com

        ErrorLog /var/log/apache2/proxysite_error.log
        CustomLog /var/log/apache2/proxysite_access.log combined
</VirtualHost>
```

Same with remote server configured with the same host name:

```
<VirtualHost *:80>
        ServerName application.example.com

        PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2

        # Reverse-Proxy
        ProxyPass / http://APPLICATION_IP/

        ProxyPreserveHost on

        ErrorLog /var/log/apache2/proxysite_error.log
        CustomLog /var/log/apache2/proxysite_access.log combined
</VirtualHost>
```

Note: The `ProxyPreserveHost` directive will forward the Host header to the protected application. To learn more about using Apache as reverse-proxy, see Apache documentation.

Tip: Some applications need the `REMOTE_USER` environment variable to get the connected user, which is not set in reverse-proxy mode. In this case, see *how convert header into environment variable*.

Add a floating menu

A little floating menu can be added to application with this simple Apache configuration:

```
PerlModule Lemonldap::NG::Handler::ApacheMP2::Menu
PerlOutputFilterHandler Lemonldap::NG::Handler::ApacheMP2::Menu->run
```

Pages where this menu is displayed can be restricted, for example:

```
<Location /var/www/html/index.php>
PerlOutputFilterHandler Lemonldap::NG::Handler::ApacheMP2::Menu->run
</Location>
```

Attention: You need to disable mod_deflate to use the floating menu

4.5.2 Nginx configuration

To protect a virtual host in Nginx, the LemonLDAP::NG FastCGI server must be launched (see *LemonLDAP::NG FastCGI server*).

Then you can take any virtual host and modify it:

- Declare the /lmauth endpoint

```
location = /lmauth {
  internal;
  include /etc/nginx/fastcgi_params;
  fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;

  # Drop post datas
  fastcgi_pass_request_body  off;
  fastcgi_param CONTENT_LENGTH "";

  # Keep original hostname
  fastcgi_param HOST $http_host;

  # Keep original request (LLNG server will receive /lmauth)
  fastcgi_param X_ORIGINAL_URI $original_uri;
}
```

- Protect the application (/ or /path/to/protect):

```
location /path/to/protect {
  auth_request /lmauth;
  set $original_uri $uri$is_args$args;
  auth_request_set $lmremote_user $upstream_http_lm_remote_user;
  auth_request_set $lmlocation $upstream_http_location;
  auth_request_set $cookie_value $upstream_http_set_cookie;
  add_header Set-Cookie $cookie_value;
  error_page 401 $lmlocation;
  try_files $uri $uri/ =404;

  # ...
}
```

- Use LUA or set manually the headers:

```
location /path/to/protect {

  # ...

  # IF LUA IS SUPPORTED
  #include /etc/lemonldap-ng/nginx-lua-headers.conf;

  # ELSE
  # Set manually your headers
  #auth_request_set $authuser $upstream_http_auth_user;
  #proxy_set_header Auth-User $authuser;
  # OR
  #fastcgi_param HTTP_AUTH_USER $authuser;

  # Then (if LUA not supported), change cookie header to hide LLNG cookie
  #auth_request_set $lmcookie $upstream_http_cookie;
  #proxy_set_header Cookie: $lmcookie;
  # OR in the corresponding block
  #fastcgi_param HTTP_COOKIE $lmcookie;

  # Set REMOTE_USER (for FastCGI apps only)
  #fastcgi_param REMOTE_USER $lmremote_user;
}
```

Hosted application

Example of a protected virtual host for a local application:

```
# Log format
include /path/to/lemonldap-ng/nginx-lmlog.conf;
server {
  listen 80;
  server_name myserver;
  root /var/www/html;
  # Internal authentication request
  location = /lmauth {
    internal;
```

(continues on next page)

(continued from previous page)

```
    include /etc/nginx/fastcgi_params;
    fastcgi_pass /path/to/llng-fastcgi-server.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will receive /lmauth)
    fastcgi_param X_ORIGINAL_URI $original_uri;
  }

  # Client requests
  location ~ \.php$ {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;
    include fastcgi_params;
    try_files $fastcgi_script_name =404;
    fastcgi_pass /path/to/php-fpm/socket;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_intercept_errors on;
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_hide_header X-Powered-By;

    ###################################
    # PASSING HEADERS TO APPLICATION #
    ###################################
    # IF LUA IS SUPPORTED
    #include /path/to/nginx-lua-headers.conf

    # ELSE
    # Set manually your headers
    #auth_request_set $authuser $upstream_http_auth_user;
    #fastcgi_param HTTP_AUTH_USER $authuser;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Reverse proxy

- Example of a protected reverse-proxy:

```
# Log format
include /path/to/lemonldap-ng/nginx-lmlog.conf;
server {
  listen 80;
  server_name myserver;
  root /var/www/html;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass /path/to/llng-fastcgi-server.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will receive /lmauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;

    proxy_pass http://remote.server/;
    include /etc/nginx/proxy_params;

    ##################################
    # PASSING HEADERS TO APPLICATION #
    ##################################
    # IF LUA IS SUPPORTED
    #include /path/to/nginx-lua-headers.conf;

    # ELSE
    # Set manually your headers
    #auth_request_set $authuser $upstream_http_auth_user;
    #proxy_set_header HTTP_AUTH_USER $authuser;
  }
}
```

If /etc/nginx/proxy_params file does not exist, you can create it with this content:

```
proxy_set_header Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

- Example of a Nginx Virtual Host using uWSGI with many URIs protected by different types of handler:

```
# Log format
include /path/to/lemonldap-ng/nginx-lmlog.conf;
server {
  listen 80;
  server_name myserver;
  root /var/www/html;

 # Internal MAIN handler authentication request
  location = /lmauth {
    internal;
    # uWSGI Configuration
    include /etc/nginx/uwsgi_params;
    uwsgi_pass 127.0.0.1:5000;
    uwsgi_pass_request_body  off;
    uwsgi_param CONTENT_LENGTH "";
    uwsgi_param HOST $http_host;
    uwsgi_param X_ORIGINAL_URI  $original_uri;
    # Improve performances
    uwsgi_buffer_size 32k;
    uwsgi_buffers 32 32k;
  }

  # Internal AUTH_BASIC handler authentication request
  location = /lmauth-basic {
    internal;
    # uWSGI Configuration
    include /etc/nginx/uwsgi_params;
    uwsgi_pass 127.0.0.1:5000;
    uwsgi_pass_request_body  off;
    uwsgi_param CONTENT_LENGTH "";
    uwsgi_param HOST $http_host;
    uwsgi_param X_ORIGINAL_URI  $original_uri;
    uwsgi_param VHOSTTYPE AuthBasic;
    # Improve performances
    uwsgi_buffer_size 32k;
    uwsgi_buffers 32 32k;
  }

  # Internal SERVICE_TOKEN handler authentication request
  location = /lmauth-service {
    internal;
    # uWSGI Configuration
    include /etc/nginx/uwsgi_params;
    uwsgi_pass 127.0.0.1:5000;
    uwsgi_pass_request_body  off;
    uwsgi_param CONTENT_LENGTH "";
    uwsgi_param HOST $http_host;
    uwsgi_param X_ORIGINAL_URI  $original_uri;
    uwsgi_param VHOSTTYPE ServiceToken;
    # Improve performances
    uwsgi_buffer_size 32k;
```

(continues on next page)

(continued from previous page)

```
    uwsgi_buffers 32 32k;
  }

  # Client requests
  location / {
    ###################################
    # CALLING AUTHENTICATION          #
    ###################################
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmremote_custom $upstream_http_lm_remote_custom;
    auth_request_set $lmlocation $upstream_http_location;
    # Remove this for AuthBasic handler
    error_page 401 $lmlocation;

    ###################################
    # PASSING HEADERS TO APPLICATION #
    ###################################
    # IF LUA IS SUPPORTED
    include /etc/nginx/nginx-lua-headers.conf;
  }

  location /AuthBasic/ {
    ###################################
    # CALLING AUTHENTICATION          #
    ###################################
    auth_request /lmauth-basic;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmremote_custom $upstream_http_lm_remote_custom;
    auth_request_set $lmlocation $upstream_http_location;
    # Remove this for AuthBasic handler
    #error_page 401 $lmlocation;

    ###################################
    # PASSING HEADERS TO APPLICATION #
    ###################################
    # IF LUA IS SUPPORTED
    include /etc/nginx/nginx-lua-headers.conf;
  }

  location /web-service/ {
    ###################################
    # CALLING AUTHENTICATION          #
    ###################################
    auth_request /lmauth-service;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    # Remove this for AuthBasic handler
    error_page 401 $lmlocation;
```

(continues on next page)

(continued from previous page)

```
    ##################################
    # PASSING HEADERS TO APPLICATION #
    ##################################
    # IF LUA IS SUPPORTED
    include /etc/nginx/nginx-lua-headers.conf;
  }
}
```

4.5.3 LemonLDAP::NG configuration

A virtual host protected by LemonLDAP::NG Handler must be registered in LemonLDAP::NG configuration.

To do this, use the Manager, and go in `Virtual Hosts` branch. You can add, delete or modify a virtual host here. Enter the exact virtual host name (for example `test.example.com`) or use a wildcard (for example `*.example.com`).

A virtual host contains:

- Access rules: check user's right on URL patterns
- HTTP headers: forge information sent to protected applications
- POST data: use form replay
- Options: redirection port and protocol

Access rules and HTTP headers

See *Writing rules and headers* to learn how to configure access control and HTTP headers sent to application by LL::NG.

Attention: With **Nginx**-based ReverseProxy, header directives can be appended by a LUA script.

To send more than **15** headers to protected applications, you have to edit and modify :

`/etc/nginx/nginx-lua-headers.conf`

Danger: * **Nginx** gets rid of any empty headers. There is no point of passing along empty values to another server; it would only serve to bloat the request. In other words, headers with **empty values are completely removed** from the passed request.

* **Nginx**, by default, will consider any header that **contains underscores as invalid**. It will remove these from the proxied request. If you wish to have Nginx interpret these as valid, you can set the `underscores_in_headers` directive to "on", otherwise your headers will never make it to the backend server.

POST data

See *Form replay* to learn how to configure form replay to POST data on protected applications.

Options

Some options are available:

- **Port**: used to build redirection URL *(when user is not logged, or for CDA requests)*
- **HTTPS**: used to build redirection URL
- **Maintenance mode**: reject all requests with a maintenance message
- **Aliases**: list of aliases for this virtual host *(avoid to rewrite rules,...)*
- **Access to trace**: can be used for overwriting REMOTE_CUSTOM with a custom function. Provide a comma separated parameters list with custom function path and args. Args can be vars or session attributes, macros, ... By example: My::accessToTrace, Doctor, Who, _whatToTrace
- **Type**: handler type (normal, *ServiceToken Handler*, *DevOps Handler*,...)
- **Required authentication level**: this option avoids to reject user with a rule based on `$_authenticationLevel`. When user has not got the required level, he is redirected to an upgrade page in the portal. This default level is required for ALL locations relative to this virtual host. It can be overrided for each locations.
- **ServiceToken timeout**: by default, ServiceToken is just valid during 30 seconds. This TTL can be customized for each virtual host.

Attention: A hash reference containing $req, $session, $vhost, $custom and an array reference with provided parameters is passed to the custom function.

```
package My;

sub accessToTrace {
    my $hash    = shift;
    my $custom  = $hash->{custom};
    my $req     = $hash->{req};
    my $vhost   = $hash->{vhost};
    my $custom  = $hash->{custom};
    my $params  = $hash->{params};
    my $session = $hash->{session};

    return "$custom alias $params->[0]_$params->[1]:$session->{groups}:$session->{
↪$params->[2]}";
}

1;
```

Danger: A same virtual host can serve many locations. Each location can be protected by a different type of handler :

```
server test1.example.com 80
  location ^/AuthBasic  => AuthBasic handler
  location ^/AuthCookie => Main handler
```

Keep in mind that AuthBasic handler use "Login/Password" to authenticate users. If you set "Authentication level required" option to "5" by example, AuthBasic requests will be ALWAYS rejected because AuthBasic authentication level is lower than required level.

Attention: A negative or null ServiceToken timeout value will be overloaded by `handlerServiceTokenTTL` (30 seconds by default).

"Port" and "HTTPS" options are used to build redirection URL *(when user is not logged, or for CDA requests)*. By default, default values are used. These options are only here to override default values.

4.6 Sessions

LL::NG rely on a session mechanism with the session ID as a shared secret between the user (in *SSO cookie*) and the *session database*.

To configure sessions, go in Manager, `General Parameters` » `Sessions`:

- **Store user password in session data**: see *password store documentation*.
- **Display session identifier**: Should the session ID be displayed in the manager's session explorer. The session ID is a sensitive information that should only be shown to highly trusted administrators.
- **Sessions timeout**: Maximum lifetime of a session. Old sessions are deleted by a cron script.
- **Sessions activity timeout**: Maximum inactivity duration.
- **Sessions update interval**: Minimum interval used to update session when activity timeout is set.

Danger: Session activity timeout requires Handlers to have a write access to sessions database.

- **Opening conditions**: rules which are evaluated before granting session, see *Grant Session plugin documentation*
- **Sessions Storage**: you can define here which session backend to use, with the backend options. See *sessions database configuration* to know which modules you can use. Here are some global options that you can use with all sessions backends:
  - **generateModule**: allows one to override the default module that generates sessions identifiers. For security reasons, we recommend to use Lemonldap::NG::Common::Apache::Session::Generate::SHA256
  - **IDLength**: length of sessions identifiers. Max is 32 for MD5 and 64 for SHA256
- **Multiple sessions**, you can restrict the number of open sessions:
  - **One session per user**: when a user logs in, all their previous sessions are removed
  - **One IP address per user**: when a user logs in, all their previous sessions on a different IP address are removed
  - **One user per IP address**: when a user logs in, all sessions that belong to a different user on that IP address are removed
  - **Display deleted sessions**: display deleted sessions on authentication phase.
  - **Display other sessions**: display other sessions on authentication phase, with a link to delete them.

- **Persistent sessions**: are used for storing users log in history, 2F devices, OIDCConsents and so on. Heavy organizations may have to disable persistent sessions storage to avoid too many database tuples.
  - **Disable storage**: Do not store user persitent sessions.

Attention: Note that since HTTP protocol is not connected, restrictions are not applied to the new session: the oldest are destroyed.

4.7 Command-line tools

New in version 2.0.9.

You can use the `lemonldap-ng-sessions` tool to search, update or delete sessions. See a few examples in *the examples page*

Deprecated since version 2.0.10.

- LLNG Portal provides a simple tool to delete a session: `llngDeleteSession`. To use it, simply give it the user identifier *(wildcard are authorizated)*:

```
# Delete all sessions opened by user "dwho"
$ llngDeleteSession dwho
# Delete all sessions opened by user starting with "dh"
$ llngDeleteSession dh*
# Delete all sessions:
$ llngDeleteSession *
```

CHAPTER

FIVE

PORTAL CONFIGURATION

5.1 The portal

The portal is the main component of LL::NG. It provides many features:

- **Authentication service** of course
  - Web based for normal users:
    - using own database (*LDAP*, *SQL*, ...)
    - using web server authentication system (used for *SSL*, *Kerberos*, *HTTP basic authentication*, ...)
    - using external identity provider (*SAML*, *OpenID*, *CAS*, *Twitter*, other LL::NG system, ...)
    - all together (based on user *choice*, *rules*, ...)
  - *SOAP based* and *REST based* for client-server software, specific development, ...
- **Identity provider**: LL::NG is able to provide identity service using:
  - *SAML*
  - *OpenID Connect*
  - *CAS*
- *Identity provider proxy*: LL::NG can be used as proxy translator between systems talking SAML, OpenID, CAS, ...
- **Internal SOAP server** used by *SOAP configuration backend* and usable for specific development (see *SOAP services* for more)
- **Internal REST server** used by *REST configuration backend* and usable for specific development (see *REST services* for more)
- Interactive **management of user passwords**:
  - Password change form (in menu)
  - Self service reset (send a mail to the user with a to change the password)
  - Force password change with LDAP password policy password reset flag
- *Application menu*: display authorized applications in categories
- *Notifications*: prompt users with a message if found in the notification database
- Second factors management

5.1.1 Functioning

LL::NG portal is a modular component. It needs 4 modules to work:

- *Authentication*: how check user credentials
- *User database*: where collect user information
- *Password database*: where change password
- *Identity provider*: how forward user identity

Tip: Each module can be disabled using the `Null` backend.

5.1.2 Kinematics

1. Check if URL asked is valid
2. Check if user is already authenticated
    - If not authenticated (or authentication is forced) try to find it (userDB module) and to authenticate it (auth module), create session, ask for second factor if required, calculate groups and macros and store them. In 1.3, LL::NG has got a captcha feature which is used in this case.
3. Modify password if asked (password module)
4. Provides identity if asked (IdP module)
5. Build *cookie(s)*
6. Redirect user to the asked URL or display menu

Note: See also *general kinematics presentation*.

5.1.3 URL parameters

Some parameters in URL can change the behavior of the portal:

- **logout**: Launch the logout process (for example: `logout=1`)
- **tab**: Preselect a tab (Choice or Menu) (for example: `tab=password`)
- **llnglanguage**: Force lang used to display the page (for example: `llnglanguage=fr`)
- **setCookieLang**: Update lang cookie to persist the language set with `llnglanguage` parameter (for example: `setCookieLang=1`)

5.2 Portal customization

Note: The portal is the visible part of LemonLDAP::NG, all user interactions are displayed on it.

5.2.1 Main Logo

You can change the default Main Logo in Manager: General Parameters > Portal > Customization > Main Logo.

A blank value disables Main Logo display.

Tip:

- Logo files must be stored in lemonldap-ng-portal/site/htdocs/static/my/path directory
- Logo file path must be like my/path/logo.png
- Main logo is included in Portal templates AND mail body

5.2.2 Show languages choice

You can disabled languages choice in Manager: General Parameters > Portal > Customization > Show languages choice.

Option enabled by default.

Tip: If languages choice is disabled, Portal displays accepted languages by your browser (EN by default).

5.2.3 Custom CSS file

You can define a custom CSS file, for example `custom.css`, which will be loaded after default CSS files. This file needs to be created in the static repository (`/usr/share/lemonldap-ng/portal/htdocs/static/boostrap/css`).

Then set this value in Custom CSS parameter : `bootstrap/css/custom.css`.

Sample CSS file, to remove white background of main logo:

```
#header img {
  background-color: transparent;
}
```

Skin

LemonLDAP::NG is shipped with bootstrap skin.

But you can make your own. See Skin customization below.

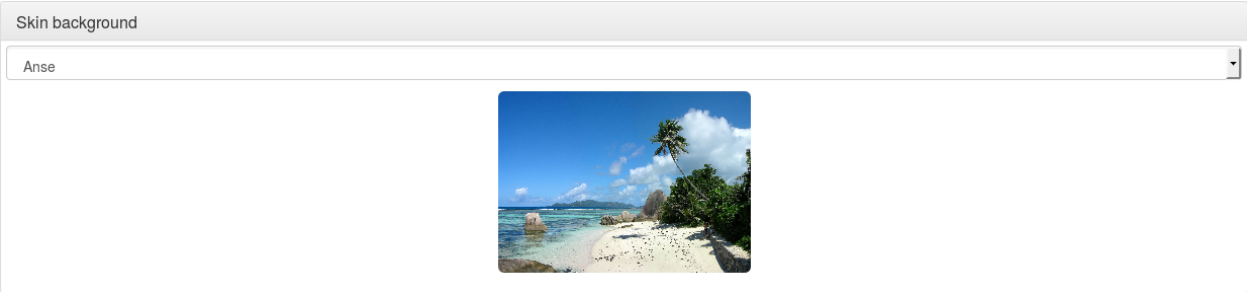
5.2.4 Default skin

You can change the default skin in Manager: `General Parameters` > `Portal` > `Customization` > `Default skin`.

Select the `Custom` skin, then set the name of the skin you want to use in the input below.

5.2.5 Skin background

Go in `General Parameters` > `Portal` > `Customization` > `Skin background`. You can define a background by selecting one of the available image. Use `None` to use the default skin background configuration.



To set your own background, copy your file in `/usr/share/lemonldap-ng/portal/htdocs/static/common/backgrounds/` and register it in `/etc/lemonldap-ng/lemonldap-ng.ini`:

```
[portal]
portalSkinBackground = file.png
```

You can also use `lemonldap-ng-cli`:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli set portalSkinBackground file.png
```

5.2.6 Skin rules

You might want to display different skin depending on the URL that was called before being redirected to the portal, or the IP address of the user.

To achieve this, you can create a rule in the Manager: select `General Parameters` > `Portal` > `Customization` > `Skin display rules` on click on "New key". Then fill the two fields;

- **Rule**: a Perl expression (you can use %ENV hash to get environment variables, or $_url to get URL called before redirection, or $ipAddr to use user IP address). If the rule evaluation is true, the corresponding skin is applied.
- **Skin**: the name of the skin to use.

5.2.7 Skin files

A skin is composed of different files:

- **.tpl**: Perl HTML::Template files, for HTML content
- **.css**: CSS (styles)
- **.js**: Javascript
- images and other media files

A skin will often refer to the `common` skin, which is not a real skin, but shared skin objects (like scripts, images and CSS).

5.2.8 Skin customization

Attention: If you modify directly the skin files, your modifications will certainly be erased on the next upgrade. The best is to create your own skin, based on an existing skin.

Here we explain how to create a new skin, named `myskin`, from the `bootstrap` skin.

First copy static content:

```
cd /usr/share/lemonldap-ng/portal/htdocs/static
mkdir myskin
cd myskin/
cp -a ../bootstrap/js/ .
cp -a ../bootstrap/css/ .
mkdir images
```

Then create symbolic links on template files, as you might not want to rewrite all HTML code (else, do as you want).

```
cd /usr/share/lemonldap-ng/portal/templates/
mkdir myskin
cd myskin/
```

We include some template files that can be customized:

- customhead.tpl : HTML header markups (like CSS, js inclusion)
- customheader.tpl : HTML code in the header
- customfooter.tpl : HTML code in the footer
- customLoginHeader.tpl : HTML code in the login header
- customLoginFooter.tpl : HTML code in the login footer

To use custom files, copy them into your skin folder:

```
cp ../bootstrap/custom* .
```

Then you can add your media to `myskin/images`, you will be able to use them in HTML template with this code:

```
<img src="<TMPL_VAR NAME="STATIC_PREFIX">myskin/images/logo.png" class="mx-auto d-block"␣
↪/>
```

To change CSS, two options:

- Edit myksin/css/styles.css and myskin/css/styles.min.css
- Create a new CSS file, for example myskin/css/myskin.css and load it in customhead.tpl:

```
<link href="<TMPL_VAR NAME="STATIC_PREFIX">myskin/css/myskin.css" rel="stylesheet" type=
↪"text/css" />
```

Put then all custom HTML code in the custom template files.

To configure your new skin in Manager, select the custom skin, and enter your skin name in the configuration field. For example with lemonldap-ng-cli:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 set portalSkin 'myskin'␣
↪portalSkinBackground ''
```

5.2.9 Messages

Messages are defined in source code. If they really do not please you, override them! You just need to know the ID of the message (look at Portal/Simple.pm).

There are two methods to do this:

- Use `lemonldap-ng.ini`:

```
[portal]

# Custom error messages
error_0 = Big brother is watching you, authenticated user

# Custom standard messages
msg_lastLogins = Your last connections
```

You can also define messages in several languages or disable message boxes by using the bareword `_hide_` :

```
error_en_0        = Big brother is watching you, authenticated user
error_fr_0        = Souriez vous êtes surveillés !
msg_fr_lastLogins = Dernières connexions
error_9           = _hide_
```

- Create a lang file in custom skin:

If you have a custom skin, then you can create a lang file in `templates/<your skin>` similar to the default lang files provided in `htdocs/static/languages/`.

For example `templates/myskin/en.json`:

```
{
"PE9":"Please authenticate!"
}
```

You can also create a file called `all.json` to override messages in all languages.

5.2.10 Menu tabs

If you modify the menu template to add some tabs, you should add the new tabs in `customMenuTabs` parameter in lemonldap-ng.ini:

```
[portal]

customMenuTabs = test, test2
```

This will allow one to display the tab directly with this URL: http://auth.example.com/?tab=test

5.2.11 Template parameters

Template parameters are defined in source code. If you need to add a template parameter for your customization, then add to `lemonldap-ng.ini`:

```
[portal]

# Custom template parameters
tpl_myparam = world
```

Then you will be able to use it in your template like this:

```
Hello <TMPL_VAR NAME="myparam">!
```

All session variables are also available in templates, with the prefix `session_`:

```
Hello <TMPL_VAR NAME="session_cn">!
```

You can also display environment variables, with the prefix `env_`:

```
Your IP is <TMPL_VAR NAME="env_REMOTE_ADDR">
```

Buttons

This node allows one to enable/disable buttons on the login page:

- **Check last logins**: display a checkbox on login form, allowing user to check his login history right after opening session
- **Reset password**: display a link to *reset your password page* (for password based authentication backends). Number of allowed retries can be set (3 times by default)
- **Register**: display a link to *register page* (for password based authentication backends)
- **Reset certificate**: display a link to *reset certificate page* (for password based authentication backends)

Password management

5.2.12 General

- **Require old password**: used only in the password changing module of the menu, will check the old password before updating it
- **Hide old password**: used only if the password need to be reset by the user (LDAP password policy), will hide the old password input
- **Send mail on password change**: send a mail if the password is changed from the Menu, or from forced password reset (LDAP password policy)

5.2.13 Password Policy

Tip: Available since version 2.0.6

- **Minimal size**: leave 0 to bypass the check
- **Minimal lower characters**: leave 0 to bypass the check
- **Minimal upper characters**: leave 0 to bypass the check
- **Minimal digit characters**: leave 0 to bypass the check
- **Minimal special characters**: leave 0 to bypass the check
- **Allowed special characters**: set '__ALL__' value to allow ALL special characters. A blanck value forbids ALL special characters (Note that _ is not a special character)
- **Display policy in password form**: enable this to display an information message about password policy constraints

Other parameters

- **User attribute**: which session attribute will be used to display `Connected as` in the menu
- **New window**: open menu links in new window
- **Anti iframe protection**: Set `X-Frame-Options` and CSP `frame-ancestors` headers (see Browser compatibility)
- **Ping interval**: Number of milliseconds between each ping (Ajax request) on the portal menu. Set to 0 to dismiss checks.
- **Show error on expired session**: Display the error "Session expired", which stops the authentication process. This is enabled by default but can be disabled to prevent transparent authentication (like SSL or Kerberos) to be stopped.
- **Show error on mail not found**: Display error if provided mail is not found in password reset by mail process. Disabled by default to prevent mail enumeration from this page.

5.3 Portal menu

Note: The menu is displayed if authentication is successful.

5.3.1 Menu modules

LemonLDAP::NG portal menu has 4 modules:

- **Application list**: display categories and applications allowed for the user
- **Password change**: form to change the password
- **Login history**: display user's last logins and last failed logins
- **OIDC Consents**: display user's OpenId Connect consents
- **Logout**: logout button

Each module can be activated through a rule, using user session information. These rules can be set through Manager: `General Parameters` > `Portal` > `Menu` > `Modules activation`.

You can use `0` or `1` to disable/enable the module, or use a more complex rule. For example, to display the password change form only for user authenticated through LDAP or DBI:

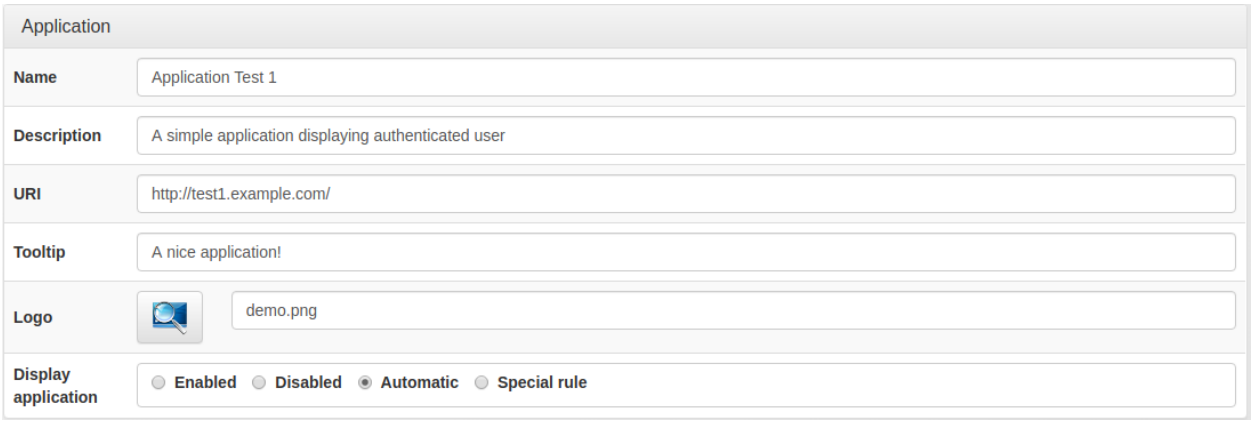
```
$_auth eq LDAP or $_auth eq DBI
```

5.3.2 Categories and applications

Configuring the virtual hosts is not sufficient to display an application in the menu. Indeed, a virtual host can serve several applications (http://vhost.example.com/appli1, http://vhost.example.com/appli2).

In Manager, you can configure categories and applications in `General Parameters` > `Portal` > `Menu` > `Categories and applications`.

Application parameters:



- **Name**: display text
- **Description**
- **URI**: URL of the application

- **Tooltip**: information display on mouse over the button
- **Logo**: file name to use as logo
- **Display application**:
  - **Enabled**: always display
  - **Disabled**: never display
  - **Automatic**: display only if the user can access it
  - **Special rule**: specify a *rule* or "sp: <name>" where "name" is the key name of the service provider, the corresponding rule will be applied *(available for CAS, SAML or OpenID-Connect)*

Tip: Categories and applications are displayed in alphabetical order.

Tip: The chosen logo file must be in portal applications logos directory (`portal/htdocs/static/common/apps/`). You can set a custom logo by setting the logo file name directly in the field, and copy the logo file in portal applications logos directory

5.4 REST/SOAP servers

5.4.1 Presentation

LL::NG portal can be configured as REST or *(deprecated)* SOAP server, for several usage:

- Configuration access
- Sessions access
- Authentication
- Specific application needs

5.4.2 Configuration

- **SOAP/REST exported attributes**: list session attributes shared through SOAP/REST
  - Use + to append the default list of technical attributes, example: `+ uid mail`

REST

Go in `General Parameters` > `Plugins` > `Portal servers` > `REST services`:

- **Session server**: Enable REST for sessions
- **Configuration server**: Enable REST for configuration
- **Authentication server**: Enable REST for authentication
- **Password reset server**: Enable REST for password reset
- **Server clock tolerance**: Allow a clock drift
- **Export secret attributes**: Secret attributes can be exported

See also *REST Services*.

SOAP *(deprecated)*

Go in `General Parameters` > `Plugins` > `Portal servers` > `SOAP services`:

- **Session server**: Enable SOAP for sessions
- **Configuration server**: Enable SOAP for configuration
- **WSDL server**: Enable WSDL server

See also *SOAP Services*.

5.5 Captcha

5.5.1 Presentation

Captcha is a security mechanism aimed to prevent robots to submit forms.

Captchas are available on the following forms:

- Login form: where user enters login and password to authenticate
- Password reset by mail form: where user enters mail to recover a lost password
- Register form: where user enters information to create a new account

Attention: We use the Perl module GD::SecurityImage to generate images, you need to install it if you enable Captcha feature.

5.5.2 Configuration

Go in `General parameters` > `Portal` > `Captcha`:

- **Activation in login form**: set to 1 to display captcha in login form
- **Activation in password reset by mail form**: set to 1 to display captcha in password reset by mail form
- **Activation in register form**: set to 1 to display captcha in register form
- **Size**: length of captcha

5.6 Public pages

Note: Public pages are available since version 1.9.8.

5.6.1 Presentation

Public pages are an easy way to build pages based on LL::NG portal skin. You can for example create a landing page or customize error pages with it.

A public page is just a template created in portal/skins/*yourskin*/public/ directory, for example test.tpl. This page can then be displayed with this URL: http://auth.example.com/public?page=test

5.6.2 Page creation

Create the public/ directory :

```
mkdir /var/lib/lemonldap-ng/portal/skins/bootstrap/public
```

Create the new page:

```
vi /var/lib/lemonldap-ng/portal/skins/bootstrap/public/test.tpl
```

```
<TMPL_INCLUDE NAME="../header.tpl">

<div class="container">
  <div class="alert alert-success">
    TEST
  </div>
</div>

<TMPL_INCLUDE NAME="../footer.tpl">
```

Display the page: http://auth.example.com/public?page=test

5.7 Second Factors

Two-Factor Authentication *(as known as 2FA)* is a kind (subset) of multi-factor authentication. It is a method to confirm a user's claimed identity by using a combination of two different factors between:

1. something they know *(login / password, . . .)*
2. something they have *(U2F Key, smartphone, . . .)*
3. something they are *(biometrics like fingerprints, . . .)*

Since 2.0, LLNG provides some second factor plugins that can be used to complete authentication module with 2FA :

- *U2F-or-TOTP (enable both U2F and TOTP)*
- *TOTP (to use with* FreeOTP,Google-Authenticator,*. . .)*
- *U2F tokens*
- *Yubikey tokens (provided by Yubico)*
- *E-Mail 2F (Send a code to an email address)*
- *External 2F (to call an external command)*
- *REST (Remote REST app)*
- *RADIUS (Remote RADIUS server)*

The E-Mail, External and REST 2F modules *may be declared multiple times* with different sets of parameters.

5.7.1 Registration on first use

If you want to force a 2F registration on first login, you can use the *Force 2FA registration at login* option.

You can use a *rule<writingrulesand_headers>* to enable this behavior only for some users.

5.7.2 Second factor expiration

You can display a message if an expired second factor has been removed by enabling *Display a message if an expired SF is removed* option or setting a rule.

5.7.3 Self-care on Portal

User may register second factors themselves on the Portal by using the 2FA Manager.

The link will be displayed if at least one SFA module is enabled. You can set a rule to display or not the link.

5.7.4 Session upgrade through 2FA

If you enable the *Use 2FA for session upgrade* option, second factor will only be asked on login if the target application requires an authentication level that is strictly higher than the one obtained by the Authentication backend (first factor).

The session upgrade mechanism will only require the second factor step, instead of doing a complete reauthentication.

5.7.5 Providing tokens from an external source

If you don't want to use self-registration features for U2F, TOTP and so on, you can set tokens by yourself *(in your LDAP server for example)* and map it to `_2fDevices` attribute. `_2fDevices` is a JSON array that contains token descriptions :

```
[ {"type" : "TOTP", "name" : "MyTOTP", ...}, {<other_token>}, ...]
```

U2F Tokens

```
{"name" : "MyU2FKey" , "type" : "U2F" , "_userKey" : "########" , "_keyHandle":"########
↪" , "epoch":"1524078936"}
```

TOTP Tokens

```
{"name" : "MyTOTP" , "type" : "TOTP" , "_secret" : "########" , "epoch" : "1523817955"}
```

Yubikey Tokens

```
{"name" : "MyYubikey" , "type" : "UBK" , "_yubikey" : "########" , "epoch" : "1523817715
↪"}
```

5.7.6 Developer corner

To develop a new 2FA plugin, read `Lemonldap::NG::Portal::Main::SecondFactor (3pm)` manpage. Your 2F module must be a Perl class named `Lemonldap::NG::Portal::2F::`//<custom_name>//. To enable it, set `available2F` key in your `lemonldap-ng.ini` file :

```
[portal]
available2F = U2F,TOTP,<custom_name>
```

To enable manager Second Factor Administration Module, set `enabledModules` key in your `lemonldap-ng.ini` file :

```
[portal]
enabledModules = conf, sessions, notifications, 2ndFA
```

5.8 Standard SSO protocols

5.8.1 SAML service configuration

Note: SAML service configuration is a common step to configure LL::NG as *SAML SP* or *SAML IDP*.

Presentation

This documentation explains how configure SAML service in LL::NG, in particular:

- Install prerequisites
- Import or generate security keys
- Set SAML end points

Attention: Service configuration will be used to generate LL::NG SAML metadata, that will be shared with other providers. It means that if you modify some settings here, you will have to share again the metadata with other providers. In other words, take the time to configure this part before sharing metadata.

Prerequisites

Lasso

SAML2 implementation is based on Lasso. You will need a very recent version of Lasso (>= 2.6.0).

Debian/Ubuntu

You can use official Debian packages or those available here: http://deb.entrouvert.org/.

Tip: We recommend Lasso 2.6 for the SHA256 support, so use the stretch-testing repository of deb.entrouvert.org.

You will only need to install liblasso-perl package:

```
sudo apt-get install liblasso-perl
```

RHEL/CentOS/Fedora

RPMs are available in LL::NG RPM "extras" repository (see *YUM repository*)

Then install lasso and lasso-perl packages:

```
yum install lasso lasso-perl
```

Attention: Only 64bits package are available.

Other

Download the Lasso tarball and compile it on your system.

Service configuration

Go in Manager and click on `SAML 2 Service` node.

Tip: You can use #PORTAL# in values to replace the portal URL.

Entry Identifier

Your EntityID, often use as metadata URL, by default #PORTAL#/saml/metadata.

Note: The value will be use in metadata main markup:

```
<EntityDescriptor entityID="http://auth.example.com/saml/metadata">
  ...
</EntityDescriptor>
```

Security parameters

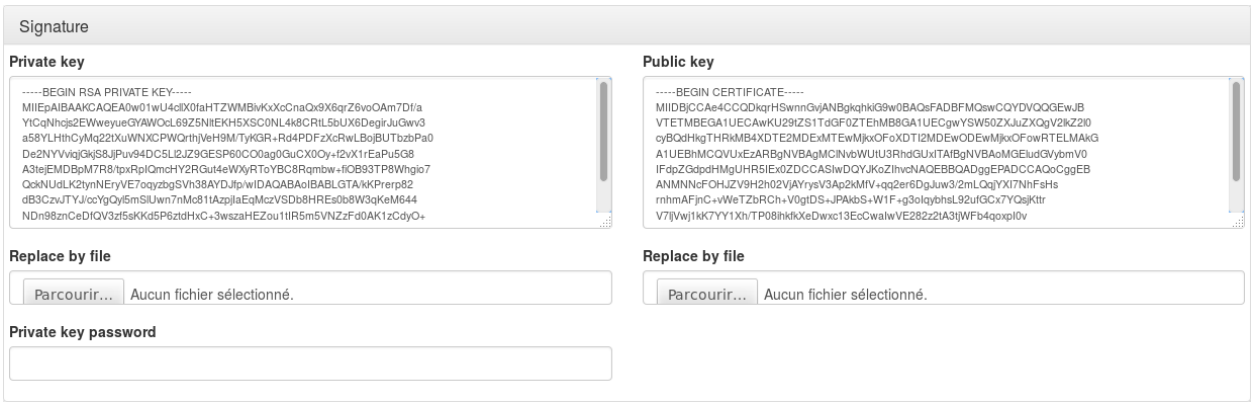
You can define keys for SAML message signature and encryption. If no encryption keys are defined, signature keys are used for signature and encryption.

To define keys, you can:

- import your own private and public keys (`Replace by file` input)
- generate new public and private keys (`New certificate` button)

Changed in version 2.0.10: A X.509 certificate is now generated instead of a plain public key. It has 20 years of validity, and is self signed with the 2048bit RSA key.

Tip: You can enter a password to protect private key with a password. It will be prompted if you generate keys, else you can set it in the `Private key password`.



- **Use certificate in response**: Certificate will be sent inside SAML responses.
- **Signature method**: set the signature algorithm

Changed in version 2.0.10: The signature method can now be overriden for a SP or IDP. This will only work if you are using a certificate for signature instead of a public key.

Attention: If you are running a version under 2.0.10, the choice of a signature algorithm will affect all SP and IDP.

Converting a RSA public key to a certificate

If your application complains about the lack of certificate in SAML Metadatas, and you generated a public RSA key instead of a certificate in a previous version of LemonLDAP::NG, you can convert the public key into a certificate without changing the private key.

Save the private key in a file, and use the `openssl` commands to issue a self-signed certificate:

```
$ openssl req -new -key private.key -out cert.pem -x509 -days 3650
```

NameID formats

SAML can use different NameID formats. The NameID is the main user identifier, carried in SAML messages. You can configure here which field of LL::NG session will be associated to a NameID format.

Note: This parameter is used by *SAML IDP* to fill the NameID in authentication responses.

Customizable NameID formats are:

- Email
- X509
- Windows
- Kerberos

Tip: For example, if you are using *AD as authentication backend*, you can use sAMAccountName for the Windows NameID format.

Other NameID formats are automatically managed:

- **Transient**: NameID is generated
- **Persistent**: NameID is restored from previous sessions
- **Undefined**: Default NameID format is used

Authentication contexts

Each LL::NG authentication module has an authentication level, which can be associated to an SAML authentication context.

Note: This parameter is used by *SAML IDP* to fill the authentication context in authentication responses. It will use the authentication level registered in user session to match the SAML authentication context. It is also used by *SAML SP* to fill the authentication level in user session, based on authentication response authentication context.

Customizable NameID formats are:

- Password
- Password protected transport
- TLS client
- Kerberos

Organization

Note: This concerns all parameters for the Organization metadata section:

```
<Organization>
  <OrganizationName xml:lang="en">Example</OrganizationName>
  <OrganizationDisplayName xml:lang="en">Example</OrganizationDisplayName>
  <OrganizationURL xml:lang="en">http://www.example.com</OrganizationURL>
</Organization>
```

- **Display Name**: should be displayed on IDP, this is often your society name
- **Name**: internal name
- **URL**: URL of your society

Service Provider

Note: This concerns all parameters for the Service Provider metadata section:

```
<SPSSODescriptor>
  ...
</SPSSODescriptor>
```

General options

- **Signed Authentication Request**: set to On to always sign authentication request.
- **Want Assertions Signed**: set to On to require that received assertions are signed.

Tip: These options can then be overridden for each Identity Provider.

Single Logout

For each binding you can set:

- **Location**: Access Point for SLO request.
- **Response Location**: Access Point for SLO response.

Available bindings are:

- HTTP Redirect
- HTTP POST
- HTTP SOAP

Assertion Consumer

For each binding you can set:

- **Default**: will this binding be used by default for authentication response.
- **Location**: Access Point for SSO request and response.

Available bindings are:

- HTTP Artifact
- HTTP POST

Artifact Resolution

The only authorized binding is SOAP. This should be set as Default.

Identity Provider

Note: This concerns all parameters for the Service Provider metadata section:

```
<IDPSSODescriptor>
  ...
</IDPSSODescriptor>
```

General parameters

- **Want Authentication Request Signed**: By default, LemonLDAP::NG requires all SAML Requests to be signed. Set it to "Off" to let each Service Provider metadata decide if their requests should be verified by LemonLDAP::NG or not.

Tip: The per-SP "Check SSO message signature" setting allows you to disable signature verification even if this option is set to "On" globally

---

This option will set the *WantAuthnRequestsSigned* attribute to *true* in LemonLDAP::NG's IDP Metadata.

Warning: This setting requires Lasso 2.6.1 to be effective. Older versions behave as if this setting was set to "Off"

Single Sign On

For each binding you can set:

- **Location**: Access Point for SSO request.
- **Response Location**: Access Point for SSO response.

Available bindings are:

- HTTP Redirect
- HTTP POST
- HTTP Artifact

Single Logout

For each binding you can set:

- **Location**: Access Point for SLO request.
- **Response Location**: Access Point for SLO response.

Available bindings are:

- HTTP Redirect
- HTTP POST
- HTTP SOAP

Artifact Resolution

The only authorized binding is SOAP. This should be set as Default.

Attribute Authority

Note: This concerns all parameters for the Attribute Authority metadata section

```
<AttributeAuthorityDescriptor>
  ...
</AttributeAuthorityDescriptor>
```

Attribute Service

This is the only service to configure, and it accept only the SOAP binding.

Response Location should be empty, as SOAP responses are directly returned (synchronous binding).

Advanced

These parameters are not mandatory to run SAML service, but can help to customize it:

- **IDP resolution cookie name**: by default, it's the LL::NG cookie name suffixed by `idp`, for example: `lemonldapidp`.
- **UTF8 metadata conversion**: set to On to force partner's metadata conversion.
- **RelayState session timeout**: timeout for RelayState sessions. By default, the RelayState session is deleted when it is read. This timeout allows one to purge sessions of lost RelayState.
- **Use specific query_string method**: the CGI query_string method may break invalid URL encoded signatures (issued for example by ADFS). This option allows one to use a specific method to extract query string, that should be compliant with non standard URL encoded parameters.
- **Override Entity ID when acting as IDP**: By default, SAML entityID is the same for SP and IDP roles. Some federations (like *Renater*) can require a different entityID for IDP. In this case, you can fill here the IDP entityID, for example: `https://auth.example.com/saml/metadata/idp`.

SAML sessions module name and options

By default, the main session module is used to store SAML temporary data (like relay-states), but SAML sessions need to use a session module compatible with the *sessions restrictions feature*.

Tip: You can also choose a different session module to split SSO sessions and SAML sessions.

Common Domain Cookie

The common domain is used by *SAML SP* to find an Identity Provider for the user, and by *SAML IDP* to register itself in user's IDP list.

Configuration parameters are:

- **Activation**: Set to On to enable Common Domain Cookie support.
- **Common domain**: Name of the common domain (where common cookie is available).
- **Reader URL**: URL used by SAML SP to read the cookie. Leave blank to deactivate the feature.
- **Writer URL**: URL used by SAML IDP to write the cookie. Leave blank to deactivate the feature.

Discovery Protocol

Note: Discovery Protocol is also know as WAYF Service. More information can be found in the specification: sstc-saml-idp-discovery-cs-01.pdf.

When Discovery Protocol is enabled, the LL::NG IDP list is no more used. Instead user is redirected on the discovery service and is redirected back to LL::NG with the chosen IDP.

Attention: If the chosen IDP is not registered in LL::NG, user will be redirected to discovery service again.

Configuration parameters are:

- **Activation**: Set to On to enable Discovery Protocol support.
- **EndPoint URL**: Discovery service page
- **Policy**: Set a value here if you don't want to use the default policy (`urn:oasis:names:tc:SAML:profiles:SSO:idp-discovery-protocol:single`)
- **Is passive**: Enable this option to avoid user interaction on discovery service page

5.8.2 OpenID Connect service configuration

Service configuration

Go in Manager and click on `OpenID Connect Service` node.

Issuer identifier

Set the issuer identifier, which should be the portal URL.

For example: http://auth.example.com

End points

Name of different OpenID Connect endpoints. You can keep the default values unless you have a specific need to change them.

- **Authorization**
- **Token**
- **User Info**
- **JWKS**
- **Registration**
- **End of session**
- **Check Session**

Tip: The end points are published inside JSON metadata.

Authentication context

You can associate here an authentication context to an authentication level.

Security

- **Keys** : define public/private key pair to do asymmetric signature. A JWKS `kid` (Key ID) is automatically derived when generating new keys.
- **Dynamic Registration**: Set to 1 to allow clients to register themselves. This may be a security risk as this will create a new configuration in the backend per registration request. You can limit this by protecting in the WebServer the registration end point with an authentication module, and give the credentials to clients.
- **Only allow declared scopes**: By default, LemonLDAP::NG will grant all requested scopes. When this option is in use, LemonLDAP will only grant:
  - Standard OIDC scopes (`openid` `profile` `email` `address` `phone`)
  - Scopes declared in *Extra Claims*
  - Scopes declared in *Scope Rules* (if they match the rule)

- **Authorization Code flow**: Set to 1 to allow Authorization Code flow
- **Implicit flow**: Set to 1 to allow Implicit flow
- **Hybrid flow**: Set to 1 to allow Hybrid flow

Sessions

It is recommended to use a separate sessions storage for OpenID Connect sessions, else they will stored in the main sessions storage.

Dynamic Registration

If dynamic registration is enabled, you can configure the following options to define attributes and extra claims when a new relying party is registered through the `/oauth2/register` endpoint:

- Exported vars for dynamic registration
- Extra claims for dynamic registration

Key rotation script

OpenID Connect specification let the possibility to rotate keys to improve security. LL::NG provide a script to do this, that should be put in a cronjob.

The script is `/usr/share/lemonldap-ng/bin/rotateOidcKeys`. It can be run for example each week:

```
5 5 * * 6 www-data /usr/share/lemonldap-ng/bin/rotateOidcKeys
```

Tip: Set the correct Apache user, else generated configuration will not be readable by LL::NG.

Session management

LL::NG implements the OpenID Connect Change Notification specification

A `changed` state will be sent if the user is disconnected from LL::NG portal (or has destroyed its SSO cookie). Else the `unchanged` state will be returned.

Tip: To work, the LL::NG cookie must not be protected against javascript (`httpOnly` option should be set to `0`).

5.9 Authentication, users and password databases

5.9.1 Active Directory

Authentication	Users	Password
✓	✓	✓

Presentation

The Active Directory module is based on *LDAP module*, with the following features:

- Specific default values for filters to match AD schema
- Compatible password modification
- Reset password on next logon workflow

Configuration

The configuration is the same as the *LDAP module*.

AD password policy

AD password policy does not follow the LDAP RFC, but Microsoft has implemented its own policy. LemonLDAP::NG implements partially the policy:

- when pwdLastSet = 0 in the user entry, it means that password has been reset, and a form is displayed to the user to change his password.
- when computed virtual attribute 'msDS-User-Account-Control-Computed' as 6th flag set to 8, the password is considered expired (support from Windows Server 2003). It is too late for the user to do anything. He must contact his administrator.
- a warning before password expiration is possible in AD, but only in GPO (Computer ConfigurationWindows SettingsLocal PoliciesSecurity Options under Interactive Logon: Prompt user to change password before expiration). However it as no reality in LDAP referential. A "password warning time before password expiration" variable can be specified in LemonLDAP::NG to do so.

Attention: Note: since AD 2012, each user can have a specific password expiration policy. Then, the "maximum password age" can have different values. This is currently unsupported in LemonLDAP::NG because every policy must be computed with their precedence to know which maximum password age to apply.

To configure warning before password expiration, you must set two variables in Active Directory parameters in Manager:

- **Password max age** : number of seconds after the last password change, before it expires. It must match AD policy
- **Password expire warning** : number of seconds between password expiration and the date from which user is warned his password will expire.

5.9.2 Apache

Authentication	Users	Password
✓		

Presentation

LL::NG can delegate authentication to Apache, so it is possible to use any Apache authentication module, for example Kerberos, Radius, OTP, etc.

Attention: To authenticate users by using Kerberos, you can now use the new *Kerberos authentication module* which allow one to chain Kerberos in a *combination*

Tip: Apache authentication module will set the `REMOTE_USER` environment variable, which will be used by LL::NG to get authenticated user.

Configuration

LL::NG

In General Parameters > Authentication modules, choose `Apache` as authentication backend.

You may want to failback to another authentication backend in case of the Apache authentication fails. Use then the *Multiple authentication module*, for example:

```
Apache;LDAP
```

Tip: In this case, the Apache authentication module should not require a valid user and not be authoritative, else Apache server will return an error and not let LL::NG Portal manage the failback authentication.

Apache

The Apache configuration depends on the module you choose, you need to look at the module documentation, for example:

- Kerberos
- NTLM
- Radius
- ...

Tips

Kerberos

The Kerberos configuration is quite complex. You can find some configuration tips *on this page*.

Tip: Prefer new *Kerberos* module.

Compatibility with Identity Provider modules

When using IDP modules (like CAS or SAML), the activation of Apache authentication can alter the operation. This is because the client often need to request directly the IDP, and the Apache authentication will block the request.

In this case, you can add in the Apache authentication module:

```
Satisfy any
Order allow,deny
allow from APPLICATIONS_IP
```

This will bypass the authentication module for request from APPLICATIONS_IP.

5.9.3 CAS

Authentication	Users	Password
✓		

Presentation

LL::NG can delegate authentication to a CAS server. This requires Perl CAS module.

Tip: LL::NG can also act as *CAS server*, that allows one to interconnect two LL::NG systems.

LL::NG can also request proxy tickets for its protected services. Proxy tickets will be collected at authentication phase and stored in user session under the form:

`_casPT<serviceID>` = **Proxy ticket value**

They can then be forwarded to applications through *HTTP headers*.

Tip: CAS authentication will automatically add a *logout forward rule* on CAS server logout URL in order to close CAS session on LL::NG logout.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose CAS for authentication.

Tip: You can then choose any other module for users and password.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

Then, go in `CAS parameters`:

- **Authentication level**: authentication level for this module.

Then create the list of CAS servers in the manager. For each, set:

- **Server URL** *(required)*: CAS server URL (must use https://)
- **Renew authentication** *(default: disabled)*: force authentication renewal on CAS server
- **Gateways authentication** *(default: disabled)*: force transparent authentication on CAS server
- **Display Name**: Name to display. Required if you have more than 1 CAS server declared
- **Icon**: Path to CAS Server icon. Used only if you have more than 1 CAS server declared
- **Order**: Number to sort CAS Servers display
- **Proxied services**: list of services for which a proxy ticket is requested:
  - **Key**: Service ID
  - **Value** Service URL (CAS service identifier)

Tip: If no proxied services defined, CAS authentication will not activate the CAS proxy mode with this CAS server.

5.9.4 Databases

Authentication	Users	Password
✓	✓	✓

Presentation

Drivers

LL::NG can use a lot of databases as authentication, users and password backend:

- MariaDB/MySQL
- PostGreSQL
- Oracle
- ...

Indeed, any Perl DBD driver can be used.

Schema

LL::NG can use two tables:

- Authentication table: where login and password are stored
- User table: where user data are stored (mail, name, etc.)

Tip: Authentication table and user table can be the same.

The password can be in plain text, or encoded with a standard SQL method:

- SHA
- SHA1
- MD5

Example 1: two tables

Authentication table

id	login	password
0	coudot	1f777a6581e478499f4284e54fe2d4a4e513dfff
1	xguimard	a15a18c8bb17e6f67886a9af1898c018b9f5a072
2	tchemineau	1f777a6581e478499f4284e54fe2d4a4e513dfff

User table

id	user	name	mail
0	coudot	Clément OUDOT	coudot@example.com
1	tchemineau	Thomas CHEMINEAU	tchemineau@example.com
2	xguimard	Xavier GUIMARD	xguimard@example.com

Example 2: single table

id	user	password	name	mail
0	coudot	1f777a6581e478499f4284e54fe2d4a4e513dfffClément OUDOT		coudot@example.com
1	tchem-ineau	1f777a6581e478499f4284e54fe2d4a4e513dfffThomas CHEM-INEAU		tchem-ineau@example.com
2	xguimard	a15a18c8bb17e6f67886a9af1898c018b9f5a072	Xavier GUIMARD	xguimard@example.com

SQL

LL::NG will operate some SQL queries:

- Authentication: select row in authentication table matching user and password
- Search user: select row in user table matching user
- Change password: update password column in authentication table matching user

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Database (DBI) for authentication, users and/or password modules.

Authentication level

The authentication level given to users authenticated with this module.

Attention: As DBI is a login/password based module, the authentication level can be:

- increased (+1) if portal is protected by SSL (HTTPS)
- decreased (-1) if the portal autocompletion is allowed (see *portal customization*)

Exported variables

List of columns to query to fill user session. See also *exported variables configuration*.

Connection

Tip: Connection settings can be configured differently for authentication process and user process. This allows one to use different databases for these process. By default, if user process connection settings are empty, authentication process connection settings will be used.

- **Chain**: DBI chain, including database driver name and database name (for example: dbi:mysql:database=lemonldapng;host=localhost).
- **User**: Connection user

- **Password**: Connection password

Schema

- **Authentication table**: authentication table name
- **User table**: user table name
- **Login field name**: name of authentication table column hosting login
- **Password field name**: name of authentication table column hosting password
- **Mail field name**: name of authentication table column hosting mail (for password reset)
- **Login field name in user table**: name of user table column hosting login

Password

- **Hash schema**: SQL method for hashing password. Can be left blank for plain text passwords.
- **Dynamic hash activation**: Activate dynamic hashing. With dynamic hashing, the hash scheme is recovered from the user password in the database during authentication.
- **Supported non-salted schemes**: List of whitespace separated hash schemes. Every hash scheme MUST match a non-salted hash function in the database. LemonLDAP::NG relies on this hashing function for computing user password hashes. These hashes MUST NOT be salted (no random data used in conjunction with the password).
- **Supported salted schemes**: List of whitespace separated salted hash schemes, of the form "sscheme", where scheme MUST match a non-salted hash function in the database. LemonLDAP::NG relies on this hashing function for computing user password hashes. Salted and non-salted scheme lists are not necessarily equivalent. (for example: non-salted="sha256" and salted="ssha ssha512" is valid)
- **Dynamic hash scheme for new passwords**: LemonLDAP::NG is able to store new passwords in the database (while modifying or reinitializing the password). You can choose a salted or non salted dynamic hashed password. The value must be an element of "Supported non-salted schemes" or "Supported salted schemes".

Attention: The SQL function MUST have hexadecimal values as input AND output

Tip: Here is an example for creating a postgreSQL SHA256 function. 1. Install postgresql-contrib. 2. Activate extension: `CREATE EXTENSION pgcrypto;` 3. Create the hash function:

```
CREATE OR REPLACE FUNCTION sha256(varchar) returns text AS $$
SELECT encode(digest(decode($1, 'hex'), 'sha256'), 'hex')
$$ LANGUAGE SQL STRICT IMMUTABLE;
```

5.9.5 Demonstration

Authentication	Users	Password
✓	✓	✓

Presentation

This mode allow one to test LemonLDAP::NG without any third-party software.

Danger: This mode must not be used for other purpose than test and demonstration!

Demonstration backend has hard coded user accounts:

Login	Password	Mail	Role
rtyler	rtyler	rtyler@badwolf.org	user
msmith	msmith	msmith@badwolf.org	user
dwho	dwho	dwho@badwolf.org	administrator

Note: As you may have guessed, these accounts are famous characters from the TV show Doctor Who.

The AuthDemo and UserDBDemo will allow you to log in and get the standard attributes (uid, cn and mail). The PasswordDBDemo will allow you to change the password with some basic checks, but as the data are hard coded, the password will never be really changed.

Configuration

Select Demonstration for authentication, user and password backend.

You can also modify list of exported variables. Only uid, cn and mail attributes are available. See also *exported variables configuration*.

5.9.6 Facebook

Authentication	Users	Password
✓	✓	

Presentation

Facebook is a famous social network service. Facebook uses OAuth2 protocol to allow applications to reuse its own authentication process (it means, if your are connected to Facebook, other applications can trust Facebook and let you in).

You need Net::Facebook::Oauth2 package.

You need to register a new application on Facebook to get an application ID and a secret. See https://developers.facebook.com/apps on how to do that.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Facebook for authentication module. You can also use Facebook as user database.

Then, go in `Facebook parameters`:

- **Authentication level**: authentication level for this module.
- **Facebook application ID**: the application ID you get
- **Facebook application secret**: the corresponding secret
- **User field**: Facebook field that will be used as default user identifier

If you use Facebook as user database, declare values in exported variables:

- use any key name you want. If you want to refuse access when a data is missing, just add a "!" before the key name
- in the value field, set the field name. You can show them using Facebook Graph API explorer and have a list of supported fields in the Graph API User reference. For example:
  - cn => name
  - mail => email
  - sn => last_name

Attention: Do not query user field in exported variables, as it is already registered by the authentication module in `$_user`.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

Tip: You can use the same Facebook access token in your applications. It is stored in session datas under the name `$_facebookToken`

5.9.7 GitHub

Authentication	Users	Password
✓		

Presentation

GitHub uses OAuth2 protocol to allow applications to reuse its own authentication process (see https://developer.github.com/apps/building-oauth-apps/authorizing-oauth-apps/).

You need to register a new application on LinkedIn to get an application ID and a secret: https://github.com/settings/apps/new.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose GitHub for authentication module.

Then, go in `GitHub parameters`:

- **Authentication level**: authentication level for this module.
- **Client ID**: the application ID you get
- **Client secret**: the corresponding secret
- **Field containing user identifier**: Field that will be used as main user identifier in LL::NG, usually `login`
- **Scope**: OAuth 2.0 scopes, see https://developer.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps/

Tip: Collected fields are stored in session in `github_` keys

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in:

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

5.9.8 Databases

Authentication	Users	Password
✓		

Presentation

LLNG can use GPG to authenticate users. It is not useful for day-to-day authentication but can be used for example if user has lost his password. The login form will ask user to sign a challenge and post result.

Configuration of LemonLDAP::NG

In Manager, go in `General Parameters` > `Authentication modules` and choose GPG for authentication, users and/or password modules. Then you just have to set GPG database. For example `/usr/share/keyrings/debian-keyring.gpg`

Tip: You can then choose any other module for users and password.

Then, go in `GPG parameters`:

- **Authentication level**: authentication level for this module
- **GPG database**: database to store users GPG public key

5.9.9 Kerberos

Authentication	Users	Password
✓		

Presentation

Kerberos is a network authentication protocol used to authenticate users based on their desktop session.

LL::NG uses GSSAPI module to validate Kerberos ticket against a local keytab.

LLNG Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Kerberos for authentication. Then go to "Kerberos parameters" and configure the following parameters:

- **keytab file** (required): the Kerberos keytab file
- **Use Ajax request**: set to "enabled" if you want to use an Ajax request instead of a direct Kerberos attempt. **This is required if you want to chain Kerberos in a** *combination*
- **Kerberos authentication level**: default to 3
- **Use Web Server Kerberos module**: set to "enabled" to use the Web Server module (for example Apache mod_auth_kerb) instead of Perl Kerberos code to validate Kerberos ticket
- **Remove domain in username**: set to "enabled" to strip username value and remove the '@domain'.
- **Allowed domains**: if set, tickets will only be accepted if they come from one of the domains listed here. This is a space-separated list. This feature can be useful when using *combination* and cross-realm Kerberos trusts.

Attention:

- Due to a perl GSSAPI issue, you may need to copy the keytab in /etc/krb5.keytab which is the default location hardcoded in the library
- As Kerberos ticket is passed inside Authorization header, you may need to set CGIPassAuth on in Apache (with old Apache, use `RewriteCond %{HTTP:Authorization}` followed by `RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]`)

Kerberos configuration

The Kerberos configuration is quite complex. You can find some configuration tips *on this page*.

Web Server Kerberos module

If you want to let Web Server Kerberos module validates the Kerberos ticket, set the according option to "enabled" and configure the portal virtual host to launch the module if "kerberos" GET parameter is in the request.

Example with Apache and mod_auth_kerb:

```
<If "%{QUERY_STRING} =~ /kerberos=/">
  <IfModule auth_kerb_module>
    AuthType Kerberos
    KrbMethodNegotiate On
    KrbMethodK5Passwd Off
    KrbAuthRealms EXAMPLE.COM
    Krb5KeyTab /etc/lemonldap-ng/auth.keytab
    KrbVerifyKDC On
    KrbServiceName Any
    require valid-user
  </IfModule>
</If>
```

5.9.10 LDAP

Authentication	Users	Password
✓	✓	✓

Presentation

LL::NG can use an LDAP directory to:

- authenticate user
- get user attributes
- get groups where user is registered
- change password (with server side password policy management)

This works with every LDAP v2 or v3 server, including *Active Directory*.

LL::NG is compatible with LDAP password policy:

- LDAP server can check password strength, and LL::NG portal will display correct errors (password too short, password in history, etc.)
- LDAP sever can block brute-force attacks, and LL::NG will display that account is locked
- LDAP server can force password change on first connection, and LL::NG portal will display a password change form before opening SSO session

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose LDAP for authentication, users and/or password modules.

Tip: For *Active Directory*, choose `Active Directory` instead of `LDAP`.

Authentication level

The authentication level given to users authenticated with this module.

Attention: As LDAP is a login/password based module, the authentication level can be:

- increased (+1) if portal is protected by SSL (HTTPS)
- decreased (-1) if the portal autocompletion is allowed (see *portal customization*)

Exported variables

List of attributes to query to fill user session. See also *exported variables configuration*.

Connection

- **Server host**: LDAP server hostname or URI (by default: localhost). Accept some specificities:
  - More than one server can be set here separated by spaces or commas. They will be tested in the specified order.
  - To use TLS, set `ldap+tls://server` and to use LDAPS, set `ldaps://server` instead of server name.
  - If you use TLS, you can set any of the Net::LDAP start_tls() sub like `ldap+tls://server/verify=none&capath=/etc/ssl`. You can also use cafile and capath parameters.
- **Server port**: TCP port used by LDAP server if different from the standard ports. Can also be specified in the server host URI.
- **Verify LDAP server certificate**: It is highly recommended to verify the identity of the remote server. This setting is only enforced for LDAPS or TLS connections.
- **Users search base**: Base of search in the LDAP directory.
- **Account**: DN used to connect to LDAP server. By default, anonymous bind is used.
- **Password**: password to used to connect to LDAP server. By default, anonymous bind is used.
- **Connection timeout**: applies only when initiating the connection
- **Operation timeout**: applies to all LDAP operations
- **Version**: LDAP protocol version.
- **Binary attributes**: regular expression matching binary attributes (see Net::LDAP documentation).
- **CA file path**: This allows you to override the default system-wide certificate authorities by giving a single file containing the CA used by the LDAP server.

- **CA directory path**: This allows you to override the default system-wide certificate authorities by giving the path of a directory containing your trusted certificates.

Attention: LemonLDAP::NG need anonymous access to LDAP Directory RootDSE in order to check LDAP connection.

Filters

Tip: In LDAP filters, $user is replaced by user login, and $mail by user email.

- **Default filter**: default LDAP filter for searches, should not be modified.
- **Authentication filter**: Filter to find user from its login (default: `(&(uid=$user)(objectClass=inetOrgPerson))`)
- **Mail filter**: Filter to find user from its mail (default: `(&(mail=$mail)(objectClass=inetOrgPerson))`)
- **Alias dereference**: How to manage LDAP aliases. (default: `find`)

Tip: For Active Directory, the default authentication filter is:

```
(&(sAMAccountName=$user)(objectClass=person))
```

And the mail filter is:

```
(&(mail=$mail)(objectClass=person))
```

Groups

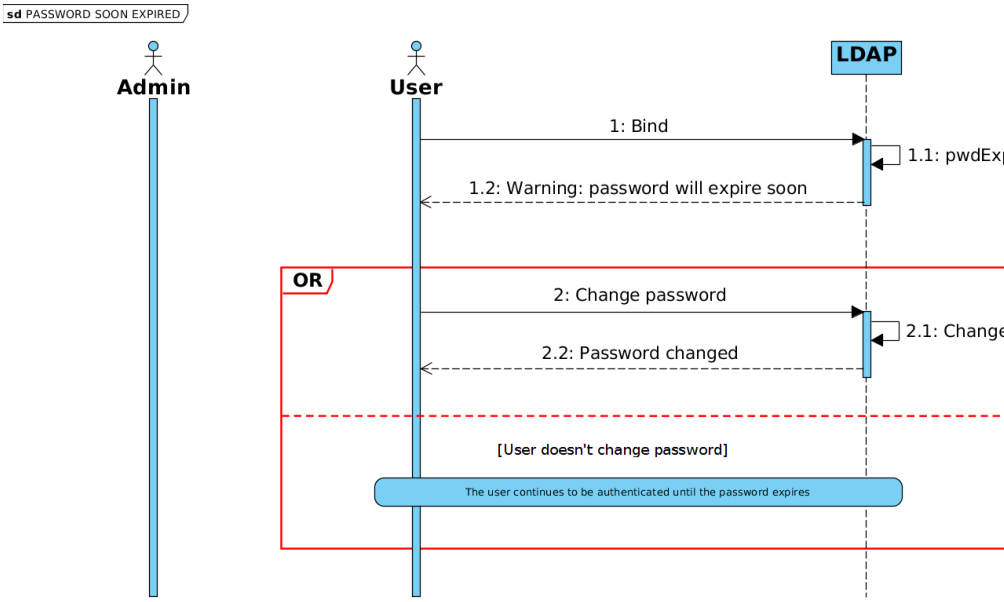
- **Search base**: DN of groups branch. If no value, disable group searching.
- **Object class**: objectClass of the groups (default: groupOfNames).
- **Target attribute**: name of the attribute in the groups storing the link to the user (default: member).
- **User source attribute**: name of the attribute in users entries used in the link (default: dn).
- **Searched attributes**: name(s) of the attribute storing the name of the group, spaces separated (default: cn).
- **Decode searched value**: with Active Directory, member DN value is sometimes bad decoded and groups are not found, activate this option to force value decoding.
- **Recursive**: activate recursive group functionality (default: 0). If enabled, if the user group is a member of another group (group of groups), all parents groups will be stored as user's groups.
- **Group source attribute**: name of the attribute in groups entries used in the link, for recursive group search (default: dn).

Note: The groups that the user belongs to are available as `$groups` and `%hGroups`, as documented *here*

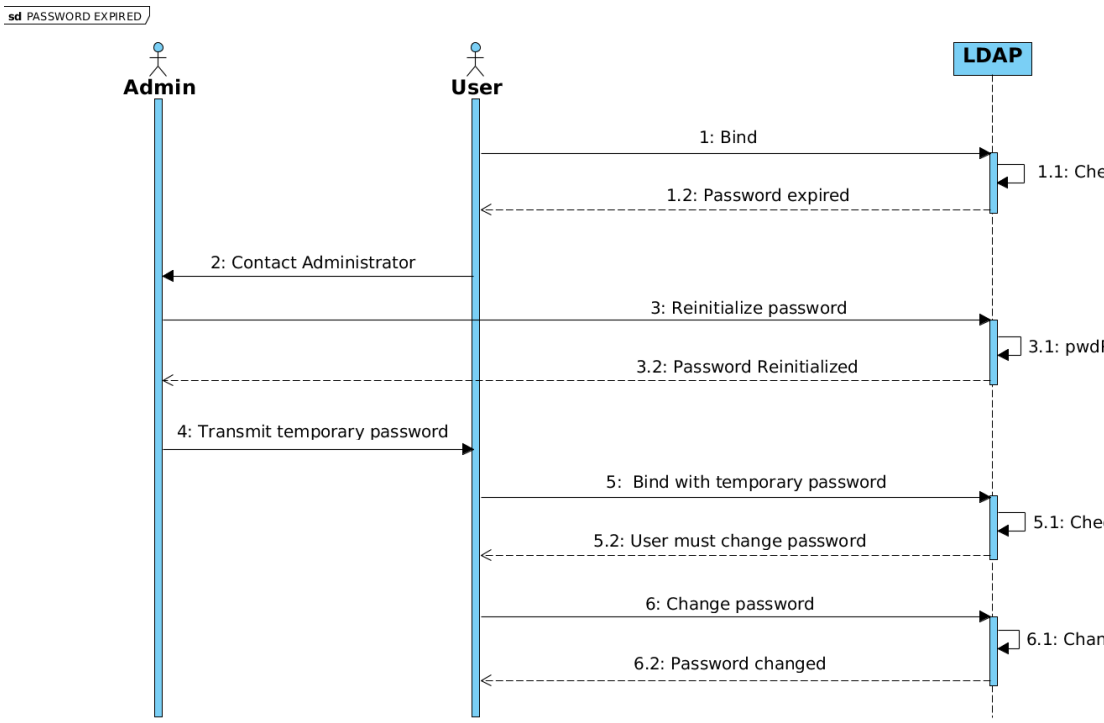
Attention: If your LDAP countains over a thousand groups, you should avoid using group processing, check out *the performance page* for alternatives

Password

- **Password policy control**: enable to use LDAP password policy. This requires at least Net::LDAP 0.38. (see ppolicy workflow below)
- **Password modify extended operation**: enable to use the LDAP extended operation `password modify` instead of standard modify operation.
- **Change as user**: enable to perform password modification with credentials of connected user. This requires to request user old password (see *portal customization*).
- **LDAP password encoding**: can allow one to manage old LDAP servers using specific encoding for passwords (default: utf-8).
- **Use reset attribute**: enable to use the password reset attribute. This attribute is set by LemonLDAP::NG when *password was reset by mail* and the user choose to generate the password (default: enabled).
- **Reset attribute**: name of password reset attribute (default: pwdReset).
- **Reset value**: value to set in reset attribute to activate password reset (default: TRUE).
- **Allow a user to reset his expired password**: if activated, the user will be prompted to change password if his password is expired (default: disabled)
- **Search for user before password change: this option forces the password** change module to search for the user again, refreshing its DN. This feature is only useful in rare cases when you use LDAP as the password module, but not as the UserDB module. (default: enabled)
- **IBM Tivoli DS support**: enable this option if you use ITDS. LL::NG will then scan error message to return a more precise error to the user.



Password expiration warning workflow



Password expiration workflow

5.9.11 LinkedIn

Authentication	Users	Password
✓		

Presentation

LinkedIn is a professional social network. It uses OAuth2 protocol to allow applications to reuse its own authentication process (see https://developer.linkedin.com/docs/oauth2).

You need to register a new application on LinkedIn to get an application ID and a secret. See https://www.linkedin.com/developer/apps/ on how to do that.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose LinkedIn for authentication module.

Then, go in `LinkedIn parameters`:

- **Authentication level**: authentication level for this module.
- **Client ID**: the application ID you get
- **Client secret**: the corresponding secret
- **Searched fields** (deprecated): Fields requested on People endpoint in v1, no more used in v2 API
- **Field containing user identifier**: Field that will be used as main user identifier in LL::NG, usually `id` (LinkedIn numeric identifer) or `emailAddress`.

- **Scope**: OAuth 2.0 scopes, use `r_liteprofile` to get first name and last name, and `r_emailaddress` to get email.

Tip: Collected fields are stored in session in `linkedIn_` keys

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

5.9.12 Null

Authentication	Users	Password
✓	✓	✓

Presentation

LL::NG Null backend is a transparent backend:

- Authentication: will create session without prompting any credentials (but will register client IP and creation date)
- Users: will not collect any data (but you can still register environment variables in session)
- Password: will not change any password

You can use Null backend to bypass some authentication process steps.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Null for authentication, users or password module.

Then, go in `Null parameters`:

- **Authentication level**: authentication level for this module.

5.9.13 OpenID Connect

Authentication	Users	Password
✓	✓	

Presentation

Note: OpenID Connect is a protocol based on REST, OAuth 2.0 and JOSE stacks. It is described here: http://openid.net/connect/.

LL::NG can act as an OpenID Connect Relying Party (RP) towards multiple OpenID Connect Providers (OP). It will get the user identity through an ID Token, and grab user attributes through UserInfo endpoint.

As an RP, LL::NG supports a lot of OpenID Connect features:

- Authorization Code flow
- Automatic download of JWKS
- JWT signature verification
- Access Token Hash verification
- ID Token validation
- Get UserInfo as JSON or as JWT
- Logout on EndSession end point

You can use this authentication module to link your LL::NG server to any OpenID Connect Provider. Here are some examples, witch their specific documentation:

Google

Presentation

Do you we have to present Google? The good news is that Google is a standard OpenID Provider, and so you can easily delegate the authentication of LL::NG to Google: https://developers.google.com/identity/protocols/OpenIDConnect

Attention: Google does not support logout through OpenID Connect. If you close your session on LL::NG side, your Google session will still be open.

Register on Google

You need a Google developer account to access to https://console.developers.google.com/

Here you can go in API Manager and get new credentials (`client_id` and `client_secret`).

You need to provide the callback URLs, for example https://auth.domain.com/?openidcallback=1.

Declare Google in your LL::NG server

Go in Manager and create a new OpenID Connect provider. You can call it `google` for example.

Click on `Metadata`, and use the OpenID Connect configuration URL to load them: https://accounts.google.com/.well-known/openid-configuration.

You can also load the JWKS data from the URL https://www.googleapis.com/oauth2/v3/certs. But as Google rotate their keys, we will also configure a refresh interval on JKWS data.

Go in `Exported attributes` to choose which attributes you want to collect. Google supports these claims:

- email
- email_verified
- family_name
- given_name
- locale
- name
- picture
- sub

Now go in `Options`:

- In `Configuration`, register the `client_id` and `client_secret` given by Google. Set also the configuration URI with https://accounts.google.com/.well-known/openid-configuration, and JWKS refresh, for example every day: 86400.
- In `Protocol`, adapt the `scope` to the exported attributes you want. You can for example use `openid profile email`.
- In `Display`, you can set the name and the logo

France Connect

Presentation

France Connect is an authentication platform made by French government.

Attention: It is for the moment only in BETA stage. This documentation will explain how to configure LL::NG with the developer reserved space.

Register on France Connect

Once *OpenID Connect service* is configured, you need to register to France Connect.

Use the following form: https://doc.integ01.dev-franceconnect.fr/inscription.

You need to provide the callback URLs, for example https://auth.domain.com/?openidcallback=1.

You will then get a `client_id` and a `client_secret`.

Declare France Connect in your LL::NG server

Go in Manager and create a new OpenID Connect provider. You can call it `france-connect` for example.

Click on `Metadata` and set manually the metadata of the service, using France Connect endpoints. For example:

```
{
"issuer": "https://fcp.integ01.dev-franceconnect.fr",
"authorization_endpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/authorize",
"token_endpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/token",
"userinfo_endpoint": "https://fcp.integ01.dev-franceconnect.fr/api/v1/userinfo",
"end_session_endpoint":"https://fcp.integ01.dev-franceconnect.fr/api/v1/logout"
}
```

You can skip JWKS data, they are not provided by France Connect. The security relies on the symmetric key `client_secret`.

Go in `Exported attributes` to choose which attributes from "identité pivot" you want to collect. See https://doc.integ01.dev-franceconnect.fr/identite-pivot

Now go in `Options`:

- In `Configuration`, register the `client_id` and `client_secret` given by France Connect
- In `Protocol`, adapt the `scope` to the exported attributes you want. See https://doc.integ01.dev-franceconnect.fr/fs-scopes
- In `Display`, you can set the name and the logo

Google	France Connect
	

Attention: OpenID-Connect specification is not finished for logout propagation. So logout initiated by relaying-party will be forward to OpenID-Connect provider but logout initiated by the provider (or another RP) will not be propagated. LLNG will implement this when spec will be published.

Configuration

OpenID Connect Service

See *OpenIDConnect service* configuration chapter.

Authentication and UserDB

In `General Parameters` > `Authentication modules`, set:

- **Authentication module**: OpenID Connect
- **Users module**: OpenID Connect

Tip: As passwords will not be managed by LL::NG, you can disable *menu password module*.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

```
General Parameters > Advanced Parameters > Security > Content Security Policy > Form
destination
```

Then in `General Parameters` > `Authentication modules` > `OpenID Connect parameters`, you can set:

- **Authentication level**: level of authentication to associate to this module
- **Callback GET parameter**: name of GET parameter used to intercept callback (default: openidconnectcallback)
- **State session timeout**: duration of a state session (used to keep state information between authentication request and authentication response) in seconds (default: 600)

Register LL::NG to an OpenID Connect Provider

To register LL::NG, you will need to give some information like application name or logo.

You will be asked to provide a *Redirect URI* for LemonLDAP::NG, which is constructed by appending the `openidcallback=1` parameter to the Portal URL.

For example:

- https://auth.example.com/?openidcallback=1

Attention: If you use the *choice backend*, you need to set SameSite cookie value to "Lax" or "None". See *SSO cookie parameters*

After registration, the OP must give you a client ID and a client secret, that will be used to configure the OP in LL::NG.

Declare the OpenID Connect Provider in LL::NG

In the Manager, select node `OpenID Connect Providers` and click on `Add OpenID Connect Provider`. Give a technical name (no spaces, no special characters), like "sample-op";

You can then access to the configuration of this OP.

Metadata

The OP should publish its metadata in a JSON file (see for example Google metadata). Copy the content of this file in the textarea. Portal discovery document can be found here: https://#portal#/.well-known/openid-configuration

If no metadata is available, you need to write them in the textarea. Mandatory fields are:

- issuer
- authorization_endpoint
- token_endpoint
- userinfo_endpoint

You can also define:

- jwks_uri
- endsession_endpoint

Example template:

```
{
  "issuer": "https://auth.example.com/",
  "authorization_endpoint": "https://auth.example.com/oauth2/authorize",
  "token_endpoint": "https://auth.example.com/oauth2/token",
  "userinfo_endpoint": "https://auth.example.com/oauth2/userinfo",
  "end_session_endpoint":"https://auth.example.com/oauth2/logout"
}
```

JWKS data

JWKS is a JSON file containing public keys. LL::NG can grab them automatically if jwks_uri is defined in metadata. Else you can paste the content of the JSON file in the textarea.

Tip: If the OpenID Connect provider only uses symmetric encryption, JWKS data is not useful.

Exported attributes

Define here the mapping between the LL::NG session content and the fields provided in UserInfo response. The fields are defined in OpenID Connect standard, and depends on the scope requested by LL::NG (see options in next chapter).

OpenID Connect claims

Claim name	Associated scope	Type	Example of corresponding LDAP attribute
sub	openid	string	uid
name	profile	string	cn
given_name	profile	string	givenName
family_name	profile	string	sn
middle_name	profile	string	
nickname	profile	string	
preferred_username	profile	string	displayName
profile	profile	string	labeledURI
picture	profile	string	
website	profile	string	
email	email	string	mail
email_verified	email	boolean	
gender	profile	string	
birthdate	profile	string	
zoneinfo	profile	string	
locale	profile	string	preferredLanguage
phone_number	phone	string	telephoneNumber
phone_number_verified	phone	boolean	
updated_at	profile	string	
formatted	address	string	registeredAddress
street_address	address	string	street
locality	address	string	l
region	address	string	st
postal_code	address	string	postalCode
country	address	string	co

So you can define for example:

- cn => name
- sn => family_name
- mail => email
- uid => sub

Options

- **Configuration**:
  - **Configuration endpoint**: URL of OP configuration endpoint
  - **JWKS data timeout**: After this time, LL::NG will do a request to get a fresh version of JWKS data. Set to 0 to disable it.
  - **Client ID**: Client ID given by OP
  - **Client secret**: Client secret given by OP
  - **Store ID token**: Allows one to store the ID token (JWT) inside user session. Do not enable it unless you need to replay this token on an application, or if you need the id_token_hint parameter when using logout.
- **Protocol**:

  - **Scope**: Value of scope parameter (example: openid profile). The `openid` scope is mandatory.
  - **Display**: Value of display parameter (example: page)
  - **Prompt**: Value of prompt parameter (example: consent)
  - **Max age**: Value of max_age parameter (example: 3600)
  - **UI locales**: Value of ui_locales parameter (example: en-GB en fr-FR fr)
  - **ACR values**: Value acr_values parameters (example: loa-1)
  - **Token endpoint authentication method**: Choice between `client_secret_post` and `client_secret_basic`
  - **Check JWT signature**: Set to 0 to disable JWT signature checking
  - **ID Token max age**: If defined, LL::NG will check the date of ID token and refuse it if it is too old
  - **Use Nonce**: If enabled, a nonce will be sent, and verified from the ID Token
- **Display**:
  - **Display name**: Name of the application
  - **Logo**: Logo of the application
  - **Order**: Number to sort buttons

5.9.14 PAM

Authentication	Users	Password
✓		

Presentation

LL::NG can use Pluggable authentication module as a simple authentication backend.

Configuration

Install Authen::PAM

You have to install the corresponding Perl module.

For CentOS/RHEL:

```
yum install perl-Authen-PAM
```

In Debian/Ubuntu, install the library through apt-get command

```
apt-get install libauthen-pam-perl
```

Configuration of LemonLDAP::NG

In Manager, go in `General Parameters` > `Authentication modules` and choose PAM for authentication.

Tip: You can then choose any other module for users and password.

Then, go in `PAM parameters`:

- **Authentication level**: authentication level for PAM module
- **PAM service**: the PAM service to use *(default: login)*

5.9.15 Proxy

Authentication	Users	Password
✓	✓	

Presentation

LL::NG is able to send (through REST or SOAP) authentication credentials to another LL::NG portal, like a proxy.

The difference with *remote authentication* is that the client will never be redirect to the main LL::NG portal. This configuration is usable if you want to expose your internal SSO portal to another network (DMZ).

Configuration

External portal

In Manager, go in `General Parameters` > `Authentication modules` and choose Proxy for authentication and users.

Then, go in `Proxy parameters`:

- **Internal portal URL**: URL of internal portal
- **Session service URL** (optional): Session service URL (default: same as previous for SOAP, same with "/session/my" for REST)
- **Cookie name** (optional): name of the cookie of internal portal, if different from external portal
- **Authentication level**: authentication level for Proxy module
- **Use SOAP instead of REST**: use a deprecated SOAP server instead of a REST one (you must set it if internal portal version is < 2.0). In this case, "Portal URL" parameter must contain SOAP endpoint (generally http://auth.example.com/index.pl/sessions for 1.9 and earlier, http://auth.example.com/sessions for 2.0)

Internal portal

The portal must be configured to accept REST or SOAP authentication requests if you chose to use SOAP. See: *REST server plugin* or *SOAP session backend (deprecated)*.

SOAP compatibility with 1.9 server

If your Proxy is a 2.0.x and your server is a 1.9.x, you should add this in your lemonldap-ng.ini:

```
soapProxyUrn = urn:Lemonldap/NG/Common/CGI/SOAPService
```

Attention: This feature needs at least LLNG version 2.0.8

5.9.16 Radius

Authentication	Users	Password
✓		

Presentation

LL::NG uses Perl Authen::Radius as a simple authentication backend.

Currently, the module is simply handling a Radius Authentication request and has been tested only against a FreeRadius server.

Configuration

Install Authen::Radius

You have to install the corresponding Perl module.

For CentOS/RHEL:

```
yum install perl-Authen-Radius
```

In Debian/Ubuntu, install the library through apt-get command

```
apt-get install libauthen-radius-perl
```

Configuration of LemonLDAP::NG

In Manager, go in `General Parameters` > `Authentication modules` and choose Radius for authentication.

Tip: You can then choose any other module for users and password.

Then, go in `Radius parameters`:

- **Authentication level**: authentication level for Radius module
- **Shared secret**: this is the passphrase to use to connect to the Radius server
- **Server hostname**: this is the hostname or IP address of the Radius server

5.9.17 REST

Authentication	Users	Password
✓	✓	✓

Presentation

This backend can be used to delegate authentication to some webservices.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose REST for authentication, users and/or password modules.

Then, go in `REST parameters` and you just have to set REST URL to provide wanted services:

Module	Parameter
Authentication level	Authentication level for this module
Authentication	Authentication URL
User database	User data URL
Password confirmation	Password confirmation URL
Password change	Password change URL

Tip: You can then choose any other module for users and password.

REST Dialog

LemonLDAP::NG will call the endpoints you declared at various steps during the login process.

The request performed by LemonLDAP::NG is a POST on the URL you specified, the content of the POST is a JSON document (`Content-Type: application/json`).

REST web services must respond with a success HTTP code (200), and the response must be a JSON document containing a `result` key. Auth/UserDB endpoints can add an `info` array that will be stored in session data (without reading "Exported variables").

URL	Query	Response
Authentication URL	`{"user":$user, "password":$password}`	`{"result":true/false,"info":{...}}`
User data URL	`{"user":$user}`	`{"result":true/false, "info":{"uid":"dwho",...}}`
Password confirmation URL	`{"user":$user, "password":$password}`	`{"result":true/false}`
Password change URL	`{"user":$user, "password":$password}`	`{"result":true/false}`

Tip: To have only one REST call during the login process, you can set REST only as an Authentication backend, configure Null as your User Database, and make sure the REST authentication URL send all your user attributes in the `info` response key

5.9.18 SAML

Authentication	Users	Password
✓	✓	

Presentation

LL::NG can use SAML2 to get user identity and grab some attributes defined in user profile on its Identity Provider (IDP). In this case, LL::NG acts like an SAML2 Service Provider (SP).

Several IDPs are allowed, in this case the user will choose the IDP he wants. You can preselect IDP with an IDP resolution rule.

For each IDP, you can configure attributes that are collected. Some can be mandatory, so if they are not returned by IDP, the session will not open.

Tip: LL::NG can also act as *SAML IDP*, that allows one to interconnect two LL::NG systems.

Configuration

SAML Service

See *SAML service* configuration chapter.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

Authentication and UserDB

In `General Parameters` > `Authentication modules`, set:

- Authentication module: SAML v2
- Users module: Same (eq SAML)

Tip: As passwords will not be managed by LL::NG, you can disable *menu password module*.

Register LemonLDAP::NG on partner Identity Provider

After configuring SAML Service, you can export metadata to your partner Identity Provider.

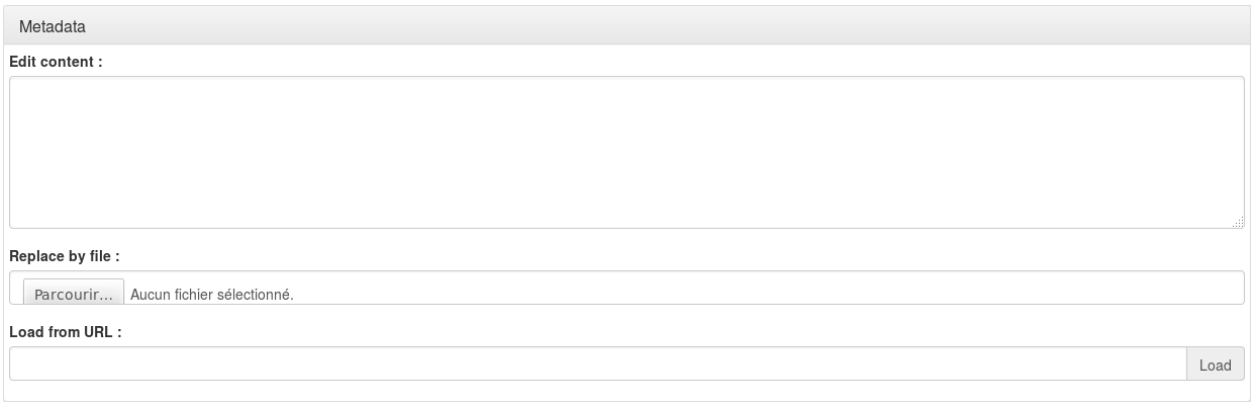
They are available at the EntityID URL, by default: http://auth.example.com/saml/metadata. You can also use http://auth.example.com/saml/metadata/sp to have only SP related metadata.

Register partner Identity Provider on LemonLDAP::NG

In the Manager, select node `SAML identity providers` and click on `Add SAML IDP`. The IDP name is asked, enter it and click OK.

Metadata

You must register IDP metadata here. You can do it either by uploading the file, or get it from IDP metadata URL (this require a network link between your server and the IDP):



Tip: You can also edit the metadata directly in the textarea

Exported attributes

For each attribute, you can set:

- **Variable name**: name of the variable in LemonLDAP::NG session that will contain this attribute. For example "uid" will then be used as $uid in access rules
- **Attribute name**: name of the SAML attribute coming from the remote IDP
- **Friendly Name**: optional, SAML attribute friendly name.
- **Mandatory**: if set to On, then session will not open if this attribute is not given by IDP.
- **Format** (optional): SAML attribute format.

Options

General options

- **Resolution Rule**: rule that will be applied to preselect an IDP for a user. You have access to all environment variable *(like user IP address)* and all session keys.

For example, to preselect this IDP for users coming from 129.168.0.0/16 network and member of "admin" group:

```
$ENV{REMOTE_ADDR} =~ /^192\.168/ and $groups =~ /\badmin\b/
```

Authentication request

- **NameID format**: force NameID format here (email, persistent, transient, etc.). If no value, will use first NameID Format activated in metadata.
- **Force authentication**: set ForceAuthn flag in authentication request
- **Passive authentication**: set IsPassive flag in authentication request
- **Allow proxied authentication**: allow an authentication response to be issued from another IDP that the one we register (proxy IDP). If you disallow this, you should also disallow direct login form IDP, because proxy restriction is set in authentication requests.
- **Allow login from IDP**: allow a user to connect directly from an IDP link. In this case, authentication is not a response to an issued authentication request, and we have less control on conditions.
- **Requested authentication context**: this context is declared in authentication request. When receiving the request, the real authentication context will be mapped to an internal authentication level (see *how configure the mapping*), that you can check to allow or deny session creation.
- **Allow URL as RelayState**: Set to On if the RelayState value sent by IDP is the URL where the user must be redirected after authentication.

Session

- **Adapt session lifetime**: session lifetime will be adapted from `SessionNotOnOrAfter` value found in authentication response. It means that if the IDP propose to close session earlier than the default LemonLDAP::NG timeout, the session _utime will be modified so that session is erased at the date indicated by the IDP.
- **Force UTF-8**: this will force UTF-8 conversion of attributes values collected from IDP.
- **Store SAML Token**: allows one to keep SAML token (assertion) inside user session. Don't enable it unless you need to replay this token on an application.
- **Attribute containing user identifier**: set the value of SAML attribute ("Name") that should be used as user main identifier ($user). If empty, the NameID content is used.

Signature

These options override service signature options (see *SAML service configuration*).

- **Signature method**: signature method for requests sent to this provider
- **Sign SSO message**: sign SSO message
- **Check SSO message signature**: check SSO message signature
- **Sign SLO message**: sign SLO message
- **Check SLO message signature**: check SLO message signature

Binding

- **SSO binding**: force binding to use for SSO (http-redirect, http-post, etc.)
- **SLO binding**: force binding to use for SLO (http-redirect, http-post, etc.)

Note: If no binding defined, the default binding in IDP metadata will be used.

Security

- **Encryption mode**: set the encryption mode for this IDP (None, NameID or Assertion).
- **Check time conditions**: set to Off to disable time conditions checking on authentication responses.
- **Check audience conditions**: set to Off to disable audience conditions checking on authentication responses.

Display

Used only if you have more than 1 SAML Identity Provider declared

- **Display name**: Name of the IDP
- **Logo**: Logo of the IDP
- **Order**: Number to sort IDP display

Tip: The chosen logo must be in Portal icons directory (`portal/static/common/`). You can set a custom icon by setting the icon file name directly in the field and copy the logo file in portal icons directory

5.9.19 Slave

Authentication	Users	Password
✓	✓	

Presentation

LL::NG Slave backend relies on HTTP headers to retrieve user login and/or attributes.

- Authentication: will check user login in a header and create session without prompting any credentials (but will register client IP and creation date)
- Users: collect data transferred in HTTP headers by the "master".

It allows one to put LL::NG::portal behind another web SSO, or behind a SSL hardware to delegate SSL authentication to that hardware.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Slave for authentication or users module.

Then, go in `Slave parameters`:

- **Authentication level**: authentication level for this module.
- **Header for user login**: header that contains the user main login
- **Master's IP address**: the IP addresses of servers which are accredited to authenticate user. This is a security point, to prevent someone to create a session by sending custom headers. You can set one or several IP addresses, spaces separated, or let this parameter empty to disable the checking.
- **Control header name**: header that contains a value to control. Let this parameter empty to disable the checking.
- **Control header content**: value to control. Let this parameter empty to disable the checking.
- **Display authentication logo**: display Slave logo.

You have then to declare HTTP headers exported by the main SSO (in **Exported Variables**). Example :

Key (LL::NG name)	Value (HTTP header name)
uid	Auth-User
mail	User-Email

Example

- Request with curl (AuthChoice with Slave and Secured cookie => double cookies for a single session):

Control header name: control

Control header content: password

```
curl -k https://127.0.0.1:19876 -H 'CN: dwho' -H 'Host: auth.example.com' -H 'Accept:␣
↪application/json' -H 'control: password' -d "lmAuth=2_Slave" | json_pp
```

- Response for good authentication:

```
{
    "result" : 1,
    "error" : 0,
    "id_http" : "5237ce20290d6110915a05d62f52618955b5f71b6dd3424481372ad419a5b122",
    "id" : "16fec9bd7a0523328568ca919ee0a6d6e329832f6c302bf36b106db92b5ec23d"
}
```

See also *exported variables configuration*.

5.9.20 SSL

Authentication	Users	Password
✓		

Presentation

LL::NG uses Apache SSL module, like any other *Apache authentication module*, with extra features:

- Choice of any certificate attribute as user main login
- Allow no certificate to chain with other authentication methods

Configuration (as the only authentication module)

By default, SSL is required before the portal is displayed (handled by webserver). If you want to display a button to connect to LLNG (compatible with *Combination*), you can activate "SSL by Ajax request" in the manager.

With Apache

Enable SSL in Apache

You have to install mod_ssl for Apache.

For CentOS/RHEL:

```
yum install mod_ssl
```

Tip: In Debian/Ubuntu mod_ssl is already shipped in `apache*-common` package.

Tip: For CentOS/RHEL, We advice to disable the default SSL virtual host configured in /etc/httpd/conf.d/ssl.conf.

Apache SSL global configuration

You can then use this default SSL configuration, for example in the head of /etc/lemonldap-ng/portal-apache2.conf:

```
SSLProtocol all -SSLv2
SSLCipherSuite HIGH:MEDIUM
SSLCertificateFile /etc/httpd/certs/ow2.cert
SSLCertificateKeyFile /etc/httpd/certs/ow2.key
SSLCACertificateFile /etc/httpd/certs/ow2-ca.cert
```

Note: Put your own files instead of `ow2.cert`, `ow2.key`, `ow2-ca.cert`:

- **SSLCertificateFile**: Server certificate
- **SSLCertificateKeyFile**: Server private key

- **SSLCACertificateFile**: CA certificate to validate client certificates

If you specify port in virtual host, then declare SSL port:

```
NameVirtualHost *:80
NameVirtualHost *:443
```

Apache portal SSL configuration

Edit the portal virtual host to enable SSL double authentication:

```
SSLEngine On
SSLVerifyClient optional
SSLVerifyDepth 10
SSLOptions +StdEnvVars
SSLUserName SSL_CLIENT_S_DN_CN
```

All SSL options are documented in Apache mod_ssl page.

Here are the main options used by LL::NG:

- **SSLVerifyClient**: set to `optional` to allow user with a bad certificate to access to LL::NG portal page. To switch to another authentication backend, use the *Multi* module, for example: `Multi SSL;LDAP`
- **SSLOptions**: set to `+StdEnvVars` to get certificate fields in environment variables
- **SSLUserName** (optional): certificate field that will be used to identify user in LL::NG portal virtual host

With Nginx

Enable SSL:

```
ssl on;
ssl_verify_client optional;
ssl_certificate /etc/letsencrypt/live/my/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/my/privkey.pem;
ssl_verify_depth 3;
# All CA certificates concatenated in a single file
ssl_client_certificate /etc/nginx/ssl/ca.pem;
ssl_crl /etc/nginx/ssl/crl/my.crl;

# Reset SSL connection. User does not have to close his browser to try connecting again
keepalive_timeout 0 0;
add_header 'Connection' 'close';
ssl_session_timeout 1s;
```

You must also export SSL_CLIENT_S_DN_CN in FastCGI params:

```
# map directive must be set in http context
map $ssl_client_s_dn $ssl_client_s_dn_cn {
           default           "";
           ~/CN=(?<CN>[^/]+) $CN; # prior Nginx 1.11.6
           #~,CN=(?<CN>[^,]+) $CN; # Nginx >= 1.11.6
```

(continues on next page)

(continued from previous page)

```
      }
fastcgi_param  SSL_CLIENT_S_DN_CN $ssl_client_s_dn_cn;
```

Nginx SSL Virtual Host example with uWSGI

```
server {
  listen 443;
  server_name authssl.example.com;
  root /usr/share/lemonldap-ng/portal/htdocs/;
  # Use "lm_app" format to get username in nginx.log (see nginx-lmlog.conf)
  access_log /var/log/nginx/access.log lm_app;

  ssl_verify_client on;
  ssl_verify_depth 3;

  # Full chain CRL is required
  # All CRLs must be concatenated in a single .pem format file
  ssl_crl /etc/nginx/ssl/crl/crls.pem;
  if ($uri !~ ^/((static|javascript|favicon).*|.*\.psgi)) {
    rewrite ^/(.*)$ /index.psgi/$1 break;
  }

  location ~ ^(?<sc>/.*\.psgi)(?:$|/) {
    # uWSGI Configuration
    include /etc/nginx/uwsgi_params;
    uwsgi_pass 127.0.0.1:5000;
    uwsgi_param LLTYPE psgi;
    uwsgi_param SCRIPT_FILENAME $document_root$sc;
    uwsgi_param SCRIPT_NAME $sc;
    uwsgi_param  SSL_CLIENT_S_DN_CN $ssl_client_s_dn_cn;
  }

  #index index.psgi;
  location / {
    try_files $uri $uri/ =404;
    add_header Strict-Transport-Security "max-age=15768000";
  }
}
```

Attention: Nginx 1.11.6 change: format of the $ssl_client_s_dn and $ssl_client_i_dn variables has been changed to follow RFC 2253 (RFC 4514); values in the old format are available in the $ssl_client_s_dn_legacy and $ssl_client_i_dn_legacy variables.

Configuration of LemonLDAP::NG

In Manager, go in `General Parameters` > `Authentication modules` and choose SSL for authentication.

Tip: You can then choose any other module for users and password.

Then, go in `SSL parameters`:

- **Authentication level**: authentication level for this module
- **Extracted certificate field**: field of the certificate affected to $user internal variable

Auto reloading SSL Certificates

A known problematic is that many browser (Firefox, Chrome) remembers the fact that the certificate is not available at a certain time. It is particularly important for smart cards: when the card is not inserted before the browser starts, the user must restart his browser, or at least refresh (F5) the page.

Apache server

It is possible with AJAX code and 3 Apache locations to bypass this limitation.

1. Modify the portal virtual host to match this example:

```
SSLEngine On
SSLCACertificateFile /etc/apache2/ssl/ca.crt
SSLCertificateKeyFile /etc/apache2/ssl/lemonldap.key
SSLCertificateFile /etc/apache2/ssl/lemonldap.crt

SSLVerifyDepth 10
SSLOptions +StdEnvVars
SSLUserName SSL_CLIENT_S_DN_CN

# DocumentRoot
DocumentRoot /var/lib/lemonldap-ng/portal/
<Directory /var/lib/lemonldap-ng/portal/>
    Order Deny,Allow
    Allow from all
    Options +ExecCGI +FollowSymLinks
    SSLVerifyClient none
</Directory>

<Location /index>
    Order Deny,Allow
    Allow from all
    SSLVerifyClient none
</Location>

<Location /testssl>
    Order Deny,Allow
    Allow from all
```

(continues on next page)

(continued from previous page)

```
    SSLVerifyClient require
</Location>

Alias /sslok /var/lib/lemonldap-ng/portal
<Location /sslok>
    Order Deny,Allow
    Allow from all
    SSLVerifyClient require
</Location>
```

- /index/ is an unprotected page to display a SSL test button
- /testssl/ is a SSL protected page to check the certificate
- /sslok/ is the new LemonLDAP::NG portal. You need to declare the new url in the manager: Portal -> URL: https://auth.example.com/sslok/

2. Then you need to construct the Ajax page, for example in /index/bouton.html. It looks like this:

```
<body>
<script src="./jquery-2.1.4.min.js"             type="text/javascript"> </script>
<!--<script src="./jquery-ui-1.8-rass.js"   type="text/javascript">  </script>-->


<a href="http://www.google.fr" class="enteteBouton" id="continuerButton"><img␣
↪src=authent.png></a>
<script>
$('.enteteBouton').click( function (e) {
  var b=navigator.userAgent.toLowerCase();
  if(b.indexOf("msie")!==-1){
    document.execCommand("ClearAuthenticationCache")
  }
  e.preventDefault();
  $.ajax({
        url:"https://auth.example.com/testssl",
        beforeSend:function(){},
        type:"GET",
        dataType:"html",
        success:function(c,a){
          if (c !== "") {
                alert("Carte OK");
                window.location.href = "https://auth.example.com/sslok/";
          }
          else {
              alert('Carte KO');
          }
        },
        error:function (xhr, ajaxOptions, thrownError){
          if(xhr.status==404) {
                alert("Carte OK");
                window.location.href = "https://auth.example.com/sslok/";
          }
          else {
              alert('Carte KO');
```

(continues on next page)

(continued from previous page)

```
          }
        },
        complete:function(c,a){}
  });
});
</script>
</body>
```

Nginx server

With Nginx, append those server context directives to force SSL connexion reset:

```
keepalive_timeout 0 0;
add_header 'Connection' 'close';
ssl_session_timeout 1s;
```

Danger: It is incompatible with authentication combination because of Apache parameter "SSLVerifyClient", which must have the value "require". To enable SSL with *Combination*, use "SSL by Ajax"

Configuration (for Combination/Choice)

If you enable this feature, you must configure 2 portal virtual hosts:

- the main *(which corresponds to portal URL)* with `SSLVerifyClient none`
- the second with `SSLVerifyClient require` and a `Header set Allow-Control-Allow-Origin https://portal-main-url`

then declare the second URL in SSL options in the Manager. That's all ! Then you can chain it in a *combination*.

Note: With *choice*, the second URL should be also declared in module URL parameter to redirect user to Portal menu.

Note: Ajax authentication request can be sent to an another URL than Portal URL.

To avoid a persistent loop between Portal and a redirection URL (pdata is not removed because domains mismatch), you have to set pdata cookie domain by editing `lemonldap-ng.ini` in section [portal]:

```
[portal]
pdataDomain = example.com
```

To avoid a bad/expired token during session upgrading (Reauthentication) if URLs are served by different load balancers, you can force Upgrade tokens to be stored into Global Storage by editing `lemonldap-ng.ini` in section [portal]:

```
[portal]
forceGlobalStorageUpgradeOTT = 1
```

Attention: **Content Security Policy** may prevent to submit Ajax Request. To avoid security warning,

Go to : `General Parameters > Advanced Parameters > Security > Content security policy`

and set :

Default value => 'self' "Ajax request URL"

Form destinations => 'self' "Ajax request URL"

Ajax destinations => 'self' "Ajax request URL"

Script source => 'self' "Ajax request URL"

Extracting the username attribute

The "Extracted certificate field" must be set to the Apache/Nginx environment variable containing the username attribute.

See the mod_ssl documentation for a list of supported variables names.

If your webserver configuration allows multiple CAs, you may configure a different environment variable for each CA.

In the "Conditional extracted certificate field", add a line for each CA.

- key: the CA subject DN (will be printed in debug logs)
- value: the variable containing the username when using certificates emitted by this CA

5.9.21 Twitter

Authentication	Users	Password
✓		

Presentation

Twitter is a famous microblogging server. Twitter use OAuth protocol to allow applications to reuse its own authentication process (it means, if your are connected to Twitter, other applications can trust Twitter and let you in).

You need Net::Twitter package, with a very recent version (>3).

You need to register a new application on Twitter to get API key and API secret. See Twitter FAQ on how to do that:.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Twitter for authentication module.

Tip: You can then choose any other module for users and password.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

Then, go in `Twitter parameters`:

- **Authentication level**: authentication level for this module.
- **API key**: API key from Twitter
- **API secret**: API secret from Twitter
- **Application name** (optional): Application name (visible in Twitter)
- **User field**: Twitter field that will be used as default user identifier. Allowed values:
  - `screen_name`
  - `user_id`

5.9.22 WebID

Authentication	Users	Password
✓	✓	

Presentation

WebID is a way to uniquely identify a person, company, organisation, or other agent using a URI and a certificate.

You need Web::ID package.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose WebID for authentication module. You can also use WebID as user database.

Then, go in `WebID parameters`:

- **Authentication level**: authentication level for this module.
- **WebID whitelist**: list of space separated hosts granted to host FOAF document. You can use '*' character. Example :`*.partner.com`

If you use WebID as user database, declare values in **exported variables** :

- use any key name you want. If you want to refuse access when a data is missing, just add a "!" before the key name
- in the value field, set the field name. Take a look at http://xmlns.com/foaf/spec/#sec-crossref. Example :`name => foaf:name`

See also *exported variables configuration*.

Apache configuration

Portal host must be configured to use SSL and must ask for client certificate. It is recommended to use optional_no_ca since WebID doesn't use certificate authorities :

```
<VirtualHost _default_:443>
ServerName auth.example.com
SSLEngine on
SSLCertificateFile ...
SSLCertificateKeyFile ...
SSLVerifyClient optional_no_ca
...
</VirtualHost>
```

Tests

To test this, you can build your own WebID certificate using one of :

- Web::ID::Certificate::Generator
- my-profile.eu
- gen-webid-cert.sh

5.9.23 Yubikey

Attention: This module has been replaced by *Yubikey Second Factor*

5.9.24 Custom authentication modules

Authentication	Users	Password
✓	✓	✓

Presentation

This artifact allows one to define its own modules (authentication, user database, password or register database).

Tip: The developer documentation is available in Portal manpages. See Auth.pod and UserDB.pod

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose 'Custom module'.

Then, you just have to define class names of your custom modules in "Custom module names". Custom parameters can be set in "Additional parameters". Full path must be specify.

You can define your own customAuth module icon. Icon must be in site/htdocs/static/common/modules/icon.png

Tip: ::Auth::My::Dev.pm means Lemonldap::NG::Portal::Auth::My::Dev

Attention: Be careful. Don' t use an already attributed name in configuration.

These parameters are available in your plugins using `$self->conf->{customAddParams}->{<customName>}`.

Read portal manpages to see how to write these plugins.

5.9.25 Backend choice by users

Authentication	Users	Password
✓	✓	✓

Presentation

By default, only the configured authentication backend is available for users.

Contrary to *multiple backend stacking*, backend choice will present all available authentication methods to users, who will choose the one they want.

The choice will concern three backends:

- Authentication
- Users
- Password

The chosen backends will be registered in session:

- `$_auth`
- `$_userDB`
- `$_passwordDB`

Authentication choice will also be registered in session:

- `$_authChoice`

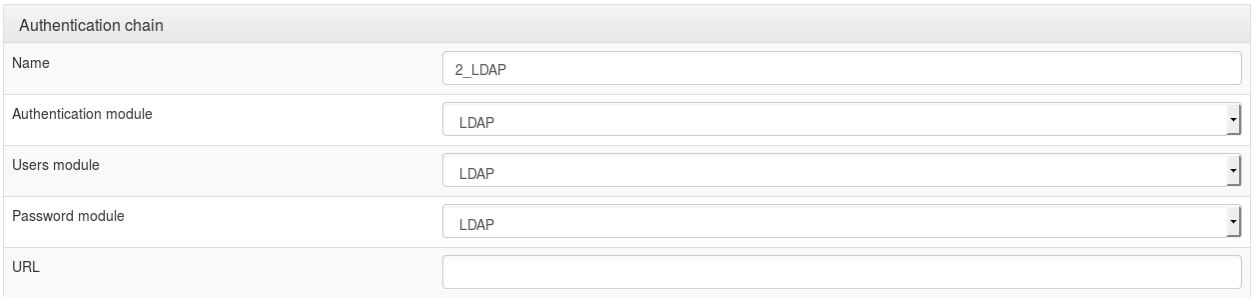
Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose Choice for authentication.

> **Attention:** When `Choice` is selected for authentication, values for Users and Password modules are also forced to `Choice`.

Then, go in `Choice Parameters`:

- **URL parameter**: parameter name used to set choice value (default: `lmAuth`)
- **Allowed modules**: click on `New chain` to add a choice.
- **Choice used for password authentication**: authentication module used by *AuthBasic handler* and *OAuth2.0 Password Grant*
- **FindUser plugin parameter**: authentication module called by Find user plugin (*Find user plugin*)



Define here:

- **Name**: Text displayed on choice tab.
- **Authentication module**
- **Users module**
- **Password module**
- **URL**: optional, can be used to redirect on another URL (for example https://authssl.example.com). This is mandatory if you want to use an Apache authentication module, which is run by Apache before showing the LemonLDAP::NG portal page.
- **Condition**: optional, can be used to evaluate an expression to display the tab. For example, to display a tab only if redirected by Handler from application `test1.example.com`, you can set this condition:

```
$env->{urldc} =~ /test1\.example\.com/
```

> **Note:** Federated authentication need pdata cookie. SameSite cookie value must be set to "Lax" or "None". See *SSO cookie parameters*

> **Note:** Authentication request to an another URL than Portal URL can lead to a persistent loop between Portal and a redirection URL (pdata is not removed because domains mismatch). To avoid this, you have to set pdata cookie domain by editing `lemonldap-ng.ini` in section [portal]:

```
[portal]
pdataDomain = example.com
```

Tip: You can prefix the key name with a digit to order them. The digit will not be shown on portal page. Underscore characters are also replaced by spaces.

Tip: You can also override some LLNG parameters for each chain. See *Parameters list* to have the key names to use

5.9.26 Combination of authentication schemes

Authentication	Users	Password
✓	✓	✓ (since *2.0.10*)

Presentation

This backend allows one to chain authentication method, for example to failback to LDAP authentication if Remote authentication failed…

Configuration

You have to use `Combination` as authentication module (users module must be set to "Same"). Then go in `Combination parameters` to :

- declare the modules that will be used
- set the rule chain

Modules declaration

Each module that will be used in combination rule must be declared. You must set:

- the name used in the rule (a uniq string)
- the type (LDAP, DBI,…)
- the scope:
  - authentication and user DB
  - authentication only
  - user DB only
- overloaded parameters: you can redefine any LLNG string parameters. For example, if you use 2 different LDAP, the first can use normal configuration and for the second, overwritten parameter can redefine ldapServer,…

Note: To overload parameters, you must select a module, add a parameter and set its value. For example:

Name	Type	Scope	Parameters
DB1	DBI	Auth only	
DB2	DBI	User DB only	dbiAuthChain => "mysql:…"

Usually, you can't declare two modules of the same type if they don't have the same parameters. For example, usually you can't declare a MySQL DBI and a PostgreSQL DBI, because there is no extra field for PostgreSQL parameters. Now with Combination, you can declare some overloaded parameters.

For example, if DBI is configured to use PostgreSQL but DB2 is a MySQL DB, you can override the "dbiChain" parameter.

You can also override a complex key like ldapExportedVars, by setting a JSON value:

```
{"cn" => "cn", "uid" => "sAMAccounName", "mail" => "mail"}
```

Attention: If your JSON is corrupted, LLNG will use it as string and just report a warning in logs.

Rule chain

Combination allows:

- to chain schemes (example: `[LDAP] and [DBI]`)
- to test different schemes (example: `[LDAP] or [DBI]`)
- to mix schemes (example: `[Kerberos,LDAP] or [LDAP,LDAP]`)
- to choose authentication scheme depending on some request values

Each scheme must be enclose in `[]`. A comma separates auth and user DB modules. If only one value is set, the same is used for both.

Boolean expression

Remember that schemes in rules are the names declared above.

Example	Explanation
`[myLDAP] or [myDBI]`	If myLDAP fails, use myDBI
`[mySSL, myLDAP] or [myLDAP, myLDAP]`	Try mySSL for auth and myLDAP for userDB. If fails, switch to myLDAP for both
`[myLDAP] or [myDBI1] or [myDBI2]`	Try myLDAP, then if it fails, myDBI1, then if it fails myDBI2
`[mySSL and myLDAP, myLDAP ]`	Use mySSL and myLDAP to authentify, myLDAP to get user

Attention: Note that "or" can't be used inside a scheme. If you think to "[mySSL or myLDAP, myLDAP]", you must write `[mySSL, myLDAP] or [myLDAP, myLDAP]`

Example	Explanation
`[myDBI1] and [myDBI2] or [myLDAP]`	Try myDBI1 and myDBI2, if it fails, try myLDAP
`[myDBI1] and [myDBI2] or [myLDAP] and [myDBI2]`	Try myDBI1 and myDBI2, if it fails, try myLDAP and myDBI2

Attention: You can't use brackets in a boolean expression and "and" has precedence on "or".

If you think to "([myLDAP] or [myDBI1]) and [myDBI2]", you must write `[myLDAP] and [myDBI2] or [myDBI1] and [myDBI2]`

Tests

Test can use only the `$env` variable. It contains the FastCGI environment variables.

Example	Explanation
`if($env->{REMOTE_ADDR} =~ /^10\./) then [myLDAP] else [mySSL, myLDAP]`	If user doesn't come from 10.0.0.0/8 network, use SSL as authentication module
`if($env->{REMOTE_ADDR} =~ /^10\./) then [myLDAP] else if($env->{REMOTE_ADDR} =~ /^192/) then [myDBI1] else [myDBI2]`	Chain tests

Attention: Note that brackets can't be used except to enclose test.

If you wants to write `if(...) then if...`, you must write `if(not ...) then ... else if(...)...`

Let's be crazy

The following rule is valid:

```
if($env->{REMOTE_ADDR} =~ /^192\./) then [mySSL, myLDAP] or [myLDAP] else [myLDAP and myDBI, myLDAP]
```

Combine second factor

Imagine you want to authenticate users either by SSL or LDAP+U2F, you can't directly write this rule: this is done in 2 steps:

- use this combination rule: `[SSL,LDAP] or [LDAP]`
- enable U2F with this rule: `$_auth eq "LDAP"` or `$_authenticationLevel < 4` *(and adapt U2F authentication level)*

Now if you want to authenticate users either by LDAP or LDAP+U2F *(to have 2 different authentication level)*, 2 possibilities:

- configure 2 portals and overwrite U2F activation in the second
- Modify login template to propose the choice *(add a "submit" button that points to the second portal)*

Display multiple forms

Combination module returns the form corresponding to the first authentication scheme available for the current request. You can force it to display the forms chosen using `combinationForms` in lemonldap-ng.ini. Example:

```
[portal]
combinationForms = standardform, openidform
```

Password management

New in version 2.0.10.

Not all configurations of the Combination module allow password management.

If your combination looks like this

```
[Kerberos, LDAP] or [LDAP]
```

Then you can simply set `LDAP` as the password module, and password changes and reset will work as expected.

If your combination looks like this

```
[LDAP1] or [LDAP2]
```

Then you can configure the `Combination` password module to automatically send password changes to the LDAP server which was used during authentication. This module also enables password reset.

Note: You can set the `_cmbPasswordDB` session variable to manually select which backend will be called when changing the password. This is useful when using SASL delegation

---

Limitations

- When using password reset with a combination of 2 or more LDAP servers, you need to make sure that there is no duplication of email addresses between all your servers. If an email exists in more than one server, the password will be reset on the first LDAP server that contains this email address
- Combinations using the `and` boolean expression will not cause passwords to be changed in both backends for now
- Forcing the user to reset their password on next login is not currently supported by the combination module

Known problems

Federation protocols

SAML, *OpenID-Connect*, *CAS* or *old OpenID* can't be chained with a "and" for authentication part. So "[SAML] and [LDAP]" isn't valid. This is because their authentication kinematic don't use the same steps.

Bad expression	Solution	Explanation
`[SAML] and [LDAP]`	`[SAML, SAML and LDAP]`	Authentication is done by SAML only but user must match an LDAP entry
`[SAML] and [LDAP] or [LDAP]`	`[SAML, SAML and LDAP] or [LDAP]`	Authentication is done by SAML or LDAP but user must match an LDAP entry

Auth::Apache authentication

When using this module, LL::NG portal will be called only if Apache does not return "401 Authentication required", but this is not the Apache behaviour: if the auth module fails, Apache returns 401. So it can be used only with a "and" boolean expression.

Tip: The new *Kerberos authentication module* solve this for Kerberos: you just have to use it instead of Apache and enable authentication by Ajax in Kerberos parameters.

Example: `[ Apache and LDAP, LDAP ]`

To bypass this, follow the documentation of *AuthApache module*

SSL authentication

To chain SSL, you have to set "SSLRequire optional" in Apache configuration, else users will be authenticated by SSL only.

Migrating from Multi

Old *Multiple backends stack* implemented only `if` and `or` keywords. Examples:

Multi expressions	Combination
`LDAP;DBI`	`[myLDAP] or [myDBI]`
`DBI $ENV{REMOTE_ADDR}=~/^192/;LDAP $ENV{REMOTE_ADDR}!~/^192/`	`if $env->{REMOTE_ADDR} then [myDBI] else [myLDAP]`

5.9.27 Multiple backends stack

Attention: This module has been removed and replaced by the more powerful *Combination of auth schemes*.

5.9.28 OpenID

Authentication	Users	Password
✓	✓	

Danger: OpenID protocol is deprecated. You should now use *OpenID Connect*.

Presentation

LL::NG can delegate authentication to an OpenID server. This requires Perl OpenID consumer module with at least version 1.0.

Tip: LL::NG can also act as *OpenID server*, that allows one to interconnect two LL::NG systems.

LL::NG will then display a form with an OpenID input, wher users will type their OpenID login.

Tip: OpenID authentication can proposed as an alternate authentication scheme using the *authentication choice* method.

LL::NG can use a white list or a black list to filter allowed OpenID domains.

If OpenID is used as users database, attributes will be requested to the server with SREG extension.

Configuration

In Manager, go in `General Parameters` > `Authentication modules` and choose OpenID for authentication and/or users.

Then, go in `OpenID parameters`:

- **Authentication level**: authentication level for this module.
- **Secret token**: used to check integrity of OpenID response.
- **Authorizated domain**:
  - **List type**: choose white list to define allowed domains or black list to define forbidden domains
  - **List**: domains list (comma separated values)

To configure requested attributes, edit **Exported variables** and define attributes:

- **Key**: internal session key, can be prefixed by `!` to make the attribute required
- **Value**: SREG attribute name:
  - fullname
  - nickname
  - language
  - postcode
  - timezone
  - country
  - gender
  - email
  - dob

See also *exported variables configuration*.

Attention: Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

In Manager, go in :

`General Parameters > Advanced Parameters > Security > Content Security Policy > Form destination`

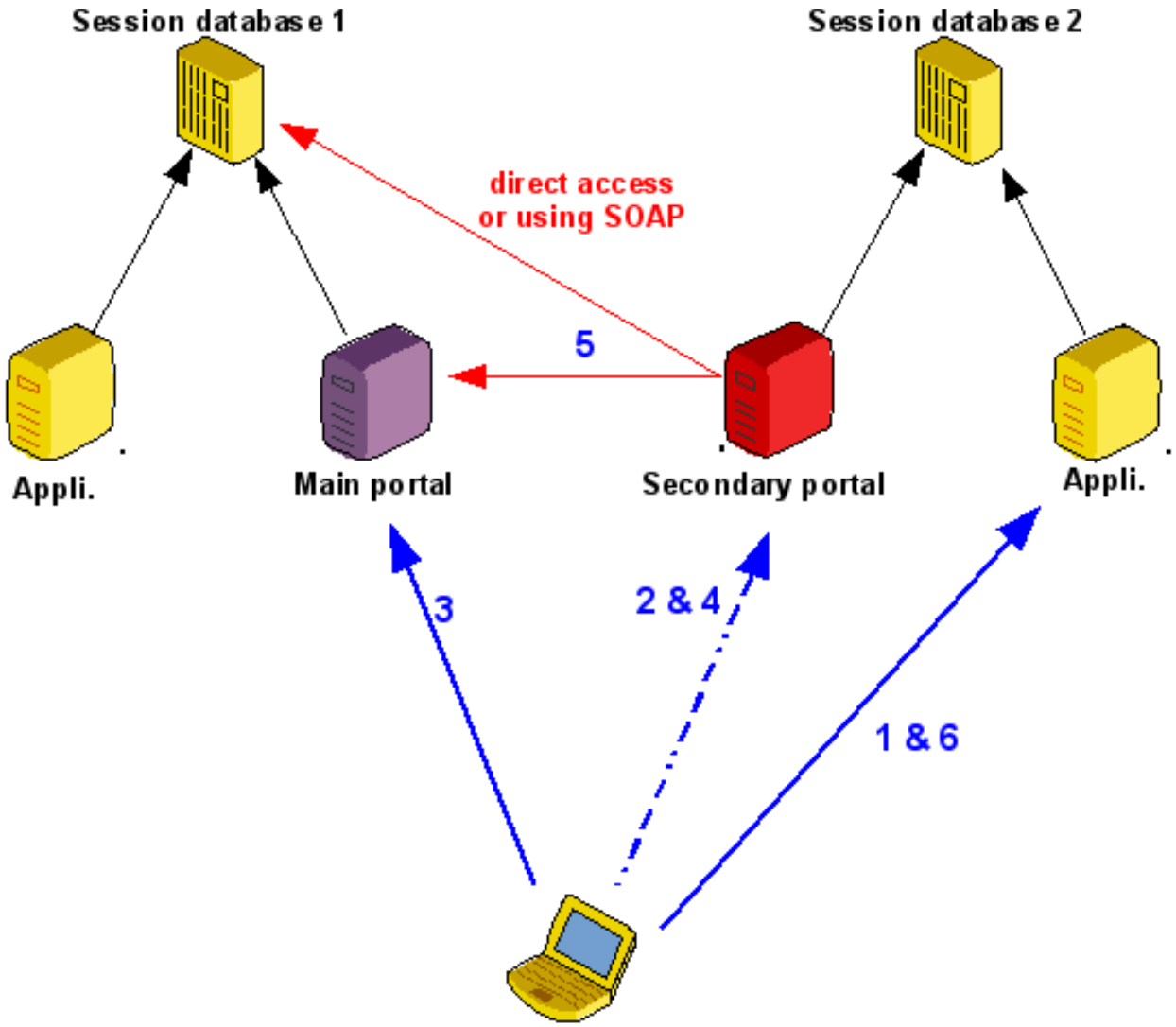
5.9.29 Remote

Authentication	Users	Password
✓	✓	

Danger: This module is a LL::NG specific identity federation protocol. You may rather use standards protocols like *SAML*, *OpenID Connect* or *CAS*.

Presentation

- The main portal is configured to use CDA. The secondary portal is declared in the Manager of the main LL::NG structure (else user will be rejected).
- The portal of the secondary LL::NG structure is configured to delegate authentication to a remote portal. A request to the main session database is done (through *SOAP session backend*) to be sure that the session exists.
- If `exportedAttr` is set, only those attributes are copied in the session database of the secondary LL::NG structure. Else, all data are copied in the session database.



1. User tries to access to an application in the secondary LL::NG structure without having a session in this area
2. Redirection to the portal of the secondary area (transparent)
3. Redirection to the portal of the main area and normal authentication (if not done before)
4. Redirection to the portal of the secondary area (transparent)
5. Secondary portal check if remote session is available. It can be done via direct access to the session database or using SOAP access. Then it creates the session (with attribute filter)
6. User can now access to the protected application

Note: Note that if the user is already authenticated on the first portal, all redirections are transparent.

Configuration

Main LL::NG structure

Go in Manager, and:

- activate CDA in `General Parameters` » `Cookies` » `Multiple domains`
- declare secondary portal in `General Parameters` » `Advanced Parameters` » `Security` » `Trusted domains`

Secondary LL::NG structure

Configure the portal to use the remote LL::NG structure.

In Manager, go in `General Parameters` » `Authentication modules` and choose Remote for authentication and users.

Then, go in `Remote parameters`:

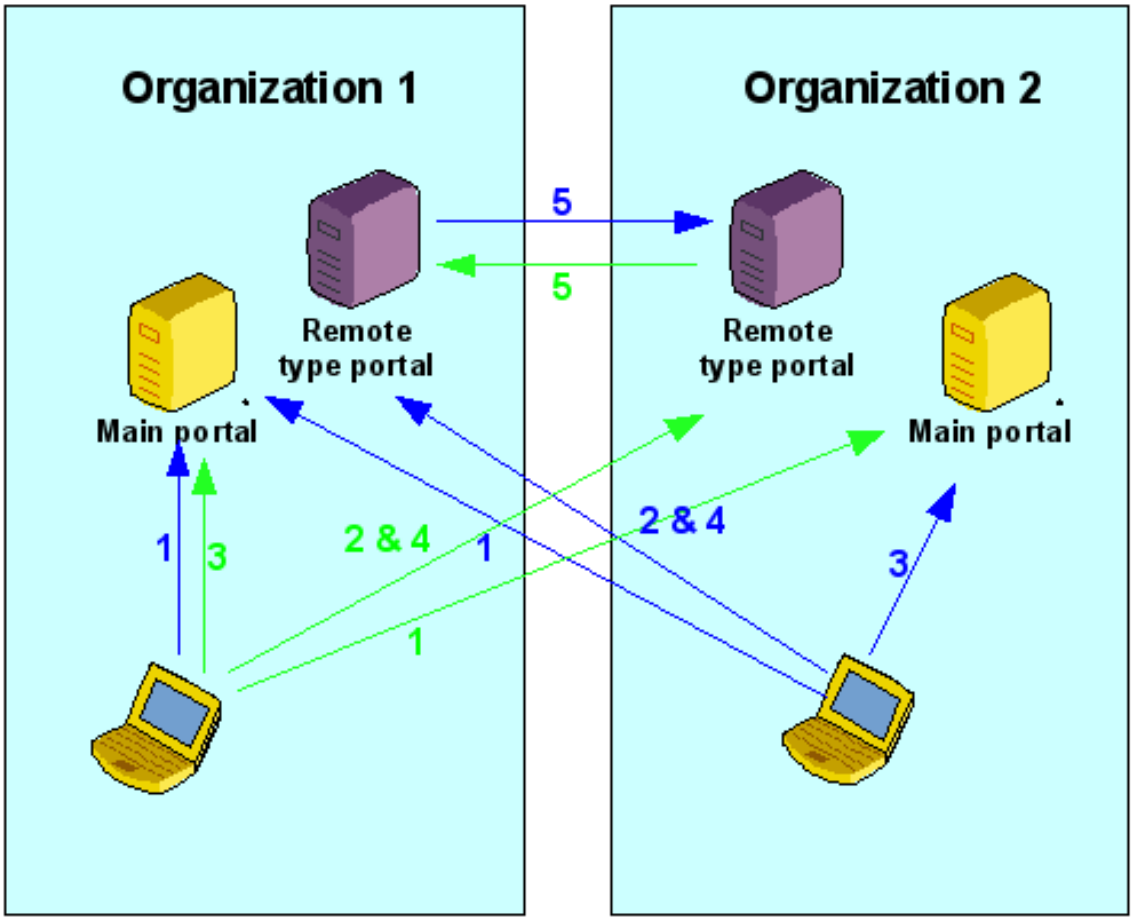
- **Portal URL**: remote portal URL
- **Cookie name** (optional): name of the cookie of primary portal, if different from secondary portal
- **Sessions module**: set `Lemonldap::NG::Common::Apache::Session::SOAP` for *SOAP session backend*.
- **Sessions module options**:
  - **proxy**: SOAP sessions end point (see *SOAP session backend* documentation)

Example: interoperability between 2 organizations

Using this, we can do a very simple interoperability system between 2 organizations using two LL::NG structures:

- each area has 2 portals:
  - One standard portal
  - One remote portal that delegates authentication to the second organization (just another file on the same server)
- The normal portal has a link included in the authentication form pointing to the remote portal for the users of the other organization

So on each main portal, internal users can access normally, and users issued from the other organization have just to click on the link:



1. One user tries to access to the portal
2. External user clicks to be redirected to the remote type portal
3. After redirection, normal authentication in the remote portal
4. Redirection to the remote type portal
5. Validation of the session: external user has now a local session

5.9.30 U2F-or-TOTP 2nd Factor Authentication

This module enables both *U2F* and *TOTP* Authentication *(like Gitlab)*. Therefore, users can use their TOTP instead if they don't have their U2F device.

Difference between enabled both U2F and TOTP is that only one page is displayed instead of displaying first a choice menu.

Configuration

In the manager (second factors), you just have to enable it:

- **Activation**: set it to "on". Note that you should not enable *U2F* and *TOTP* separately *(except for self-registration: see below)*
- **Authentication level**: you can overwrite here auth level for registered users. Leave it blank keeps auth level provided by first authentication module (By default: 2 for user/password based modules). It is recommended to set an higher value here if you want to give access to apps just for enrolled users.
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

---

Tip: Every other parameters of *U2F* and *TOTP* can be set in the corresponding 2F modules except that you should not enable them.

Attention: If you want to give a different level for U2F or TOTP, leave this parameter blank and set U2F and TOTP "authentication level" in corresponding modules.

Self-registration

This module has no self-registration. You have to use U2F and TOTP self registration modules. Example: suppose you want to allow U2F registration only if a TOTP secret is registered:

- TOTP self-registration => enabled
- U2F self-registration => `$_2fDevices =~ /"type":\s*"TOTP"/s`

Automatically, U2F registration will be hidden for unregistered TOTP users and displayed then.

5.9.31 Universal 2nd Factor Authentication (U2F)

Universal 2nd Factor (U2F) is an open authentication standard that strengthens and simplifies two-factor authentication using specialized USB or NFC devices.

LLNG can propose to users to register their keys. When done, 2F registered users can not login without using their key.

Tip: Note that it's a second factor, not an authentication module. Users are authenticated by both login form and U2F form.

Prerequisites and dependencies

This feature uses Crypt::U2F::Server::Simple.

It is available as package on Debian:

```
apt install libcrypt-u2f-server-perl
```

For other systems, use CPAN. Before compiling it, you must install Yubico's C library headers.

> **Attention:** An HTTPS portal is required to use U2F

Configuration

In the manager (second factors), you just have to enable it:

- **Activation**: set it to "on"
- **Self registration**: set it to "on" if users are authorized to register their keys
- **Authentication level**: you can overwrite here auth level for U2F registered users. Leave it blank keeps auth level provided by first authentication module *(default: 2 for user/password based modules)*. **It is recommended to set an higher value here if you want to give access to some apps only for enrolled users**
- **Allow users to remove U2F key**: If enabled, users can unregister enrolled U2F device.
- **Lifetime**: Unlimited by default. Set a Time To Live in seconds. TTL is checked at each login process if set. If TTL is expired, relative 2F device is removed.
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

> **Attention:** If you want to use a custom rule for "activation" and enable self-registration, you have to include this in your rule: `$_2fDevices =~ /"type":\s*"U2F"/s`, else U2F will be required even if users are not registered. This is automatically done when "activation" is set to "on".

Browser compatibility

- Chrome/Chromium 38
- Firefox :
  - 38 to 56 with U2F Support Add-on
  - 57 to 59, with "security.webauth.u2f" set to "true" in about:config (see Yubico explanations)
  - probably enabled by default for versions 60
- Opera 40

Enrollment

If you have enabled self registration, users can register their U2F keys using https://portal/2fregisters

Assistance

If a user loses its key, you can delete it from the manager Second Factor module. To enable manager Second Factor Administration Module, set `enabledModules` key in your `lemonldap-ng.ini` file :

```
[portal]
enabledModules = conf, sessions, notifications, 2ndFA
```

Developer corner

If you have another U2F registration interface, you have to set these keys in Second Factor Devices array (JSON) in your user-database. Then map it to the _2fDevices attribute (see *exported variables*):

```
$_2fDevices = [{"name" : "MyU2FKey" , "type" : "U2F" , "_userKey" : "########" , "_
↪keyHandle":"########" , "epoch":"1524078936"}, ...]
```

Attention: `_userKey` must be base64 encoded

Note that both "origin" and "appId" are fixed to portal URL.

5.9.32 TOTP 2nd Factor Authentication

Time based One Time Password (TOTP) is an algorithm that computes a one-time password from a shared secret key and the current time. This is currently use by Google Authenticator or FreeOTP.

LLNG can propose users to register this kind of software to increase authentication level.

Tip: Note that it's a second factor, not an authentication module. Users are authenticated both by login form and TOTP.

Prerequisites and dependencies

This feature uses libconvert-base32-perl. Before enable it, on Debian you must install libconvert-base32-perl by :

```
apt update
apt install libconvert-base32-perl
apt install libdigest-hmac-perl
```

Or from CPAN repository :

```
cpanm Convert::Base32
```

Configuration

In the manager (advanced parameters), you just have to enable it:

- **Activation**: set it to "on"
- **Self registration**: set it to "on" if users are authorized to generate themselves a TOTP secret
- **Authentication level**: you can overwrite here auth level for TOTP registered users. Leave it blank keeps auth level provided by first authentication module *(default: 2 for user/password based modules)*. **It is recommended to set an higher value here if you want to give access to some apps only to users enrolled**
- **Issuer**: default to portal hostname
- **Interval**: interval for TOTP algorithm (default: 30)
- **Range**: number of additional intervals to test (default: 1)
- **Digits**: number of digit by codes (default: 6)
- **Display existing secret**: display an already registered secret (default: disabled)
- **Change existing secret**: authorize a user to change its previoulsy registered TOTP secret
- **Allow users to remove TOTP**: If enabled, users can unregister TOTP.
- **Lifetime**: Unlimited by default. Set a Time To Live in seconds. TTL is checked at each login process if set. If TTL is expired, relative TOTP is removed.
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

Attention: If you want to use a custom rule for "activation" and want to keep self-registration, you must include this in your rule that `$_2fDevices =~ /"type":\s*"TOTP"/s` is set, else TOTP will be required even if users are not registered. This is automatically done when "activation" is simply set to "on".

Danger: Range is tested backward and forward to prevent positive or negative clock drift.

Enrollment

If you've enabled self registration, users can register their keys by using https://portal/2fregisters

Assistance

If a user loses its key, you can remove it from manager Second Factor module.// // To enable manager Second Factor Administration Module, set `enabledModules` key in your `lemonldap-ng.ini` file :// //

```
[portal]
enabledModules = conf, sessions, notifications, 2ndFA
```

Developer corner

If you have another TOTP registration interface, you have to set these keys in Second Factor Devices array (JSON) in your user-database. Then map it to the _2fDevices attribute (see *exported variables*):

```
[{"name" : "MyTOTP" , "type" : "TOTP" , "_secret" : "########" , "epoch":"1524078936"}, .
↪..]
```

5.9.33 E-Mail as Second Factor

This plugin adds the user's e-mail account as a second authentication factor.

After logging in through another authentication module, a one-time code will be generated by the portal and sent to the user's e-mail address. The user will be prompted for this code in order to finish the login process.

Attention: This plugin will only improve security in situations where the user's email is not protected by the same password used to login on LemonLDAP::NG. And of course, if the user's email account is also protected by LemonLDAP::NG, they will not be able to open their mailbox to find out their one-time code.

Configuration

Before configuring this module, make sure the user's email address is correctly fetched from your UserDB plugin and appears in the session browser. If you want to store the user e-mail in a different session field than `mail`, go to "General Parameters » Advanced parameters » SMTP" and set the "Session key containing mail address" parameter.

All parameters are configured in "General Parameters » Second factors » Mail second factor".

- **Activation**: Set to `On` to activate this module. If a user does not have an email address, they will encounter an error on login. If you want to use this plugin only for users who have an email address, use `$mail` (or whatever your e-mail session key is) as the activation rule.
- **Code regex**: The regular expression used to generate one-time codes. The default is a 6-digit code.
- **Code timeout**: It might take a while for users to open their e-mail account and find the code. Raise this timeout if the default (2 minutes) isn't enough.
- **Mail subject**: The subject of the email the user will receive. If you leave it blank, it will be looked up in translation files.
- **Mail body**: The plain text content of the email the user will receive. If you leave it blank, the `mail_2fcode` HTML template will be used. The one-time code is stored in the `$code` variable
- **Authentication level** (Optional): if you want to overwrite the value sent by your authentication module, you can define here the new authentication level. Example: 5
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

5.9.34 External Second Factor

This basic plugin can be used to add a second factor authentication device (SMS, OTP,...). It uses external commands to send or validate a second factor. Any language is allowed to call your 2nd factor system.

Commands

Commands receive arguments on command line and must return a 0 code if succeed, another else. **Nothing must be written to STDOUT**, STDERR is reported in logs (but may be lost with FastCGI server).

Configuration

All parameters are configured in "General Parameters » Portal Parameters » Extensions » External 2nd Factor".

- **Activation**
- **Code RegEx**: regular expression to create an OTP code. Let this option blank to delegate code Generation / Verification to an external provider
- **Send command**: define your command using *$attribute* like in rules. Example: `/usr/local/bin/sendOtp --uid $uid` or `/usr/local/bin/sendCode --uid $uid --code $code` if code is generated by the Portal
- **Validation command**: Required ONLY if you delegate code Generation / Verification to an external provider. You must also use *$code* which is the value entered by user; Example: `/usr/local/bin/verify --uid $uid --code $code`
- **Authentication level** (Optional): if you want to overwrite the value sent by your authentication module, you can define here the new authentication level. Example: 5
- **Logo** (Optional): logo file (in static/<skin> directory)
- **Label** (Optional): label that should be displayed to the user on the choice screen

> **Attention:** The command line is split in an array and launched with exec(). So you don't need to enclose arguments in quotes to protect your system against shell injection. However, you can not use any space except to separate arguments.

SELinux note

If your server is enforcing SELinux policies, make sure your external script has a label that is allowed to be executed by `httpd`.

For example, storing your script in `/usr/local/bin/` will give it a `bin_t` label that will work correctly.

If your script has a `httpd_sys_script_exec_t` type, it will only be able to do external network requests if the SELinux boolean `httpd_can_network_connect` is enabled.

If your script has any other label, it will probably not work at all.

5.9.35 Radius as Second Factor

Some proprietary, OTP-based second factor implementations expose a Radius server that allow an authenticating application (such as LemonLDAP::NG) to verify the validity of an OTP using the standard Radius protocol.

Tip: This page is about using Radius to connect to an external 2FA system for the *second factor only*. If your 2FA system works by concatenating the user's password and their OTP (LinOTP), you should probably be using *regular Radius authentication* instead

---

After choosing the Radius second factor type, the user is prompted with a code that will be checked against the Radius server.

Prerequisites and dependencies

This feature uses `Authen::Radius`. Before enable it, on Debian you must install it :

For CentOS/RHEL:

```
yum install perl-Authen-Radius
```

In Debian/Ubuntu, install the library through apt-get command

```
apt-get install libauthen-radius-perl
```

Configuration

Configuration

All parameters are configured in "General Parameters » Second factors » Mail second factor".

- **Activation**: Set to `On` to activate this module, or use a specific rule to select which users may use this type of second factor
- **Server hostname**: The hostname of the Radius server
- **Shared secret**: The secret key shared with the Radius server
- **Session key containing login** (Optional): When verifying the OTP code against the Radius server, use this attribute as the login and the OTP code as password. By default, the attribute designated as `whatToTrace` is used.
- **Authentication timeout** (Optional) :
- **Authentication level** (Optional): if you want to overwrite the value sent by your authentication module, you can define here the new authentication level. Example: 5
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

Vendor specific

Some configuration examples for specific vendors:

InWebo Second Factor

InWebo is a proprietary MFA solution. You can use is as second factor through *Radius 2FA module*.

Configuration

On InWebo side :

- Create a connector of type `Radius Push`.
- Fill in the "IP Address" field with the IP of the public interface of your LL::NG server.
- Enter a secret, that you will also configure on LL::NG side.

See InWebo Radius documentation for more details.

On LL::NG side, go in "General Parameters » Second factors » Radius second factor".

- **Activation**: Set to `On` to activate this module, or use a specific rule to select which users may use this type of second factor
- **Server hostname**: The hostname of InWebo Radius server (for example *radius2.myinwebo.com*)
- **Shared secret**: The secret key declared on InWebo side

See *Radius 2FA module* for more details.

5.9.36 REST Second Factor

This plugin can be used to append a second factor authentication device like SMS or OTP. It uses an external web service to submit and validate the second factor.

Configuration

All parameters are set in "General Parameters » Portal Parameters » Second Factors » REST 2nd Factor".

- **Activation**
- **Init URL** *(optional)*: REST URL to initialize dialog *(send OTP)*. Leave it blank if your API doesn't need any initialization
- **Init arguments**: list of arguments to send *(see below)*
- **Verify URL** *(required)*: REST URL to verify code
- **Verify arguments**: list of arguments to send *(see below)*
- **Authentication level** (Optional): if you want to overwrite the value sent by your authentication module, you can define here the new authentication level. Example: 5
- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

Arguments

Arguments are a list of key/value. Key is the name of JSON entry, value is attribute or macro name.

> **Attention:** For Verify URL, you should send `$code` at least

REST Dialog

REST web services have just to reply with a "result" key in a JSON file. Auth/UserDB can add an "info" array. It will be stored in session data (without reading "Exported variables").

URL	Query	Response
Init URL	JSON file: `{"user":$user,...}`	JSON file: `{"result":true/false}`
Verify URL	JSON file: `{"user":$user,"code":"$code",...}`	JSON file: `{"result":true/false}`

5.9.37 Yubikey Second Factor

A Yubikey is a small material token manufactured by Yubico. It sends an OTP, which is validated via Yubico server.

Prerequisites and dependencies

You must install Auth::Yubikey_WebClient package.

You have to retrieve a client ID and a secret key from Yubico. See Yubico API page.

Configuration

In the manager (second factors), you just have to enable it:

- **Activation**: set it to "on"
- **Self registration**: set it to "on" if users are authorized to register their keys
- **Authentication level**: you can overwrite here auth level for Yubikey registered users. Leave it blank keeps auth level provided by first authentication module *(default: 2 for user/password based modules)*. **It is recommended to set an higher value here if you want to give access to some apps only to enrolled users**
- **Client ID**: given by Yubico or another service
- **API secret key**: given by Yubico or another service
- **Nonce (optional)**: if any
- **URL**: Url of service (leave blank to use Yubico cloud services)
- **OTP public ID part size**: leave it to default (12) unless you know what you are doing
- **Allow users to remove Yubikey**: If enabled, users can unregister Yubikey device.
- **Get Yubikey ID from session attribute**: If non-empty, the Yubikey ID will be read from this session attribute. This allows external provisionning of Yubikeys.
- **Lifetime**: Unlimited by default. Set a Time To Live in seconds. TTL is checked at each login process if set. If TTL is expired, relative Yubikey is removed.

- **Logo** (Optional): logo file *(in static/<skin> directory)*
- **Label** (Optional): label that should be displayed to the user on the choice screen

Attention: If you want to use a custom rule for "activation" and want to keep self-registration, you must include this in your rule: `$_2fDevices =~ /"type":\s*"UBK"/s`, else Yubikey will be required even if users are not registered. This is automatically done when "activation" is simply set to "on".

Provisioning

If you don't want to use self-registration, set public part of user's yubikey in Second Factor Devices array (JSON) in your user-database. Then map it to the _2fDevices attribute (see *exported variables*):

```
[{"name" : "MyYubikey" , "type" : "UBK" , "_secret" : "########" , "epoch":"1524078936"},
↪ ...]
```

Enrollment

If you have enabled self registration, users can register their U2F keys using https://portal/2fregisters

5.9.38 Additional Second Factors

Starting with version 2.0.6, LemonLDAP::NG lets you configure multiple instances of second factor authentication modules, in a manner similar to the *Combination module*.

Only the following Second Factor modules are compatible with this feature:

- *E-Mail*
- *External*
- *REST*

Using this option, lets you give your users a wider range of possible second factors. They could decide between using their work email or home email. And as an administrator you may now plug in more than one Second Factor solution through *REST* or *external commands*.

Configuration

You can find the configuration for this feature in `General parameters` » `Second factors` » `Additional second factors`

- `Name`: the technical name of this second factor, it should be all lowercase, and it is used as a sort key when second factors are displayed to the user
- `Type`: what type of second factor you want to use
- `Label`: what title to display in the 2F choice screen
- `Logo` : URL path of a logo to display in the 2F choice screen
- `Level`: authentication level that will be set if this 2F is used
- `Rule`: If you leave this field empty, this second factor will always be enabled. You may use a perl expression to decide when this second factor is available.

  - `$homeMail` : this second factor will only trigger if the `$homeMail` session key exists
  - `defined $hGroups->{'admin'}` : this second factor will only trigger if the user is in the `admin` group

After adding your second factors, don't forget to add overload parameters to them. You usually should at least give them different logos so that the user can tell the difference between two second factors of the same type.

See the *parameters list* page for a full list of parameters you may overload. Here are the most useful ones:

- **E-Mail**
  - `mail2fLogo`
  - `mailSessionKey`
  - `mail2fCodeRegex`
  - `mail2fSubject`
  - `mail2fBody`
- **External**
  - `ext2fLogo`
  - `ext2fCodeActivation`
  - `ext2FSendCommand`
  - `ext2FValidateCommand`
- **REST**
  - `rest2fLogo`
  - `rest2fVerifyUrl`
  - `rest2fVerifyArgs` (must be a JSON object)
  - `rest2fInitUrl`
  - `rest2fInitArgs` (must be a JSON object)

5.10 Identity provider

5.10.1 CAS server

Presentation

LL::NG can be used as a CAS server. It can allow one to federate LL::NG with:

- Another *CAS authentication* LL::NG provider
- Any CAS consumer

LL::NG is compatible with the CAS protocol versions 1.0, 2.0 and part of 3.0 (attributes exchange).

Configuration

Enabling CAS

In the Manager, go in `General Parameters` » `Issuer modules` » `CAS` and configure:

- **Activation**: set to `On`.
- **Path**: it is recommended to keep the default value (`^/cas/`)
- **Use rule**: a rule to allow user to use this module, set to `1` to always allow.

Tip: For example, to allow only users with a strong authentication level:

```
$authenticationLevel > 2
```

Configuring the CAS Service

Then go in `CAS Service` to define:

- **CAS login**: the session key transmitted to CAS client as the main identifier (CAS Principal). This setting can be overriden per-application.
- **Access control policy**: define if access control should be done on CAS service. Three options:
  - **none**: no access control. The CAS service will accept non-declared CAS applications and ignore access control rules. This is the default.
  - **error**: if user has no access, an error is shown on the portal, the user is not redirected to CAS service
  - **faketicket**: if the user has no access, a fake ticket is built, and the user is redirected to CAS service. Then CAS service has to show a correct error when service ticket validation will fail.
- **CAS session module name and options**: choose a specific module if you do not want to mix CAS sessions and normal sessions (see *why*).
- **CAS attributes**: list of attributes that will be transmitted by default in the validate response. Keys are the name of attribute in the CAS response, values are the name of session key.
- **Use strict URL matching**: (since *2.0.12*) enforces a stricter URL matching. By default, LemonLDAP::NG will try to find a declared CAS Application matching the hostname of the requested application if it cannot find a match using the full path. See *URL Matching* for details

---

Tip: If `CAS login` is not set, it uses `General Parameters` » `Logs` » `REMOTE_USER` data, which is set to `uid` by default

---

Configuring CAS Applications

If an access control policy other than `none` is specified, applications that want to authenticate users through the CAS protocol have to be declared before LemonLDAP::NG accepts to issue service tickets for them.

Go to `CAS Applications` and then `Add CAS Application`. Give a technical name (no spaces, no special characters), like "app-example".

You can then access the configuration of this application.

Exported Attributes

You may add a list of attributes that will be transmitted in the validate response. Keys are the name of attribute in the CAS response, values are the name of session key.

The attributes defined here will completely replace any attributes you may have declared in the global `CAS Service` configuration. In order to re-use the global configuration, simply set this section to an empty list.

Options

- **Service URL** : the service (user-facing) URL of the CAS-enabled application. See *URL Matching*
- **User attribute** : session field that will be used as main identifier.
- **Authentication Level** : required authentication level to access this application
- **Rule** : The access control rule to enforce on this application. If left blank, access will be allowed for everyone.

Attention: If the access control policy is set to `none`, this rule will be ignored

Macros

You can define here macros that will be only evaluated for this service, and not registered in the session of the user.

URL Matching

Changed in version 2.0.10.

Before version 2.0.10, only the hostname was taken into account, which made it impossible to have two different CAS services behind the same hostname.

Since version 2.0.10, the entire service URL is compared to the Service URL defined in LemonLDAP::NG. The longest prefix wins.

For example, if you declared two applications in LemonLDAP::NG with the following service URLs:

- https://cas.example.com/applications/zone1
- https://cas.example.com/applications/

An application located at https://cas.example.com/applications/zone1/myapp will match the first CAS service definition

An application located at https://cas.example.com/undeclared/ will also be accepted in order to preserve the previous behavior of matching on hostnames only.

Changed in version 2.0.12: The *Strict URL matching* option now lets you decide if LemonLDAP::NG should fall back to legacy host-based matching if it cannot find a declared service matching an incoming service URL. In the previous example, https://cas.example.com/undeclared/ will no longer match if strict URL matching is enabled

5.10.2 SAML Identity Provider

Presentation

LL::NG can act as an SAML 2.0 Identity Provider, that can allow one to federate LL::NG with:

- Another LL::NG system configured with *SAML authentication*
- Any SAML Service Provider

Configuration

SAML Service

See *SAML service* configuration chapter.

IssuerDB

Go in `General Parameters` » `Issuer modules` » `SAML` and configure:

- **Activation**: set to `On`.
- **Path**: keep `^/saml/` unless you have change SAML end points suffix in *SAML service configuration*.
- **Use rule**: a rule to allow user to use this module, set to `1` to always allow.

Tip: For example, to allow only users with a strong authentication level:

```
$authenticationLevel > 2
```

Register LemonLDAP::NG on partner Service Provider

After configuring SAML Service, you can export metadata to your partner Service Provider.

They are available at the Metadata URL, by default: http://auth.example.com/saml/metadata.

You can also use http://auth.example.com/saml/metadata/idp to have only IDP related metadata.

In both cases, the entityID of the LemonLDAP::NG server is http://auth.example.com/saml/metadata

Register partner Service Provider on LemonLDAP::NG

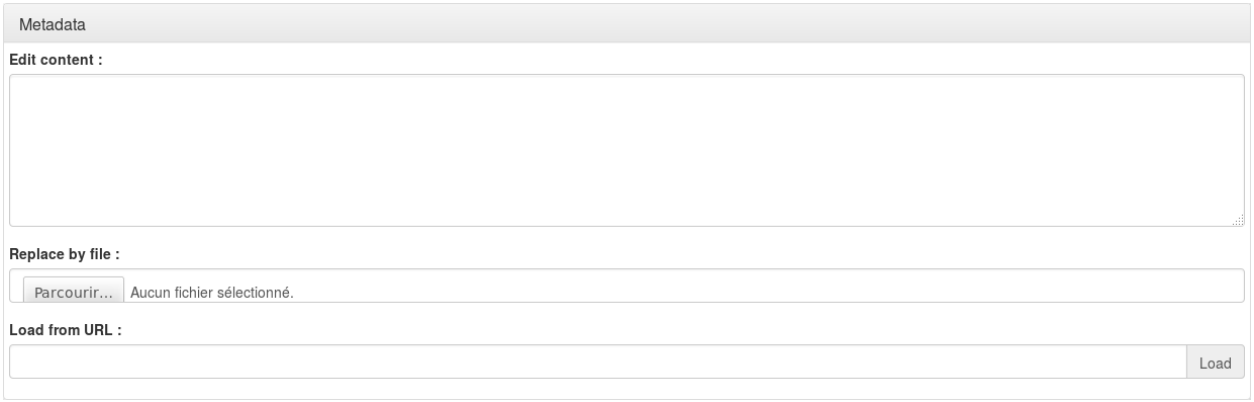
In the Manager, select node SAML service providers and click on `Add SAML SP`.

The SP name is asked, enter it and click OK.

Now you have access to the SP parameters list.

Metadata

You must register SP metadata here. You can do it either by uploading the file, or get it from SP metadata URL (this require a network link between your server and the SP).



Tip: You can also edit the metadata directly in the textarea

Exported attributes

For each attribute, you can set:

- **Variable name**: name of the variable in LemonLDAP::NG session
- **Attribute name**: name of the SAML attribute that will be seen by applications
- **Friendly Name**: optional, friendly name of the SAML attribute seen by applications
- **Mandatory**: if set to "On", then this attribute is required to build the SAML response, an error will displayed if there is no value for it. Optional attribute will be sent only if there is a value associated. Else it just will be sent through an attribute response, if explicitly requested in an attribute request.
- **Format**: optional, SAML attribute format.

Options

Authentication response

- **Default NameID format**: if no NameID format is requested, or the NameID format undefined, this NameID format will be used. If no value, the default NameID format is Email.
- **Force NameID session key**: if empty, the NameID mapping defined in *SAML service* configuration will be used. You can force here another session key that will be used as NameID content.
- **One Time Use**: set the OneTimeUse flag in authentication response (<Condtions>).
- **sessionNotOnOrAfter duration**: Time in seconds, added to authentication time, to define sessionNotOnOrAfter value in SAML response (<AuthnStatement>):

```
<saml:AuthnStatement AuthnInstant="2014-07-21T11:47:08Z"
  SessionIndex="loVvqZX+Vja2dtgt/
↪N+AymTmckGyITyVt+UJ6vUFSFkE78S8zg+aomXX7oZ9qX1UxOEHf6Q4DUstewSJh1uK1Q=="
  SessionNotOnOrAfter="2014-07-21T15:47:08Z">
```

- **notOnOrAfter duration**: Time in seconds, added to authentication time, to define notOnOrAfter value in SAML response (<Condtions> and <SubjectConfirmationData>):

```
<saml:SubjectConfirmationData NotOnOrAfter="2014-07-21T12:47:08Z"
  Recipient="http://simplesamlphp.example.com/simplesamlphp/module.php/saml/sp/saml2-acs.
↪php/default-sp"
  InResponseTo="_3cfa896ab05730ac81f413e1e13cc42aa529eceea1"/>
```

```
<saml:Conditions NotBefore="2014-07-21T11:46:08Z"
  NotOnOrAfter="2014-07-21T12:48:08Z">
```

Attention: There is a time tolerance of 60 seconds in <Conditions>

- **Force UTF-8**: Activate to force UTF-8 decoding of values in SAML attributes. If set to 0, the value from the session is directly copied into SAML attribute.

Signature

These options override service signature options (see *SAML service configuration*).

- **Signature method**: the algorithm used to sign messages sent to this service
- **Sign SSO message**
- **Check SSO message signature**: "On" means that LemonLDAP::NG will verify signatures if IDP and SP metadata require it. "Off" will disable signature verification entirely.
- **Sign SLO message**
- **Check SLO message signature**

Security

- **Encryption mode**: set the encryption mode for this SP (None, NameID or Assertion).
- **Enable use of IDP initiated URL**: set to `On` to enable IDP Initiated URL on this SP.
- **Authentication Level**: required authentication level to access this SP
- **Access Rule**: lets you specify a *Perl rule* to restrict access to this SP

Extra variables

The following environment variables are available in SAML access rules and macros:

- `$env->{llng_saml_sp}` : entityID of the SAML service
- `$env->{llng_saml_spconfkey}` : configuration key of the SAML service

New in version 2.0.10.

- `$env->{llng_saml_acs}` : AssertionConsumerServiceURL, if specified in the AuthnRequest

IDP Initiated mode

The IDP Initiated URL is the SSO SAML URL with GET parameters:

- `IDPInitiated`: `1`
- One of:
  - `sp`: Service Provider entity ID
  - `spConfKey`: Service Provider configuration key

For example: http://auth.example.com/saml/singleSignOn?IDPInitiated=1&spConfKey=simplesamlphp

- Optionally, if you may also specify, in addition to `sp` or `spConfKey`:
  - `spDest`: URL of Service Provider's AssertionConsumerService

The URL specified in `spDest` *must* be present in the Service Provider metadata registered in LemonLDAP::NG. This is only useful if your Service Provider is reachable over multiple URLs.

Macros

You can define here macros that will be only evaluated for this service, and not registered in the session of the user.

Known issues

Using both Issuer::SAML and Auth::SAML on the same LLNG may have some side-effects on single-logout.

5.10.3 OpenID server

Danger: OpenID protocol is deprecated, you should now use *OpenID Connect*

Presentation

LL::NG can act as an OpenID 2.0 Server, that can allow one to federate LL::NG with:

- Another LL::NG system configured with *OpenID authentication*
- Any OpenID consumer

LL::NG is compatible with the OpenID Authentication protocol version 2.0 and version 1.0. It can be used just to share authentication or to share user's attributes following the OpenID Simple Registration Extension 1.0 (SREG) specification.

When LL::NG is configured as OpenID identity provider, users can share their authentication using [PORTAL]/openidserver/[login] where:

- [PORTAL] is the portal URL
- [login] is the user login (or any other session information, *see below*)

Example:

```
http://auth.example.com/openidserver/foo.bar
```

Configuration

In the Manager, go in `General Parameters` » `Issuer modules` » `OpenID` and configure:

- **Activation**: set to `On`
- **Path**: keep `^/openidserver/` unless you have change *Apache portal configuration* file.
- **Use rule**: a rule to allow user to use this module, set to 1 to always allow.

Tip: For example, to allow only users with a strong authentication level:

```
$authenticationLevel > 2
```

Then go in `Options` to define:

- **Secret token**: a secret token used to secure transmissions between OpenID client and server (*see below*).
- **OpenID login**: the session key used to match OpenID login.
- **Authorized domains**: white list or black list of OpenID client domains (*see below*).
- **SREG mapping**: link between SREG attributes and session keys (*see below*).

Tip: If `OpenID login` is not set, it uses `General Parameters` » `Logs` » `REMOTE_USER` data, which is set to `uid` by default

Shared attributes (SREG)

SREG permit the share of 8 attributes:

- Nick name
- Email
- Full name
- Date of birth
- Gender
- Postal code
- Country
- Language
- Timezone

Each SREG attribute will be associated to a user session key. A session key can be associated to more than one SREG attribute.

Note: If the OpenID consumer ask for data, users will be prompted to accept or not the data sharing.

Security

- LL::NG can be configured to restrict OpenID exchange using a white or a black list of domains.
- If not set, the secret token is calculated using the general encryption key.

Attention: Note that *SAML* protocol is more secured than OpenID, so when your partners are known, prefer *SAML*.

5.10.4 OpenID Connect Provider

Presentation

Note: OpenID Connect is a protocol based on REST, OAuth 2.0 and JOSE stacks. It is described here: http://openid.net/connect/.

LL::NG can act as an OpenID Connect Provider (OP). It will answer to OpenID Connect requests to give user identity (through ID Token) and information (through User Info end point).

As an OP, LL::NG supports a lot of OpenID Connect features:

- Authorization Code, Implicit and Hybrid flows
- Publication of JSON metadata and JWKS data (Discovery)
- `prompt`, `display`, `ui_locales`, `max_age` parameters
- Extra claims definition

- Authentication context Class References (ACR)
- Nonce
- Dynamic registration
- Access Token Hash generation
- ID Token signature (HS256/HS384/HS512/RS256/RS384/RS512)
- UserInfo endpoint, as JSON or as JWT
- Request and Request URI
- Session management
- FrontChannel Logout
- BackChannel Logout
- PKCE (Since `2.0.4`) - See RFC 7636
- Introspection endpoint (Since `2.0.6`) - See RFC 7662
- Offline access (Since `2.0.7`)
- Refresh Tokens (Since `2.0.7`)

Configuration

OpenID Connect Service

See *OpenID Connect service* configuration chapter.

IssuerDB

Go in `General Parameters` » `Issuer modules` » `OpenID Connect` and configure:

- **Activation**: set to `On`.
- **Path**: keep `^/oauth2/` unless you need to use another path
- **Use rule**: a rule to allow user to use this module, set to `1` to always allow.

Tip: For example, to allow only users with a strong authentication level:

```
$authenticationLevel > 2
```

Configuration of LL::NG in Relying Party

Each Relying Party has its own configuration way. LL::NG publish its OpenID Connect metadata to ease the configuration of client.

The metadata can be found at the standard "Well Known" URL: http://auth.example.com/.well-known/openid-configuration

An example of its content:

```
{
   "end_session_endpoint" : "http://auth.example.com/oauth2/logout",
   "jwks_uri" : "http://auth.example.com/oauth2/jwks",
   "token_endpoint_auth_methods_supported" : [
      "client_secret_post",
      "client_secret_basic"
   ],
   "token_endpoint" : "http://auth.example.com/oauth2/token",
   "response_types_supported" : [
      "code",
      "id_token",
      "id_token token",
      "code id_token",
      "code token",
      "code id_token token"
   ],
   "userinfo_signing_alg_values_supported" : [
      "none",
      "HS256",
      "HS384",
      "HS512",
      "RS256",
      "RS384",
      "RS512"
   ],
   "id_token_signing_alg_values_supported" : [
      "none",
      "HS256",
      "HS384",
      "HS512",
      "RS256",
      "RS384",
      "RS512"
   ],
   "userinfo_endpoint" : "http://auth.example.com/oauth2/userinfo",
   "request_uri_parameter_supported" : "true",
   "acr_values_supported" : [
      "loa-4",
      "loa-1",
      "loa-3",
      "loa-5",
      "loa-2"
   ],
   "request_parameter_supported" : "true",
```

(continues on next page)

(continued from previous page)

```
   "subject_types_supported" : [
      "public"
   ],
   "issuer" : "http://auth.example.com/",
   "grant_types_supported" : [
      "authorization_code",
      "implicit",
      "hybrid"
   ],
   "authorization_endpoint" : "http://auth.example.com/oauth2/authorize",
   "check_session_iframe" : "http://auth.example.com/oauth2/checksession",
   "scopes_supported" : [
      "openid",
      "profile",
      "email",
      "address",
      "phone"
   ],
   "require_request_uri_registration" : "false",
   "registration_endpoint" : "http://auth.example.com/oauth2/register"
}
```

Configuration of Relying Party in LL::NG

Go in Manager and click on `OpenID Connect Relying Parties`, then click on `Add OpenID Relying Party`. Give a technical label (no spaces, no special characters), like “sample-rp”;

You can then access to the configuration of this RP.

Exported attributes

You can map here the attribute names from the LL::NG session to an OpenID Connect claim.

OpenID Connect claims

Claim name	Associated scope	Type	Example of corresponding LDAP attribute
sub	openid	string	uid
name	profile	string	cn
given_name	profile	string	givenName
family_name	profile	string	sn
middle_name	profile	string	
nickname	profile	string	
preferred_username	profile	string	displayName
profile	profile	string	labeledURI
picture	profile	string	
website	profile	string	
email	email	string	mail
email_verified	email	boolean	
gender	profile	string	
birthdate	profile	string	
zoneinfo	profile	string	
locale	profile	string	preferredLanguage
phone_number	phone	string	telephoneNumber
phone_number_verified	phone	boolean	
updated_at	profile	string	
formatted	address	string	registeredAddress
street_address	address	string	street
locality	address	string	l
region	address	string	st
postal_code	address	string	postalCode
country	address	string	co

For each OpenID Connect claim you want to release to applications, you can define:

- **Claim name**: the name of the claim as it will appear in Userinfo responses
- **Variable name**: the name of the LemonLDAP::NG session variable containing the claim value
- **Type**: the data type of the attribute. By default, a string. Choosing integer or boolean will make the claim appear as the corresponding JSON type.
- **Array**: choose how to process multi-valued attributes
  - **Auto**: If the session key contains a single value, it will be released as a JSON number, string or boolean, depending on the previously specified type. If the session key contains multiple values, it will be released as an array of numbers, strings or booleans.
  - **Always**: Return an array even if the attribute only contains one value
  - **Never**: If the session key contains a single value, it will be released as a JSON number, string or boolean. If the session key contains multiple values, it will be released as a single string with a separator character.

Attention: The specific `sub` attribute is not defined here, but in User attribute parameter (see below).

Extra Claims

Attention: By default, only claims that are part of standard OpenID Connect scopes will be sent to a client. If you want to send a claim that is not in the OpenID Connect specification, you need to declare it in the Extra Claims section

If you want to make custom claims visible to OpenID Connect clients, you need to declare them in a scope.

Add your additional scope as the **Key**, and a space-separated list of claims as the **Value**:

- timelord => rebirth_count bloodline

In this example, an OpenID Client asking for the `timelord` scope will be able to read the `rebirth_count` and `bloodline` claims from the Userinfo endpoint.

Danger: Any Claim defined in this section must be mapped to a LemonLDAP::NG session attribute in the **Exported Attributes** section

Scope Rules

New in version 2.0.12.

This feature may change in a future version in a way that breaks compatibility with existing configuration

By default, LemonLDAP::NG grants all scopes requested by the application, as long as the user consents to them.

This configuration screen allows you to change that behavior by attaching *a rule* to a particular scope.

- If the rule evaluates to true, the scope is added to the current request, even if it was not requested by the application
- If the rule evaluates to false, the scope is removed from the current request
- Scopes which are not declared in the "Scope rules" list are left untouched

When writing scope rules, you can use the special `$requested` variable. This variables evaluates to *true* within a scope rule when the corresponding scope has been requested by the application. You can use this variable in a dynamic rule when you only want to add a scope when the application requested it.

Examples:

- `read`: `inGroup('readers')`
  - the `read` scope will be granted if the user is a member of the `readers` group even if the application did not request it.
- `write`: `$requested and inGroup('writers')`
  - the `write` scope will be granted if the user is a member of the `writers` group, but only if the application requested it.

Options

- **Basic**
  - **Client ID**: Client ID for this RP
  - **Client secret**: Client secret for this RP (can be use for symmetric signature)
  - **Public client** (since version `2.0.4`): set this RP as public client, so authentication is not needed on token endpoint
  - **Redirection addresses**: Space separated list of redirect addresses allowed for this RP
- **Advanced**
  - **Bypass consent**: Enable if you never want to display the scope sharing consent screen (consent will be accepted by default). Bypassing the consent is **not** compliant with OpenID Connect standard.
  - **User attribute**: session field that will be used as main identifier (`sub`)
  - **Force claims to be returned in ID Token**: This options will make user attributes from the requested scope appear as ID Token claims.
  - **Use JWT format for Access Token** (since version `2.0.12`): When using this option, Access Tokens will use the JWT format, which means they can be verified by external OAuth2.0 resource servers without using the introspection or userinfo endpoint.
  - **Release claims in Access Token** (since version `2.0.12`): If Access Tokens are in JWT format, this option lets you release the claims defined in the *Extra Claims* section inside the Access Token itself.
  - **Additional audiences** (since version `2.0.8`): You can specify a space-separate list of audiences that will be added the audiences of the ID Token
  - **Use refresh tokens** (since version `2.0.7`): If this option is set, LemonLDAP::NG will issue a Refresh Token that can be used to obtain new access tokens as long as the user session is still valid.
- **Timeouts**
  - **Authorization Code expiration**: Expiration time of authorization code, when using the Authorization Code flow. The default value is one minute.
  - **ID Token expiration**: Expiration time of ID Tokens. The default value is one hour.
  - **Access token expiration** (since version `2.0.12`): Expiration time of Access Tokens. The default value is one hour.
  - **Offline session expiration**: This sets the lifetime of the refresh token obtained with the **offline_access** scope. The default value is one month. This parameter only applies if offline sessions are enabled.
- **Security**
  - **ID Token signature algorithm**: Select one of the available public key (RSXXX) or HMAC (HSXXX) based signature algorithms
  - **Access Token signature algorithm**: Select one of the available public key signature algorithms
  - **Userinfo signature algorithm** (since version `2.0.12`): Select one of the available signature algorithms to release user information as a JWT on the `/userinfo` endpoint. If this option is left empty, user information will be released as a plain JSON object. The `None` value will release user information as an unsigned JWT.
  - **Require PKCE** (since version `2.0.4`): a code challenge is required at token endpoint (see RFC7636)
  - **Allow offline access** (since version `2.0.7`): After enabling this feature, an application may request the **offline_access** scope, and will obtain a Refresh Token that persists even after the user has logged off. See

https://openid.net/specs/openid-connect-core-1_0.html#OfflineAccess for details. These offline sessions can be administered through the Session Browser.

- **Allow OAuth2.0 Password Grant** (since version `2.0.8`): Allow the use of the *Resource Owner Password Credentials Grant* by this client. This feature only works if you have configured a form-based authentication module.
- **Allow OAuth2.0 Client Credentials Grant** (since version `2.0.11`): Allow the use of the *Client Credentials Grant* by this client.
- **Authentication Level**: required authentication level to access this application
- **Access Rule**: lets you specify a *Perl rule* to restrict access to this client

- **Logout**
  - **Allowed redirection addresses for logout**: A space separated list of URLs that this client can redirect the user to once the logout is done (through `post_logout_redirect_uri`)
  - **URL**: Specify the relying party's logout URL
  - **Type**: Type of Logout to perform (only Front-Channel is implemented for now)
  - **Session required**: Whether to send the Session ID in the logout request

Resource Owner Password Credentials Grant

The Resource Owner Password Credentials Grant allows you to exchange a user's login and password for an access token. This must be considered a legacy form of authentication, since the Authorization Code web-based flow is prefered for all applications that support it. It can however be useful in some scenarios involving technical accounts that cannot implement a web-based authentication flow.

Changed in version 2.0.12: when using the *Choice* authentication module, the *Choice used for password authentication* setting can be used to select which authentication choice is used by the Resource Owner Password Credentials Grant. Naturally, the selected choice must be a password-based authentication method (LDAP, DBI, REST, etc.)

See also:

Specification for the Resource Owner Password Credentials Grant

Client Credentials Grant

The Client Credentials Grant allows you to obtain an Access Token using only a Relying Party's Client ID and Client Secret.

The following attributes are made available in the created session:

- The `_whatToTrace` attribute (main session identifier), is set to the relying party's configuration key
- The `_scope` attribute is set to the requested scopes
- The `_clientId` attribute is set to the Client ID that obtained the access token.
- The `_clientConfKey` attribute is set to the LemonLDAP::NG configuration key for the client that obtained the access token.

The Access Rule, if defined, will have access to those variables, as well as the *@ENV* array. You can use it to restrict the use of this grant to pre-determined scopes, a particular IP address, etc.

These session attribute will be released on the UserInfo endpoint if they are mapped to Exported Attributes and Extra Claims

See also:

Specification for the Client Credentials Grant

Macros

You can define here macros that will be only evaluated for this service, and not registered in the session of the user.

Display

- **Display name**: Name of the RP application
- **Logo**: Logo of the RP application

5.10.5 Get parameters Provider

Presentation

For application not managing other provider protocols (CAS, OpenID Connect, SAML,...) it is possible to configure LL::NG as a provider of GET parameters:

- An application can call LL::NG portal with a redirection url, such as `http://auth.example.com/get/login?url=base64(application_url)`
- When computing redirection, LL::NG portal will transmit any GET parameter you have configured for this application. (session id for example)

Danger: Passing such sensitive information can be dangerous. Using other well-known secured protocols is recommended.

There is also the possibility to trigger a logout action by passing the return url , such as `http://auth.example.com/get/logout?url=base64(return_url)`

Configuration

In the Manager, go in `General Parameters` » `Issuer modules` » `GET` and configure:

- **Activation**: set to `On`.
- **Path**: keep `^/get/` unless you have change *Apache portal configuration* file.
- **Use rule**: a rule to allow user to use this module, set to 1 to always allow.

Tip: For example, to allow only users with a strong authentication level:

```
$authenticationLevel > 2
```

Then go in `Get parameters` to define variables to transmit:

- Define a new virtual host
- Declare all get parameters you need. You have access to any *variable or macro* (but no perl expression).

For example:

```
"test1.example.com" => {
    "id" => "_session_id",
}
```

Danger: In the previous example, _session_id is quite sensitive, thus it is encouraged that the application revalidate _session_id using getCookie() SOAP call to avoid some security problems

Tip: If host is not already registered in virtual hosts, you need to declare it in *trusted domains* to allow redirection

5.11 Attacks and Protection

5.11.1 Brute Force Protection plugin

This plugin prevents brute force attack. Plugin DISABLED by default.

After some failed login attempts, user must wait before trying to log in again.

The aim of a brute force attack is to gain access to user accounts by repeatedly trying to guess the password of an user. If disabled, automated tools may submit thousands of password attempts in a matter of seconds.

Attention: This plugin relies on the Login History, stored in users' persistent sessions. This means that the authentication and persistent session backends will be accessed for every login attempt, even fraudulent ones. This plugin is not meant to protect against denial of service attacks.

Configuration

To enable Brute Force Attack protection:

Go in Manager, `General Parameters` » `Advanced Parameters` » `Security` » `Brute-force attack protection` » `Activation`and set to `On`.

- **Parameters**:
  - **Activation**: Enable/disable brute force attack protection
  - **Lock time**: Waiting time before another login attempt
  - **Allowed failed login**: Number of failed login attempts allowed before account is locked
  - **Incremental lock**: Enable/disable incremental lock times
  - **Incremental lock times**: List of comma separated lock time values in seconds

Incremental lock time enabled

You just have to activate it in the Manager :

Go in Manager, `General Parameters` » `Advanced Parameters` » `Security` » `Brute-force attack protection` » `Incremental lock times` and set to `On`. (DISABLED by default) or in `lemonldap-ng.ini` [portal] section:

```
[portal]
bruteForceProtectionIncrementalTempo = 1
```

Lock time increases between each failed login attempt after allowed failed logins.

```
[portal]
bruteForceProtectionLockTimes = 5, 15, 60, 300, 600
bruteForceProtectionMaxLockTime = 900
```

Note: Max lock time value is used if a lock time is missing (number of failed logins higher than listed lock time values). Lock time values can not be higher than max lock time.

Incremental lock time disabled

After allowed failed login attempts, user must wait the lock time before trying to log in again. To modify delta (MaxAge) between current and last stored failed login (300 seconds by default) edit `lemonldap-ng.ini` in [portal] section:

```
[portal]
bruteForceProtectionTempo = 30
bruteForceProtectionMaxAge = 300
bruteForceProtectionMaxFailed = 3
```

Attention: Number of failed login attempts history might be also higher than number of incremental lock time value plus allowed failed login attempts. Incremental lock time values list will be truncated if not.

Danger: Number of failed login attempts stored in history MUST be higher than allowed failed logins for this plugin takes effect. See *History plugin*

5.11.2 Safe jail

Presentation

LemonLDAP::NG uses Safe jail to evaluate all expressions:

- Access rules
- Headers
- Form replay parameters
- Macros

- Groups
- Conditions:
  - Menu modules display
  - Multi modules display
  - IssuerDB use
  - Session opening

More information about Safe on CPAN

Disable Safe jail

Safe can be very annoying when using *extended functions* or *custom functions*. In this case, you might want to disable it.

To do this, go into Manager > General Parameters > Advanced Parameters > Security > Use Safe Jail and disable it.

5.11.3 Assignment test

Presentation

Perl comparaisons are done by using `eq` for strings or == for integers. To avoid an unwanted assignment like `$authLevel = 5` (BAD EXPRESSION!), you can enable `Avoid assignment in expressions` option.

To do this, go into Manager > General Parameters > Advanced Parameters > Security > Avoid assignment in expressions and enable it.

DISABLE by default.

5.12 Plugins

5.12.1 Adaptative Authentication Level

Presentation

A user obtain an authentication level depending on which authentication module was used, and eventually which second factor module.

This plugin allows to adapt this authentication level depending on other conditions, like network, device, etc.

Sample use case: a strategic application is configured to require an authentication level of 5. Users obtain level 2 with their login/password and level 5 using a TOTP second factor. You can trust users form internal network by incrementing their authentication level based on their IP address, they would then not be forced to use 2FA to access the strategic application.

Tip: This use case works if you enable the *Use 2FA for session upgrade* option.

Configuration

This plugin is enabled when at least one rule is defind.

To configure rules, go in `General Parameters` > `Plugins` > `Adapative Authentication Level`.

You can then create rules with these fields:

- **Rule**: The condition that will be evaluated. If this condition does not return true, then the level is not changed.
- **Value**: How change the authentication level. First character is +, - or =, the second part is the number to add, remove or set.

Tip: By example, to add 3 to authentication level for users from 192.168.0.0/24 network:

- Rule: `$env->{REMOTE_ADDR} =~ /^192\.168\./`
- Value: `+3`

5.12.2 Auto Signin Addon

Auto-Signin plugin provides an easy way to bypass authentication process based on rules. For example, a TV can be automatically authenticated by its IP address.

Configuration

This add-on is automatically enabled if a rule is defined. A rule links rule to an username. The only availble variable here is `$env`. Example:

Key (username)	Rule
dwho	''$env->{REMOTE_ADDR} eq '192.168.42.42' ''

Attention: Username must be defined in users database.

5.12.3 Cross Domain Authentication

Presentation

Cross Domain Authentication (CDA)

Configuration

Go in Manager, `General Parameters` » `Cookies` » `Multiple domains` and set to `On`.

To use this feature only locally, edit `lemonldap-ng.ini` in section [all]:

```
[all]
cda = 1
```

Attention: If your handler is being served by Nginx, you have to uncomment the following lines in your nginx configuration file:

```
# If CDA is used, uncomment this
auth_request_set $cookie_value $upstream_http_set_cookie;
add_header Set-Cookie $cookie_value;
```

Handlers

Choose "CDA" as type for each virtualHost concerned by CDA *(ie not in main domain)*.

5.12.4 Check DevOps plugin

This plugin can be used to check the *DevOps* file.

Configuration

Just enable it in the manager (section "plugins").

- **Parameters**:
  - **Activation**: Enable / Disable this plugin
  - **Download file**: Allow users to download DevOps file from a remote server by providing an URL (By example: http://myapp.example.com:8080). Plugin will try to retrieve remote file by sending a request (i.e. http://myapp.example.com:8080/rules.json)

Usage

When enabled, `/checkdevops` URL path is handled by this plugin. Then, you can paste a file to test your rules and headers.

Example

DevOps handler requires a rules.json file to define access rules and headers:

```
{
  "rules": {
    "^/admin": "$uid eq 'admin'",
    "default": "accept"
  },
  "headers": {
    "Auth-User": "$uid"
  }
}
```

Note: This plugin displays ALL user session attributes except the hidden ones.

You have to restrict access to specific users like DevOps teams by setting an access rule like other VirtualHosts.

By example: `$groups =~ /\bdevops\b/`

Attention: Be careful to not display secret attributes.

checkDevOps plugin uses hidden attributes option.

5.12.5 Check state plugin

This plugin can be used to check if portal instance is ready. This can be a health check to request keep-alive service to force a fail-over on the backup-node.

Configuration

To enable Check state: Go in Manager, `General Parameters` » `Plugins` » `State Check`. You must also set a shared secret.

Usage

When enabled, `/checkstate` URL path is handled by this plugin. GET parameters:

GET Parameter	Need	Value
`secret`	required	Same value as the shared secret given to the manager
`user`	optional	If set (with password), a login/logout process will be tried
`password`	optional	

Example

- Basic availability check: `https://auth.example.com/checkstate?secret=qwerty`
- Try to log a user in: `https://auth.example.com/checkstate?secret=qwerty&user=dwho&password=dwho`

5.12.6 Check user plugin

This plugin allows us to check session attributes, access rights and transmitted headers for a specific user and URL. This can be useful for IT Ops, dev teams or administrators to debug or check rules. Plugin DISABLED by default.

Configuration

Just enable it in the manager (section "plugins").

- **Parameters**:
  - **Activation**: Enable / Disable this plugin
  - **Identities use rule**: Rule to define which profiles can be displayed (by example: `!$anonymous`)
  - **Unrestricted users rule**: Rule to define which users can check ALL users. `Identities use rule` is bypassed.
  - **Hidden attributes**: Session attributes not displayed

  - **Attributes used for searching sessions**: User's attributes used for searching sessions in backend if `whatToTrace` fails. Useful to look for sessions by mail or givenName. Let it blank to search by `whatToTrace` only
  - **Hidden headers**: Sent headers whose value is masked except for unrestricted users. Key is a Virtualhost name and value represents a space-separated headers list. A blank value obfuscates ALL relative Virtualhost sent headers. Note that just valued hearders are masked.

- **Display**:
  - **Computed sessions**: Rule to define which users can display a computed session if no SSO session is found
  - **Empty headers**: Rule to define which users can display ALL headers appended by LemonLDAP::NG including empty ones
  - **Normalized headers**: Rule to define which users can see headers name sent by the web server (see RFC3875)
  - **Empty values**: Rule to define which users can display ALL attributes even empty ones
  - **Persistent session data**: Rule to define which users can display persistent session data

---

Note: By example:

* test1.example.com => `Auth-User mail` Just 'Auth-User' and 'mail' headers are masked if valued.

* test2.example.com => '' ALL valued headers are masked.

Unrestricted users can see the masked headers.

Note: By example:

* Search attributes => `mail uid givenName`

If `whatToTrace` fails, sessions are searched by `mail`, next `uid` if none session is found and so on...

* Display empty headers rule => `$uid eq "dwho"` -> Only 'dwho' will see empty headers

---

Note: Keep in mind that Nginx HTTP proxy module gets rid of empty headers. If the value of a header field is an empty string then this field will not be passed to a proxied server. To avoid misunderstanding, it might be useful to not display empty headers.

Attention: Be careful to not display secret attributes.

checkUser plugin hidden attributes are concatenation of `checkUserHiddenAttributes` and `hiddenAttributes`. You just have to append checkUser specific attributes.

Danger: This plugin displays ALL user session attributes except the hidden ones.

You have to restrict access to specific users (administrators, DevOps, power users and so on...) by setting an access rule like other VirtualHosts.

By example: `$groups =~ /\bsu\b/`

To modify persistent sessions attributes ('_loginHistory _2fDevices notification_' by default), edit `lemonldap-ng.ini` in [portal] section:

```
[portal]
persistentSessionAttributes = _loginHistory _2fDevices notification_
```

Usage

When enabled, `/checkuser` URL path is handled by this plugin.

Attention: With federated authentication, checkUser plugin works only if a session can be found in backend.

5.12.7 CrowdSec

Presentation

CrowdSec is a free and open-source security automation tool leveraging local IP behavior detection and a community-powered IP reputation system.

LL::NG provides a **CrowdSec** bouncer that can reject Crowdsec banned-IP requests or just provide an environment variable that can be used in another plugin rule. For example, a second factor may be required if user's IP is CrowdSec-banned.

Configuration

To configure bouncer plugin, go in `General Parameters` > `Plugins` > `CrowdSec`.

You can then configure:

- **Activation**: enable this plugin *(default: disabled)*
- **Action**: reject **or** warn and set `$env->{CROWDSEC_REJECT} = 1`
- **Base URL of local API**: base URL of CrowdSec local API *(default: http://localhost:8080)*
- **API key**: API key, usually given by `cscli bouncers add mylemon`

5.12.8 Viewer module

This module can be useful to allow certain users to edit WebSSO configuration in Read Only mode.

Configuration

Parameters are set in `lemonldap-ng.ini` file, section [manager]:

```
[manager]
enabledModules = conf, sessions, notifications, 2ndFA, viewer

defaultModule = viewer

viewerHiddenKeys = samlIDPMetaDataNodes samlSPMetaDataNodes managerPassword ManagerDn␣
↪globalStorageOptions persistentStorageOptions
```

(continues on next page)

(continued from previous page)

```
viewerAllowBrowser = $groups =~ /\bsu\b/
viewerAllowDiff = $groups =~ /\bsu\b/
```

- **Parameters**:
  - **enabledModules**: list of modules to enable
  - **defaultModule**: module displayed by default route (http://manager.example.com/manager.(fcgi|psgi)
  - **viewerHiddenKeys**: keys not displayed by Viewer
  - **viewerAllowBrowser**: allow to browse other configurations
  - **viewerAllowDiff**: enable "difference with previous" link

Danger: You have to set access rules to allow/deny users to access modules.

In Manager: * Declare a Virtual Host : manager.example.com * Set an access rule for each enabled module :

1. Configuration : ^/(.*?.(fcgi|psgi)/)?(manager.html|confs|$) = $uid eq 'dwho'
2. Notifications : ^/(.*?.(fcgi|psgi)/)?notifications = $uid eq 'dwho'
3. Sessions : ^/(.*?.(fcgi|psgi)/)?sessions = $uid eq 'dwho'
4. Viewer : ^/(.*?.(fcgi|psgi)/)?viewer = $uid =~ /b(?:dwho|rtyler)b/
5. Default : $uid =~ /b(?:dwho|rtyler)b/

Attention: To avoid that Read-Only users can access to configuration module by using default route, keep in mind to set 'defaultModule' option

5.12.9 ContextSwitching plugin

This plugin allows certain users to switch context other user. This may be useful when providing assistance or when testing privileges. Enter the uid of the user you'd like to switch context to.

Configuration

Just enable it in the Manager (section "plugins") by setting a rule. ContextSwitching can be allowed or denied for specific users. Furthermore, specific identities like administrators or anonymous users can be forbidden to assume.

- **Parameters**:
  - **Use rule**: Rule to enable or define which users may use this plugin (By example: $uid eq 'dwho' && $authenticationLevel > 2).
  - **Identities use rule**: Rule to define which identities can be assumed. Useful to prevent impersonation of certain sensitive identities like CEO, administrators or anonymous/protected users.
  - **Unrestricted users rule**: Rule to define which users can switch context of ALL users. `Identities use rule` is bypassed.
  - **Allow 2FA modifications**: This option must be enabled to append, verify or delete a second factor during context switching.

  - **Stop by logout**: Stop context switching by sending a logout request.

Danger: During context switching authentication process, all plugins are disabled. In other words, all entry points like afterData, endAuth and so on are skipped. Therefore, second factors or notifications by example will not be prompted and login history is not updated!

Attention: ContextSwitching plugin works only with a userDB backend. You can not switch context with federated authentication.

Attention: Used identity, start and end of switching context process are logged!

contextSwitchingPrefix is used to store real user's session Id. You can set this prefix ('switching' by default) by editing `lemonldap-ng.ini` in [portal] section:

```
[portal]
contextSwitchingPrefix = switching
```

5.12.10 Decrypt value plugin

This plugin allows us to decrypt ciphered values. LL::NG can be configured to send encrypted values to protected applications by using *extended functions*.

Configuration

Just enable it in the Manager (section "plugins") by setting a rule. DecryptValue plugin can be allowed or denied for specific users.

- **Parameters**:
  - **Use rule**: Select which users may use this plugin
  - **Decrypt functions**: Set functions used for decrypting ciphered values. Each function is tested until one succeeds. Let it blank to use internal decrypt function.

Danger: Custom functions must be defined into `Lemonldap::NG::Portal::My::Plugin` and set:

```
My::Plugin::function1 My::Plugin::function2
```

5.12.11 Login History

Presentation

LemonLDAP::NG allows one to store user logins and login attempts in their persistent session.

Users can see their own history in menu, if menu module `Login history` is enabled.

Session history is always visible in session explorer for administrators.

Configuration

This feature can be enabled and configured in Manager, in `General Parameters` » `Plugins` » `Login History`. You can define how many logins and failed logins will be stored.

A login is considered as successful if user get authenticated and is granted a session; as failed, if he fails to authenticate or if he is not allowed to open a session. In other cases which result on impossibility to authenticate user, to retrieve data or to create a session, nothing is stored.

By default, login time and IP address are stored in history, and the error message prompted to the user for failed logins. It is possible to store any additional session data. For example to store authentication mode, you can set in `Session data to store` a new key `_auth` with value `Authentication mode`. The value will be used to display the data.

To allow the Login History tab in Menu, configure it in `General Parameters > Portal > Menu > Modules` (see *portal menu configuration*).

You can also display a check box on the authentication form, to allow user to see their login history before being redirected to the protected application (see *portal customization*).

5.12.12 Force Authentication Addon

forceAuthentication plugin forces users to authenticate again to access to Portal. Plugin DISABLED by default.

Users can access all protected applications except Portal.

Users have to authenticate again to access to Portal if there last login is older than 5 seconds by default.

Configuration

To enabled forceAuthentication plugin :

Go in Manager, `General Parameters` » `Advanced Parameters` » `Security` » `Force authentication` and set to `On`.

To modify last login interval (5 seconds by default) edit `lemonldap-ng.ini` in section [portal]:

```
[portal]
portalForceAuthnInterval = 5
```

5.12.13 Global logout plugin

This plugin allows a user to log out of all his active sessions.

Configuration

Just enable it in the Manager (section "plugins").

- **Parameters**:
  - **Activation**: Enable/Disable or set a rule to select which users are allowed to close there sessions.
  - **Auto accept time**: Enable/Disable timer. If timer is disabled, all opened sessions will be immediately closed.
  - **Custom parameter**: Session attribut to display at global logout

---

Note: To display more than one session attribute, you can create a macro like this :

```
user_USER => "$uid_" . uc $uid
```

5.12.14 Grant Session

Presentation

The goal of this plugin is to evaluate different conditions before allowing a user to open a session on the portal. When a condition is not met, then the user is prompted with a customized message.

Configuration

This plugin is enabled by default.

To configure rules, go in `General Parameters` > `Sessions` > `Opening conditions`.

You can then create rules with these fields:

- **Comment**: a label for your rule, than can be used to order it (rules are evaluated by alphabetical order).
- **Rule**: The condition that will be evaluated. If this condition does not return true, then the session is refused.
- **Message**: The message that will be displayed. That message can contain session data as user attributes or macros.

Tip: By example, you can set a message like this: "hello $uid your are not allowed to login"

5.12.15 Impersonation plugin

This plugin allows certain users to assume the identity of another user. A privileged user first logs in with its real account and can then choose another profile to appear as. This feature can be especially useful for training/learning or development platforms.

Attention: This plugin should not be used on production instance, prefer *ContextSwitching plugin*.

Configuration

Just enable it in the Manager (section "plugins") by setting a rule. Impersonation can be allowed or denied for specific users. Furthermore, specific identities like administrators or anonymous users can be protected from being impersonated.

- **Parameters**:
  - **Use rule**: Rule to allow/deny users to impersonate or define which users may use this plugin.
  - **Identities use rule**: Rule to define which identities can be assumed. Useful to prevent impersonation of certain sensitive identities like CEO, administrators or anonymous/protected users
  - **Unrestricted users rule**: Rule to define which users can assume ALL users. `Identities use rule` is bypassed.
  - **Hidden attributes**: Attributes not displayed
  - **Skip empty values**: Do not use empty profile attributes
  - **Merge spoofed and real SSO groups**: Can be useful for administrators to keep higher privileges. "Special rule" field can be used to set SSO groups to merge if exist in real session. Multivalue `separator` is used. By example : `su; admins; anonymous`

Danger: You HAVE TO modify **REMOTE_USER** to log both real AND spoofed uid.

Set a macro like this :

```
_whatToTrace -> $real__user ? "$real__user/$_user" :  "$_user/$_user"
```

and set `General Parameters > Logs > REMOTE_USER` with `_whatToTrace`

Attention: Both spoofed and real session attributes can be used to set access rules, groups or macros.

By example : `$real_uid && $real_uid eq 'dwho'` or `$real_groups && $real_groups =~ /\bsu\b/`

Keep in mind that real session is computed first. Afterward, if access is granted, impersonated session is computed with real and spoofed session attributes if Impersonation is allowed. So, `real_` attributes are computed by second authentication process. To avoid Perl warnings, you have to prefix regex with `$real_var &&`.

Attention: By example, to prevent impersonation as 'dwho' set **Identities use rule** like :

```
$uid ne 'dwho'
```

impersonationPrefix is used to rename user's real profile attributes. You can set real attributes prefix ('real_' by default) by editing `lemonldap-ng.ini` in section [portal]:

```
[portal]
impersonationPrefix = real_
```

5.12.16 Find user plugin

This plugin allows unauthenticated users to search for an user account to impersonate. This may be useful to randomly provide an identifier depending on allowed searching attributes and excluded values.

Attention: FindUser plugin works only if *Impersonation plugin* is enabled.

Configuration

Just enable it in the Manager (section "plugins"). Then, set searching attributes used for selecting accounts and randomly suggest one of them in login form. Excluding attributes can also be defined to exclude some user accounts and avoid to provide them.

- **Parameters**:
  - **Activation**: Enable / Disable this plugin
  - **Character used as wildcard**: Character that can be used by users as wildcard. An empty value disable wildcarded search requests
  - **Parameters control**: Regular expression used for checking searching values syntax
  - **User accounts URL**: User database URL to search on if REST backend is used. Let it blank to use default user data URL.
  - **Searching attributes**: For each attribute, you have to set a key (attribute as defined in UserBD) and a value that will be display in login form (placeholder). A value can be a multivalued list separated by multiValuesSeparator parameter (General Parameters > Advanced parameters > Separator). See note below.
  - **Excluding attributes**: You can defined here attributes used for excluding accounts. Set keys corresponding to UserBD attributes and values to exclude. A value can be a multivalued list separated by multiValuesSeparator parameter (General Parameters > Advanced parameters > Separator)

Note: You can provide a 'multiValuesSeparator' separated list of allowed searching values that will be displayed as an HTML <select> list

```
attribute#placeholder[#empty] => value1; placeholder1; value2; placeholder2
```

For example

```
uid#Identity   => dwho; Dr Who; rtyler; Rose Tyler; msmith; Mr Smith

uid#Identity#1 => dwho; Dr Who; rtyler; Rose Tyler (allow empty value)
```

Entries are sorted by alphabetical order.

Attention: LDAP filter works only if an objectClass is set.

Attention: Searching request is built based on provided parameters value depending on users backend like this:

request => searchAttr1=value && searchAttr2=value && not excludeAttr1=value && not excludeAttr2=value

Danger: This plugin works only with a users backend and of course if the searching or excluding attributes are existing.

Danger: With AuthChoice, you must set which module will be called by this plugin (*Backend choice by users*).

5.12.17 Notifications system

LemonLDAP::NG can be used to notify some messages to users. If a user has got some messages, they will be displayed when he access to the portal. If a message contains some check boxes, the user has to check all of them else he can not access to the portal and retrieves his session cookie.

A notification explorer is available in Manager, and notifications can be set for all users, with possibility to use display conditions. When the user accept the notification, notification reference is stored in his persistent session.

Installation

Activation

To activate notifications system:

Go to Manager `General Parameters` » `Plugins` » `Notifications` » `Activation`

or in `lemonldap-ng.ini` [portal] section:

```
[portal]
notification = 1
```

Explorer

Notifications explorer allows users to see and display theirs accepted notifications. Disable by default, you just have to activate it in the Manager (`General Parameters` » `Plugins` » `Notifications` » `Explorer`)

or in `lemonldap-ng.ini` [portal] section:

```
[portal]
notificationsExplorer = 1
```

By default, just the three last notifications are displayed. You can modify this by editing `lemonldap-ng.ini` [portal] section:

```
[portal]
notificationsMaxRetrieve = 3
```

Usage

When enabled, `/mynotifications` URL path is handled by this plugin.

Known issue

An XML document can contain several notifications messages. Just the first one can be searched and displayed!

Attention: Listed notifications are extracted from users persistent session (notification reference and accepted date). ONLY the notifications explorer can found in notifications backend are available to be displayed. Notifications content (title, subtitle and so on…) is not stored into persistent session.

Storage

By default, notifications will be stored in the same database as configuration:

- if you use "File" system and your "dirName" is set to /usr/local/lemonldap-ng/conf/, the notifications will be stored in /usr/local/lemonldap-ng/notifications/
- if you use "CDBI" or "RDBI" system, the notifications will be stored in the same database as configuration and in a table named "notifications".
- if you use "LDAP" system, the notifications will be stored in the same directory as configuration and in a branch named "notifications".

You can change default parameters using the "notificationStorage" and "notificationStorageOptions" parameters with the same syntax as configuration storage parameters. To do this in Manager, go in General Parameters > Plugins > Notifications.

File

Parameters for File backend are the same as *File configuration backend*.

Attention: You need to create yourself the directory and set write access to Apache user. For example:

```
mkdir /usr/local/lemonldap-ng/notifications/
chown www-data /usr/local/lemonldap-ng/notifications/
```

Tip: The file name default separator is _, this can be a problem if you register notifications for users having _ in their login. You can change the separator with the `fileNameSeparator` option, and set another value, for example `@`.

To summary available options:

- **dirName**: directory where notifications are stored.

- **fileNameSeparator**: file name separator.

DBI

Parameters for DBI backend are the same as *DBI configuration backend*.

Attention: You have to create the table by yourself:

```
CREATE TABLE notifications (
  date datetime NOT NULL,
  uid varchar(255) NOT NULL,
  ref varchar(255) NOT NULL,
  cond varchar(255) DEFAULT NULL,
  xml longblob NOT NULL,
  done datetime DEFAULT NULL,
  PRIMARY KEY (date, uid,ref)
)
```

To summary available options:

- **dbiChain**: DBI connection.
- **dbiUser**: DBI user.
- **dbiPassword**: DBI password.
- **dbiTable**: Notifications table name.

LDAP

Parameters for LDAP backend are the same as *LDAP configuration backend*.

Attention: You have to create the branch by yourself

To summary available options:

- **ldapServer**: LDAP URL.
- **ldapBindDN**: LDAP user.
- **ldapBindPassword**: LDAP password.
- **ldapConfBase**: Notifications branch DN.

Note: DBI configuration example:

```
notificationStorage = DBI
notificationStorageOptions={  \
    'dbiChain'    => 'DBI:Pg:dbname=llng;host=mabdd;port=5432', \
    'dbiTable'    => 'notifications', \
    'dbiUser'     => 'user', \
    'dbiPassword' => 'qwerty', \
```

(continues on next page)

(continued from previous page)

```
    'type'        => 'CDBI', \
}
```

Wildcard

The notifications module uses a wildcard to manage notifications for all users. The default value of this wildcard is `allusers`, but you can change it if `allusers` is a known identifier in your system.

To change it, go in General Parameters > Plugins > Notifications > Wildcard for all users, and set for example `alluserscustom`.

Then creating a notification for `alluserscustom` will display the notification for all users.

Using notification system

Attention: Since version 2.0, notifications are now stored in JSON format. If you want to keep old format, select "use old format" in the Manager. Note that notification server depends on chosen format: REST for JSON and SOAP for XML.

Notification format

Notifications are JSON (default) or XML files containing:

- <notification> element(s) :
  - Required attributes:
    - date: creation date (format YYYY-MM-DD WITHOUT time!)
    - ref: a reference that can be used later to know what has been notified and when (Avoid _ character)
    - uid: the user login (it must correspond to the attribute set in whatToTrace parameter, uid by default), or the wildcard string (by default: `allusers`) if the notification should be displayed for every user.
  - Optional attributes:
    - condition: condition to display the notification, can use all session variables.
  - Sub elements:
    - <title>: title to display: will be inserted in HTML page enclosed in <h2 class="notifText">...</h2>
    - <subtitle>: subtitle to display: will be inserted in HTML page enclosed in <h2 class="notifText">...</h2>
    - <text>: paragraph to display: will be inserted in HTML page enclosed in ...
    - <check>: paragraph to display with a checkbox: will be inserted in HTML page enclosed in <input type="checkbox" />...

Attention: All other elements will be removed including HTML elements like <b>.

Tip: One notification XML document can contain several notifications messages.

Several notifications can be inserted with a single request by using an array of JSON (Tested with an array of 10,000 elements)

Examples

JSON

```
[{
"uid": "foo",
"date": "2009-01-27",
"reference": "ABC",
"title": "You have new authorizations",
"subtitle": "Application 1",
"text": "You have been granted to access to appli-1",
# An array is required to set multi checkboxes
"check": [
  "I agree",
  "Yes, I'm sure"
]
},
{
"uid": "bar",
"date": "2009-01-27",
"reference": "ABC",
"title": "You have new authorizations",
"subtitle": "Application 1",
"text": "You have been granted to access to appli-1",
"check": "I agree"
}] # No comma at the end
```

Tip: JSON format notifications are displayed sorted by date and reference

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
<notification uid="foo.bar" date="2009-01-27" reference="ABC">
<title>You have new authorizations</title>
<subtitle>Application 1</subtitle>
<text>You have been granted to access to appli-1</text>
<subtitle>Application 2</subtitle>
<text>You have been granted to access to appli-2</text>
<subtitle>Acceptation</subtitle>
<check>I know that I can access to appli-1 </check>
<check>I know that I can access to appli-2 </check>
```

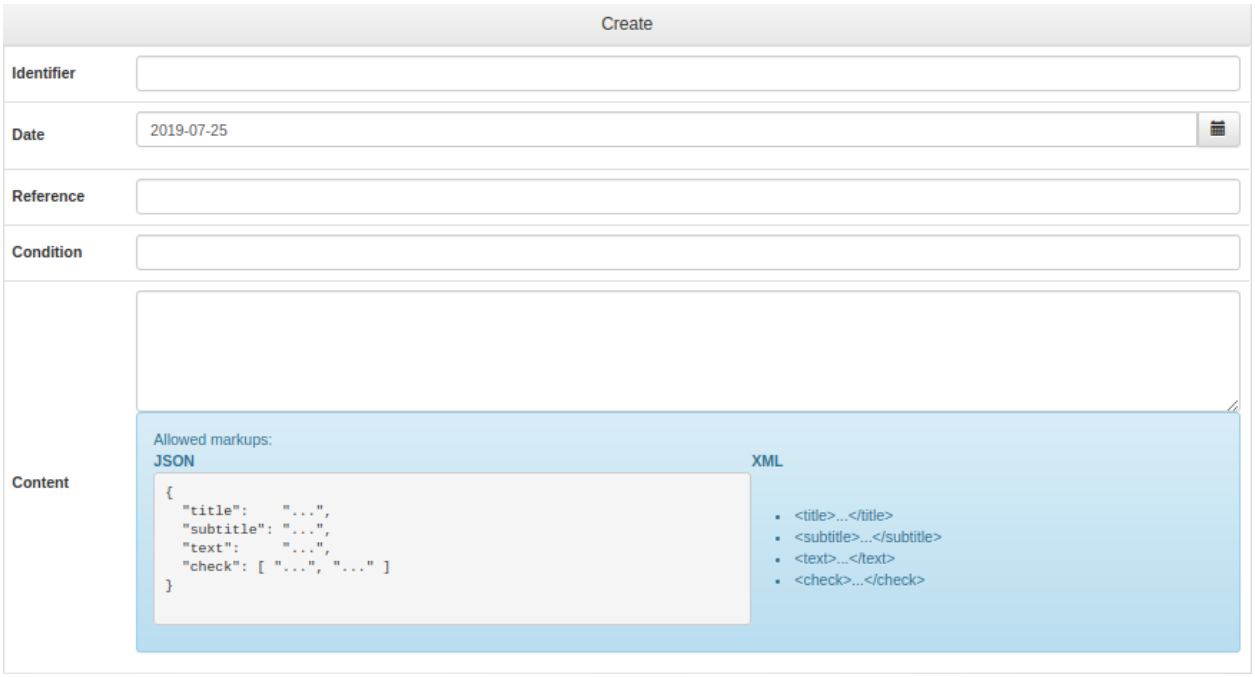
(continues on next page)

(continued from previous page)

```
</notification>
<notification uid="allusers" date="2009-01-27" reference="disclaimer" condition="$ipAddr
↪=~ /^192/">
<title>This is your first access on this system</title>
<text>Be a nice user and do not break it please.</text>
<check>Of course I am not evil!</check>
</notification>
</root>
```

Create new notifications with notifications explorer

In Manager, click on `Notifications` and then on the `Create` button.



Then fill all inputs to create the notification. Only the condition is not mandatory.

When all is ok, click on `Save`.

Notification server

LemonLDAP::NG provides two notification servers : SOAP and REST depending on format.

If enabled, the server URL is https://auth.your.domain/notifications.

Notification server provides three API to insert (POST), delete (DELETE) or list (GET) notification(s).

Available options:

- **Server**: Enable/Disable notification server
- **Default condition**: Condition appended to ALL notifications inserted by notification server (JSON format only)
- **Notification parameters to send**: Notifications parameters returned by `GET` method

- **HTTP methods**: Enable/Disable HTTP methods

Attention: If notification server is enabled, you have to protect this URL by using the web server because there is no authentication required to use it.

Example:

```
# REST/SOAP functions for insert/delete/list notifications (disabled by default)
<LocationMatch ^/(index\.fcgi/)?notifications>
    <IfVersion >= 2.3>
        Require ip 192.168.2.0/24
    </IfVersion>
    <IfVersion < 2.3>
        Order Deny,Allow
        Deny from all
        Allow from 192.168.2.0/24
    </IfVersion>
</LocationMatch>
```

XML notifications through SOAP

If you use old XML format, new notifications can be inserted or deleted by using SOAP request, once SOAP is activated:

* Insertion example in Perl

```
#!/usr/bin/perl

use SOAP::Lite;
use utf8;

my $lite = SOAP::Lite
        ->uri('urn:Lemonldap::NG::Common::PSGI::SOAPService')
        ->proxy('http://auth.example.com/notifications');


$r = $lite->newNotification(
'<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
<notification uid="foo.bar" date="2009-01-27" reference="ABC">
<text> You have been granted to access to appli-1 </text>
<text> You have been granted to access to appli-2 </text>
<check> I know that I can access to appli-1 </check>
<check> I know that I can access to appli-2 </check>
</notification>
</root>
');

if ( $r->fault ) {
    print STDERR "SOAP Error: " . $r->fault->{faultstring};
```

(continues on next page)

(continued from previous page)

```
}
else {
    my $res = $r->result();
    print "$res notification(s) have been inserted\n";
}
```

* Deletion example in Perl

```
#!/usr/bin/perl

use SOAP::Lite;
use utf8;

my $lite = SOAP::Lite
        ->uri('urn:Lemonldap::NG::Common::CGI::SOAPService')
        ->proxy('http://auth.example.com/index.pl/notification');


$r = $lite->deleteNotification('foo.bar', 'ABC');

if ( $r->fault ) {
    print STDERR "SOAP Error: " . $r->fault->{faultstring};
}
else {
    my $res = $r->result();
    print "$res notification(s) have been deleted\n";
}
```

JSON notifications through REST

Insertion example with REST API

Using JSON, you just have to POST json files.

For example with curl:

```
curl -X POST -H "Content-Type: application/json" -H "Accept: application/json" -d @notif.
↪json http://auth.example.com/notifications
```

Deletion example with REST API

DELETE API is available with LLNG 2.0.6

For example with curl:

```
curl -X DELETE -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications/<uid>/<reference>
```

List example with REST API

GET API is available with LLNG 2.0.6

For example with curl:

```
# Retrieve 'wildcard' notifications
curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications

# Retrieve all pending notifications
curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications/_allPending_

# Retrieve all existing notifications
curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications/_allExisting_

# Retrieve all <uid>'s notifications
curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications/<uid>

# Retrieve <uid>/<reference> notification parameters
curl -X GET -H "Content-Type: application/json" -H "Accept: application/json" http://
↪auth.example.com/notifications/<uid>/<reference>
```

Test notification

You've just to insert a notification and connect to the portal using the same UID. You will be prompted.

Try also to create a global notification (to the uid "allusers"), and connect with any user, the message will be prompted.

5.12.18 Status pages

Portal Status (experimental)

The Portal displays in JSON format its activity. It can provide a view of all returned codes.

Configuration

- Ordered List ItemSet `portalStatus = 1` in lemonldap-ng.ini file (section `[Portal]`)
- Note that handler status must also been enabled
- The URL http://portal/status must be protected by your webserver configuration

Handler Status

Presentation

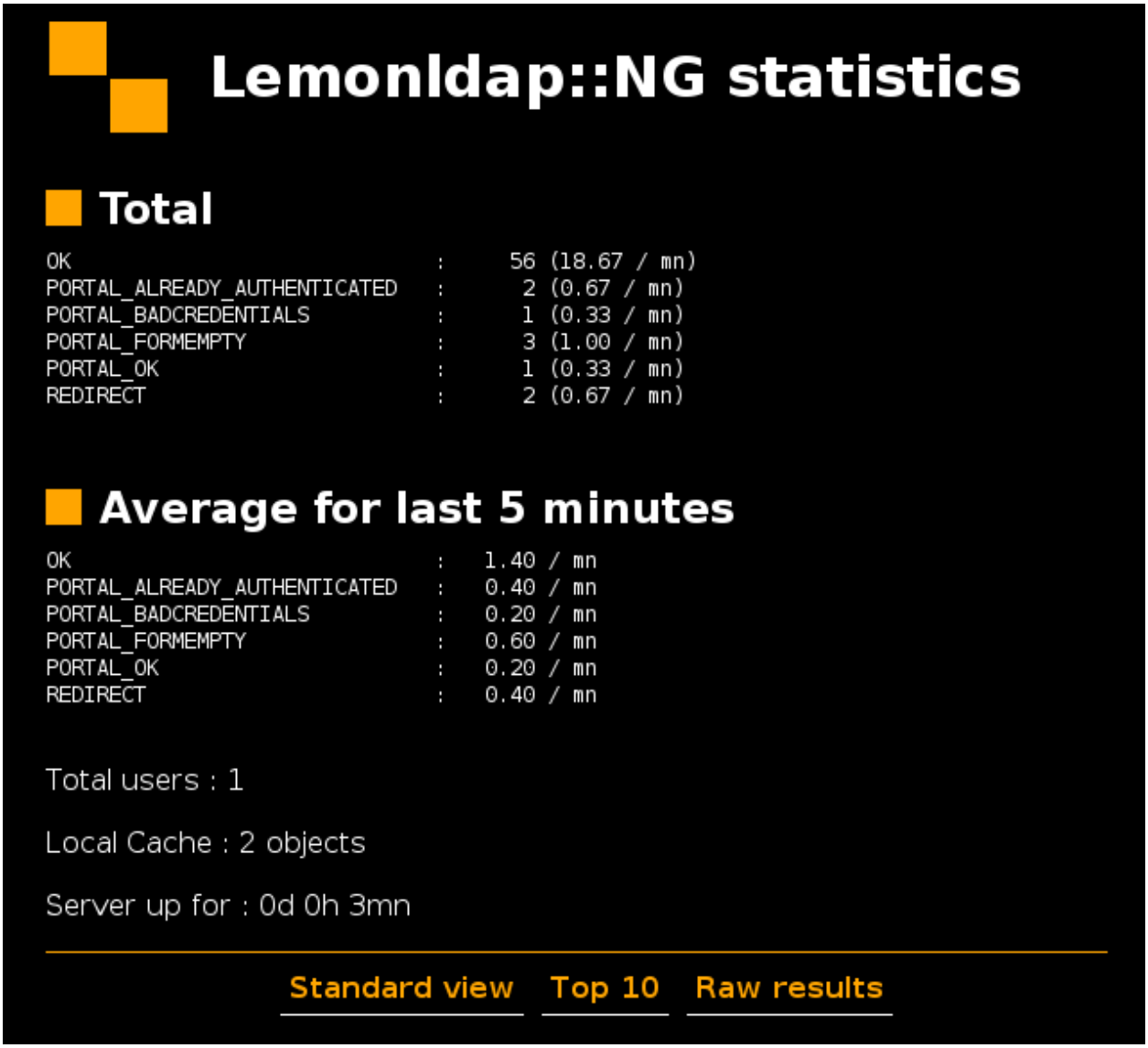
When status feature is enabled, Handlers and portal will collect statistics and save them in their local cache. This means that if several Handlers are deployed, each will manage its own statistics.

Tip: This page can be browsed for example by MRTG using the *MRTG monitoring script*.

Statistics are collected through a daemon launched by the Handler. It can be supervised in system processes.

The statistics are displayed when calling the status path on an Handler (for example: http://reload.example.com/status).

Example of status page:



Configuration

Nginx

You need to give access to status path in the Handler Nginx configuration:

```
server {
  listen __PORT__;
  server_name reload.__DNSDOMAIN__;
  root /var/www/html;
  ...
  location = /status {
    allow 127.0.0.1;
    deny all;
    include /etc/nginx/fastcgi_params;
```

(continues on next page)

(continued from previous page)

```
    fastcgi_pass unix:__FASTCGISOCKDIR__/llng-fastcgi.sock;
    fastcgi_param LLTYPE status;
  }
}
```

Apache

You need to give access to status path in the Handler Apache configuration:

```
# Uncomment this to activate status module
<Location /status>
    Order deny,allow
    Allow from 127.0.0.0/8
    PerlHeaderParserHandler Lemonldap::NG::Handler->status
</Location>
```

Then restart Apache.

Tip: You should change the `Allow` directive to match administration IP, or use another Apache protection mean.

Portal data

By default Apache handler status process listen to `localhost:64321` *(UDP)*. You can change this using `LLNGSTATUSLISTEN` environment variable. If you want to collect portal data, you just have to set `LLNGSTATUSHOST` environment variable *(see comments in our ``portal-apache2.conf``)*.

```
<Files *.fcgi>
  SetHandler fcgid-script
  # For Authorization header to be passed, please uncomment one of the following:
  # for Apache >= 2.4.13
  #CGIPassAuth On
  # for Apache < 2.4.13
  #RewriteCond %{HTTP:Authorization} ^(.*)
  #RewriteRule .* - [e=HTTP_AUTHORIZATION:%1]
  Options +ExecCGI
  header unset Lm-Remote-User
</Files>
FcgidInitialEnv LLNGSTATUSHOST 127.0.0.1:64321
```

LemonLDAP::NG

Edit `lemonldap-ng.ini`, and activate status in the `handler` section:

```
[all]
# Set status to 1 if you want to have the report of activity (used for
# example to inform MRTG)
status = 1
```

Then restart webserver.

Advanced

1. You can also open the UDP port with Nginx if you set `LLNGSTATUSLISTEN` environment variable *(host:port)*
2. When querying status *(using portal or handler status)* and if UDP is used, query is given to `LLNGSTATUSHOST` *(host:port)* and response is waiting on a dynamic UDP port given in query *(between 64322 and 64331)*. By default this dynamic UDP port is opened on loopback *(``localhost`` entry in ``/etc/hosts``)*. To change this, set an IP address or a host using `LLNGSTATUSCLIENT` environment variable.

5.12.19 Refresh session plugin (API)

This plugin appends an endpoint to refresh sessions by user. It provides `https://portal/refreshsession` endpoint. Protect it by webserver configuration.

This plugin is available with LLNG 2.0.7.

Usage

This endpoint accepts only POST requests with a JSON content:

Request	Response
`{"uid":"userid"}`	`{"updated":1,"errors":0}`

5.12.20 Reset password by mail

Presentation

LL::NG can propose a password reset form, for users who loose their password (this kind of application is also called a self service password interface).

Kinematics:

- User clicks on the link `Reset my password`
- User enters his email (or another information) in the password reset form
- LL::NG try to find the user in users database with the given information
- A mail with a token is sent to user
- The user click on the link in the mail
- LL::NG validate the token and propose a password change form

- The user can choose a new password or ask to generate one
- The new password is sent to user by mail if user ask to generate one, else the mail only confirm that the password was changed

---

Tip: If *LDAP backend* is used, and LDAP password policy is enabled, the 'password reset flag is set to true when password is generated, so that the user is forced to change his password on next connection. This feature can be disabled in *LDAP configuration*.

Tip: If the user do a new password reset request but there is already a request pending, the user can ask the confirmation mail to be resent. The request validity time is a configuration parameter.

Configuration

The reset password link must be activated, see *portal customization*.

The SMTP server must be setup, see *SMTP server setup*.

Then go in Manager, `General Parameters` » `Plugins` » `Password management` :

- **Password reset mail content**:
  - **Success mail subject**: Subject of mail sent when password is changed (default: [LemonLDAP::NG] Your new password)
  - **Success mail content** (optional): Content of mail sent when password is changed
  - **Confirmation mail subject**: Subject of mail sent when password change is asked (default: [LemonLDAP::NG] Password reset confirmation)
  - **Confirmation mail content** (optional): Content of mail sent when password change is asked

Attention: By default, mail content are empty in order to use HTML templates:

- portal/skins/common/mail_confirm.tpl
- portal/skins/common/mail_password.tpl

If you define mail contents in Manager, HTML templates will not be used.

- **Other**:
  - **Page URL**: URL of password reset page (default: [PORTAL]/resetpwd)
  - **Validity time of a password reset request**: number of seconds for password reset request validity. During this period, user can ask the confirmation mail to be resent (default: session timeout value)
  - **Display generate password box**: display a checkbox to allow user to generate a new password instead of choosing one (default: disabled)

```
* **Regexp for password generation**: Regular expression used to generate the password␣
↪(default: [A-Z]{3}[a-z]{5}.\d{2})
```

5.12.21 Certificate reset

Presentation

This plugin allows users to reset their certificate informations.

Kinematics:

- User click reset certificate button.
- He enters his mail.
- LL::NG looks for the user in users database with given information.
- An email with a link is sent if user exists.
- User clicks on the link and he is redirected to the portal.
- The portal asks him to upload his certificate file (base64, pem only).
- A confirmation mail is sent to confirm the certificate has been successfully reset.

> **Danger:** LDAP backend supported only

Configuration

Requirements

You have to activate the certificate reset link in the login page, go in Manager, `General Parameters` → `Portal` → `Customization` → `Buttons on login page`→ `Reset your Certificate`

The SMTP server must be setup, see *SMTP server setup*.

The register module also must be setup. Go in Manager, `General Parameters` → `Authentication parameters` → `Register Module` and choose your module.

Manager Configuration

Go in Manager, `General Parameters` → `Plugins` → `Certificate Reset Management`:

Certificate reset mail content:

- **Certificat reset mail subject**: Subject of mail sent when certificate is reset
- **Certificat reset mail content**: (optional): Content of mail sent when certificate is reset
- **Confirmation mail subject**: Subject of mail sent when certificate reset is asked
- **Confirmation mail content**: (optional) Content of mail sent when certificate is asked

> **Attention:** By default, mail contents are empty in order to use templates:
>
> - portal/skins/common/mail_certificateConfirm.tpl
> - portal/skins/common/mail_certificateReset.tpl
>
> If you define custom mail contents in Manager, then templates won't be used.

Other

- **Reset Page URL**: URL of certificate reset page (default: [PORTAL]/certificateReset)

- **Certificate descrition attribute Name**: Attribute where to save certificate description name (Default description)
- **Certificate hash attribute Name**: Attribute where to store certificate hash (Default userCertificate;binary)
- **Minimun duration before expiration**: number of days of validity before certificate expires. Default 0.

Danger: .p12 certificates only.

5.12.22 REST services

LL::NG portal is a REST server that gives access to configuration, session and also authentication.

Portal REST services

Authentication

The authentication service is always available with REST, you just need to send credentials on portal URL. But by default, the portal is protected by *one time tokens to prevent CSRF*. You must disable them or set a rule (configuration parameter `requireToken`) so token will not be required for REST requests, for example:

```
$env->{HTTP_ACCEPT} !~ m:application/json:
```

API

Request parameters:

- Endpoint: `/`
- Method: `POST`
- Request headers:
  - `Accept`: `application/json`
- POST data:
  - `user`: user login
  - `password`: user password
  - xxx: optional parameters, like `lmAuth` if your portal uses `Choice` or `spoofId` to impersonate.

The JSON response fields are:

- `result`: authentication result, `0` if it fails, `1` if it succeed
- `error`: error code, the corresponding error can be found in `Lemonldap::NG::Portal::Main::Constants`
- `id`: if authentication succeed, the session id is returned in this field

Tip: You can also get the cookie by reading the response header `Cookie` returned by the portal.

Attention: Before version 2.0.4, the response to a success authentication had no `id` field, and `error` field was named `code` (use Cookie header to get id value).

Example

- Request with curl:

```
curl -H "Accept: application/json" -d user=rtyler -d password=rtyler http://auth.example.
↪com/ | json_pp
```

Attention: With `cURL > 7.18.0`, to include special characters like @, & or + in the cURL POST data:

```
curl -H "Accept: application/json" -d name=rtyler --data-urlencode passwd=@31&3+*J␣
↪http://auth.example.com/ | json_pp
```

- Response for bad authentication:

```
{
    "result" : 0,
    "error" : 5
}
```

- Response for good authentication:

```
{
    "result" : 1,
    "error" : "0",
    "id" : "b048bf87ca401da1d89419813e3acf466d5e4465fe3a1f7adfd8240bd161bde2"
}
```

Sessions

REST functions for sessions are protected by Web Server, you can change this in *portal configuration*.

See *REST session backend documentation* for more.

Configuration

REST functions for configuration are protected by Web Server, you can change this in *portal configuration*.

See *REST configuration backend documentation* for more.

5.12.23 SOAP services (deprecated)

LL::NG portal provide a SOAP server that can be enable to give configuration and/or session. These features can be enabled using the manager.

Portal SOAP services

SOAP functions are not accessible by network by default. SOAP functions are protected by Web Server, you can change this in *portal configuration*.

- Read-only functions (index.pl/sessions or index.pl/adminSessions paths):
  - **getCookies(user,password)**: authentication system. Returns cookie(s) name and values
  - **getAttributes(cookieValue)**: get elements stored in session
  - **isAuthorizedURI(cookieValue,url)**: check if user is granted to access to the function
  - **getMenuApplications(cookieValue)**: return a list of authorizated applications (based on menu calculation)
- Read/Write functions (index.pl/adminSessions paths):
  - **setAttributes(cookieValue,hashtable)**: update a session
  - **newSession**: create a session (return attributes)
  - **deleteSession**: delete a session
  - **get_key_from_all_sessions**: list all sessions and return asked keys
- Notification send function (index.pl/notification):
  - **newNotification(xmlString)**: insert a notification for a user (see *Notifications system* for more)
- Notification delete function:
  - **deleteNotification**: delete notification(s) for a user (see *Notifications system* for more)

Attention: When you use *SOAP sessions backend*, it is recommended to use read-only URL (/index.fcgi/sessions). Write session path is needed only if you use a remote session explorer or a remote portal

WSDL

You can enable WSDL server in the manager. It will deliver WSDL file (/portal.wsdl).

5.12.24 Stay connected plugin

This plugin enables persistent connection. It allows us to connect automatically from the same browser.

Configuration

Just enable it in the manager (section “plugins”).

- **Parameters**:
  - **Activation**: Enable / Disable this plugin
  - **Expiration time**: Persistent session connection and cookie timeout
  - **Cookie name**: Persistent connection cookie name

CHAPTER

SIX

HANDLERS

6.1 AuthBasic Handler

6.1.1 Presentation

The AuthBasic Handler is a special Handler using AuthBasic method to authenticate and grante access to a virtual host.

The Handler sends a WWW-Authenticate header to the client, to request user id and password. Then it checks credentials by using LL::NG REST web service (REST session service must be enabled in the manager). Once session is granted, the Handler will check authorizations like the standard Handler.

This feature can be useful to allow a third party application to access a virtual host with user credentials by sending a Basic challenge to it.

6.1.2 Configuration

Portal

REST server must be enabled on portal.

Virtual host

You just have to set "Type: AuthBasic" in the virtualHost options in the manager.

If you want to protect only a virtualHost part, keep type on "Main" and set type in your configuration file:

- Apache: use simply a `PerlSetVar VHOSTTYPE AuthBasic`
- Nginx: create another FastCGI with a `fastcgi_param VHOSTTYPE AuthBasic;` *(and remove error_page 401)*

Handler parameters

No parameters needed. But you have to allow REST sessions web services, see *REST sessions backend*, enable local cache (enabled by default in lemonldap-ng.ini) and allow source IP addresses to access required locations in Portal Virtual Host.

Danger: With AuthBasic handler, you have to disable CSRF token by setting a special rule based on source IP addresses like this :

requireToken => $env->{REMOTE_ADDR} !~ /^127.0.[1-3].1$/

With *Backend choice by users*, you have to declare which authentication module is requested by handler to create global session.

Go to: `General Parameters > Authentication parameters > Choice parameters`

and set authentication module's name :

Choice used for password authentication => 2_LDAP (by example)

Attention: With HTTPS, you may have to set **LWP::UserAgent object** with `verify_hostname => 0` and `SSL_verify_mode => 0`.

Go to:

`General Parameters > Advanced Parameters > Security > SSL options for server requests`

6.2 SSO as a service (SSOaaS)

6.2.1 Our concept of SSOaaS

Access management provides 3 services:

- Global Authentication: Single Sign-On
- Authorization: to grant authentication is not enough. User rights must be checked
- Accounting: SSO logs (access) + application logs *(transactions and results)*

LL::NG affords all these services (except application logs of course, but headers are provided to permit this).

Headers setting is an another LL::NG service. LL::NG can provide any user attributes to an application (see *Rules and headers*)

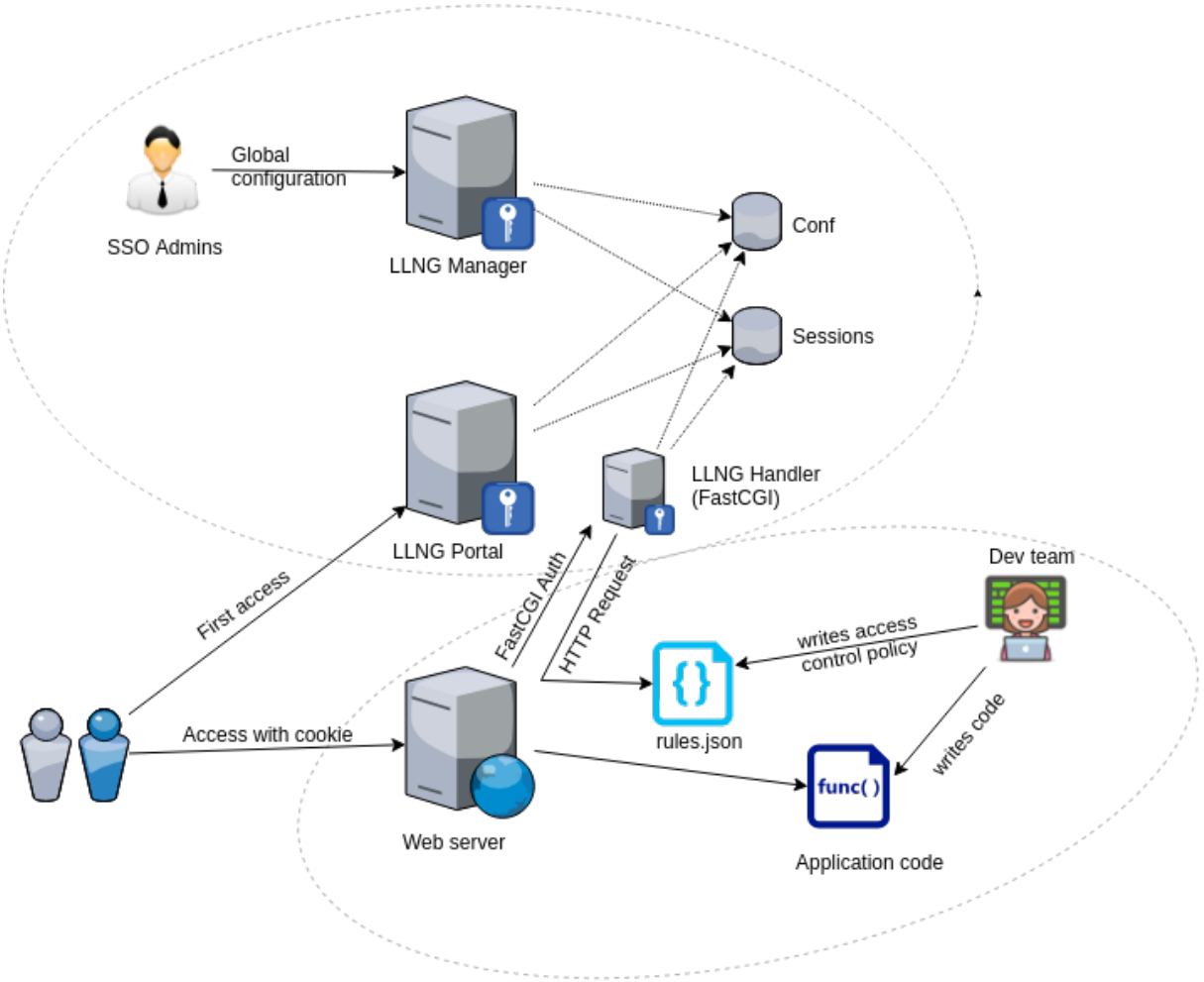
`*aaS` means that application can drive underlying layer (IaaS for infrastructure, PaaS for platform,...). So for us, `SSOaaS` must provide the ability for an app to manage authorizations and choose user attributes to set. Authentication can not be really `*aaS`: app must just use it, not manage it.

LL::NG affords some features that can be used to provide SSO as a service: a web application can manage its rules and headers. Docker or VM images (Nginx only) includes LL::NG Nginx configuration that aims to a global *LL::NG authorization server*. By default, all authenticated users can access and one header is set: `Auth-User`. If application gives a RULES_URL parameter that refers to a JSON file, authorization server will read it, apply specified rules and set required headers (see *DevOps Handler*).

There are two different architectures to do this:

- Using a *global FastCGI (or uWSGI) server*
- Using front reverse-proxies *(some cloud installations use reverse-proxies in front-end)*

Example of a global FastCGI architecture:



In both case, Handler type must be set to *DevOps*.

6.2.2 Examples of webserver configuration for Docker/VM images

Using a global FastCGI (or uWSGI) server

Nginx

In this example, web server templates (Nginx only) are configured to request authorization from a central FastCGI server:

```
server {
  server_name myapp.domain.com;
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    # Pass authorization requests to Central FastCGI server:
    fastcgi_pass 10.1.2.3:9090;
```

(continues on next page)

(continued from previous page)

```
    fastcgi_param VHOSTTYPE DevOps;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /lmauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;

    # Set dynamically rules (LLNG will poll it every 10 mn)
    fastcgi_param RULES_URL http://rulesserver/my.json
  }
  location /rules.json {
    auth_request off;
    allow 10.1.2.3;
    deny all;
  }
  location ~ ^(.*\.php)$ {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    include /etc/lemonldap-ng/nginx-lua-headers.conf;
    # ...
    # Example with php-fpm:
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/var/run/php/php7.0-fpm.sock;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Apache

There is an experimental FactCGI client in LLNG. You just have to install FCGI::Client and add this in the apache2.conf or your web applications or proxies.

The following configuration example assumes that you are in a "central FastCGI" configuration.

```
<VirtualHost ...>
    ServerName app.tls
    PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2::FCGIClient

    # This must point to the central FastCGI server
    PerlSetVar LLNG_SERVER 192.0.2.1:9090

    # Declare this vhost as a DevOps vhost, so that we do not have
    # to declare it in the LemonLDAP::NG Manager
    PerlSetVar VHOSTTYPE DevOps
```

(continues on next page)

(continued from previous page)

```
    # This URL will be fetched by the central FastCGI server and
    # used to make the authentication decision about this virtualhost
    # Make sure the central FastCGI server can reach it
    PerlSetVar RULES_URL http://app.tld/rules.json
    ...
</VirtualHost>
```

Node.js

Using express and fastcgi-authz-client, you can protect also an Express server. Example:

```
var express = require('express');
var app = express();
var FcgiAuthz = require('fastcgi-authz-client');
var handler = FcgiAuthz({
  host: '127.0.0.1',
  port: 9090,
  PARAMS: {
    RULES_URL: 'http://my-server/rules.json'
  }
});

app.use(handler);

// Simple express application
app.get('/', function(req, res) {
  return res.send('Hello ' + req.upstreamHeaders['auth-user'] + ' !');
});

// Launch server
app.listen(3000, function() {
  return console.log('Example app listening on port 3000!');
});
```

Plack application

You just have to enable Plack::Middleware::Auth::FCGI. Simple example:

```
use Plack::Builder;

my $app   = sub {
  my $env = shift;
  my $user = $env->{fcgiauth-auth-user};
  return [ 200, [ 'Content-Type' => 'text/plain' ], [ "Hello $user" ] ];
};

# Optionally ($fcgiResponse is the PSGI response of remote FCGI auth server)
#sub on_reject {
```

(continues on next page)

(continued from previous page)

```
#    my($self,$env,$fcgiResponse) = @_;
#    my $statusCode = $fcgiResponse->{status};
#    ...
#}

builder
{
  enable "Auth::FCGI",
    host => '127.0.0.1',
    port => '9090',
    fcgi_auth_params => {
      RULES_URL => 'https://my-server/my.json',
    },
    # Optional rejection subroutine
    #on_reject => \&on_reject;
    ;
  $app;
};
```

Using front reverse-proxies

This is a simple Nginx configuration file. It looks like a standard LL::NG nginx configuration file except for:

- VHOSTTYPE parameter forced to use DevOps handler
- /rules.json must not be protected by LL::NG but by the web server itself

This configuration handles `*.dev.sso.my.domain` URL and forwards authenticated requests to `<vhost>.internal.domain`. Rules can be defined in `/rules.json` which is located at the website root directory.

```
server {
  server_name "~^(?<vhost>.+?)\.dev\.sso\.my\.domain$";
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Force handler type:
    fastcgi_param VHOSTTYPE DevOps;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /lmauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }
  location /rules.json {
    auth_request off;
    allow 127.0.0.0/8;
    deny all;
  }
  location / {
```

(continues on next page)

(continued from previous page)

```
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    include /etc/lemonldap-ng/nginx-lua-headers.conf;
    proxy_pass https://$vhost.internal.domain;
  }
}
```

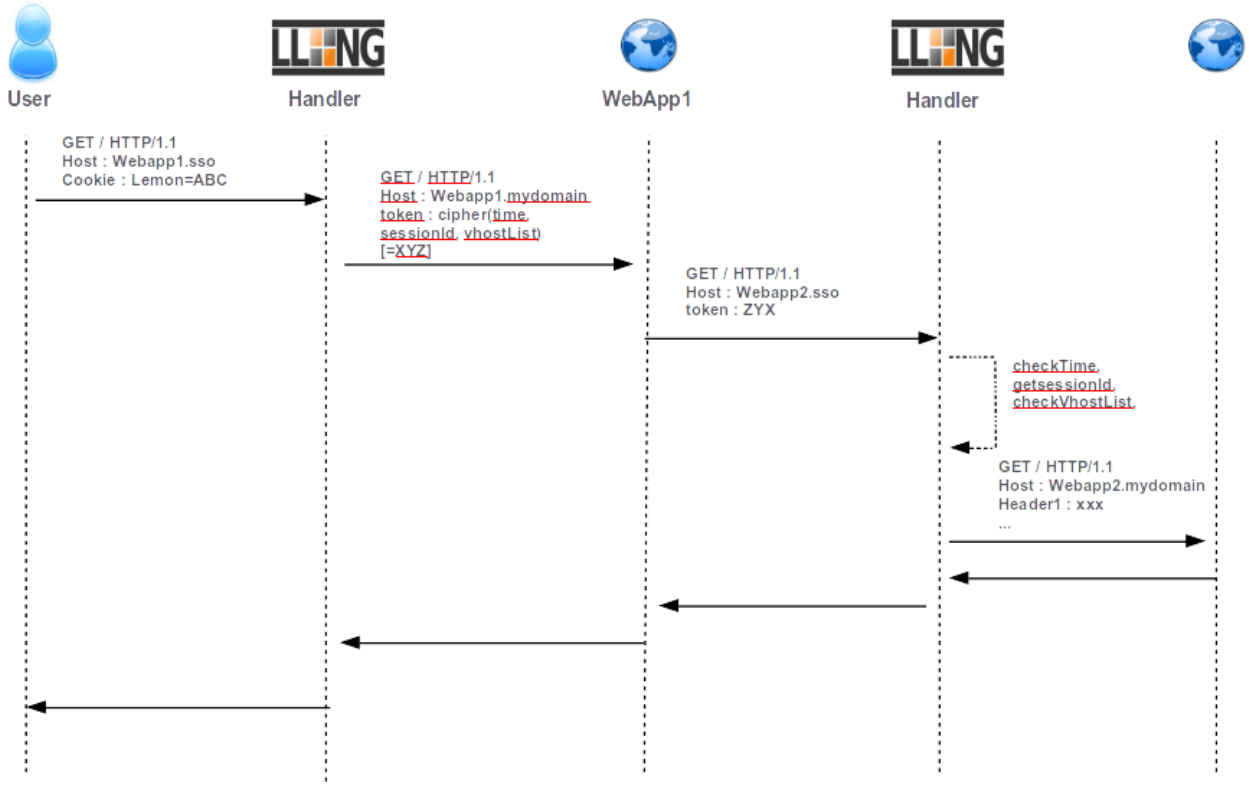
6.3 Handling server webservice calls

In modern applications, web application may need to request some other web applications on behalf of the authenticated users. There are three ways to do this:

- the Ugly : provide to all applications SSO cookie. Not secured because SSO cookie can be caught and used everywhere, every time by everyone!!! **NOT RECOMMENDED**.
- the Bad (*Secure Token Handler*) : **Deprecated**. Can be used in specific cases
- the Good (Service Token Handler): See below ! (Thanks Sergio. . .)

The "Bad" method consists to give the token (cookie value) to WebApp1 which uses it as cookie header in its request. Since 2.0 version, LL::NG gives a better way (the Good !) to do this by using limited scope tokens.

Tokens are time limited (30 seconds by default) and URL restricted.



6.3.1 Webapp1 handler configuration

Select **Main** handler type to protect WebApp1 and insert a header named **X-Llng-Token** filled with this value:

```
token( $_session_id, 'webapp2.example.com', 'webapp3.example.com',
↪'serviceHeader1=webapp1.example.com', "testHeader=$uid" )
```

WebApp1 can read this header and use it in its requests by setting the `X-Llng-Token` header. The token is built by using the session ID and authorized virtualhosts list. By default, the Service Token is only available during 30 seconds and for specified virtualhosts. The token can be use to send service headers to webapp2 like origin host by example.

You can set ServiceToken TTL in the virtualHost options in Manager for each required virtualHost.

You can also set ServiceToken default timeout (30 seconds) by editing `lemonldap-ng.ini` in section [handler]:

```
[handler]
handlerServiceTokenTTL = 30
```

Note: Service token timeout can be set for each virtual hosts.

6.3.2 Webapp2 handler configuration

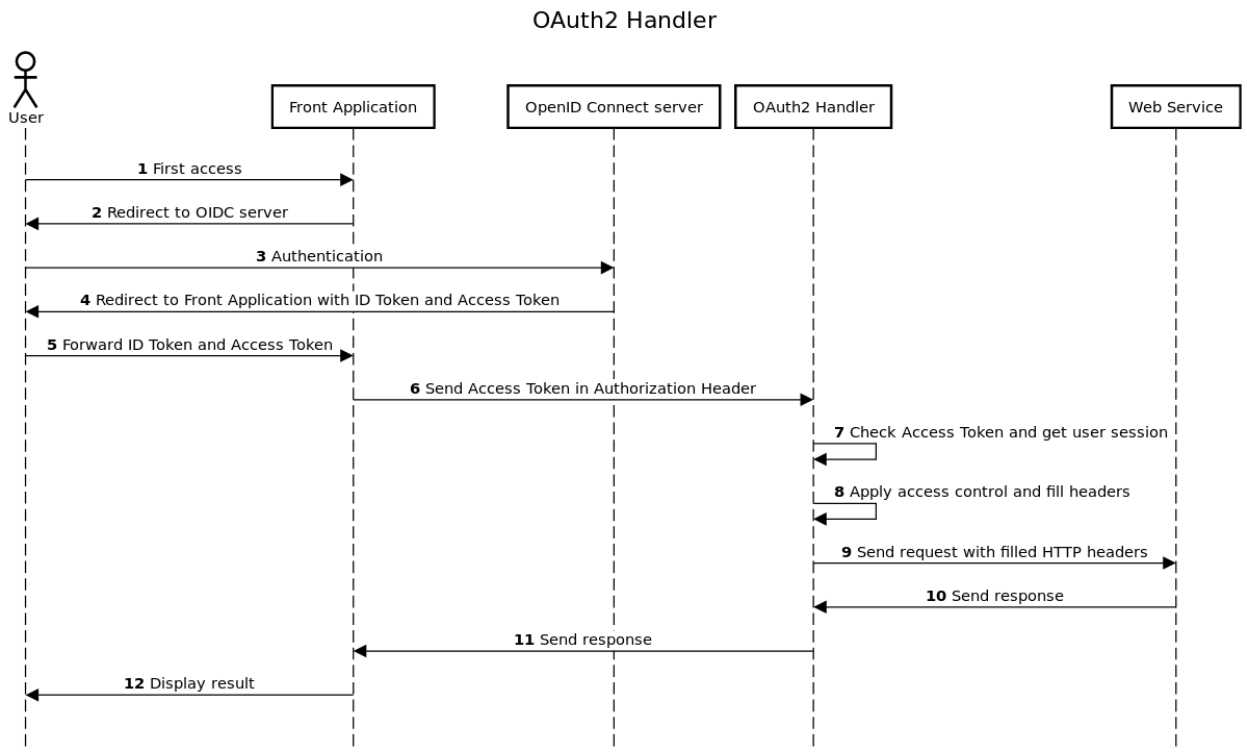
Change handler type to **ServiceToken**. So it is able to manage both user and server connections. And that's all !

6.4 OAuth2 Handler

6.4.1 Presentation

This Handler is able to check an OAuth2 access token to retrieve the user real session and protect a virtual host like a standard Handler (access control and HTTP headers transmission).

This requires to get an OAuth2 access token through LL::NG Portal (OpenID Connect server). This access token can then be used in the `Authorization` header to authenticate to the Web Service / API protected by the OAuth2 Handler.



Tip: In the above schema, the OpenID Connect process is simplified. How the front application receives the Access Token depends on the requested flow (Authorization code, Implicit or Hybrid). In all cases, the application will have an Access Token and will be able to use it to request a Web Service.

Example:

```
curl -H "Authorization: Bearer de853461341e88e9def8fcb9db2a81c4" https://oauth2.example.
↪com/api/test | json_pp
```

```
{
    check: true,
    user: "dwho"
}
```

6.4.2 Additional variables

The OAuth2 handler defines a few extra variables that you can use in *rules and headers*.

- `$_clientId`: client ID of the application which requested the Access Token
- `$_clientConfKey`: configuration key of the application which requested the Access Token
- `$_scope`: list of space-separated scopes granted by the Access Token

For example, to grant access to access tokens containing the `write` scope, use

```
$_scope =~ /(?<!\S)write(?!\S)/
```

6.4.3 Configuration

Protect you virtual host like any other virtual host with the standard Handler.

Define access rules and headers. Then in `Options` > `Type`, choose `OAuth2`.

6.4.4 Reference

RFC6750

6.5 Secure Token Handler

6.5.1 Presentation

The Secure Token Handler is a special Handler that creates a token for each request and send it to the protected application. The real user identifier is stored in a Memcached server and the protected application can request the Memcached server to get user identifier.

This mechanism allows one to protect an application with an unsafe link between Handler and the application, but with a safe link between the Memcached server and the application.

6.5.2 Configuration

Install Cache::Memcached dependency.

Virtual host

You just have to set "Type: SecureToken" in the VirtualHost options in the manager.

If you want to protect only a virtualHost part, keep type on "Main" and set type in your configuration file:

- Apache: use simply a `PerlSetVar VHOSTTYPE AuthBasic`
- Nginx: create another FastCGI with a `fastcgi_param VHOSTTYPE SecureToken;`

Note: This handler uses Apache2Filter Module to hide token, prefer *Handling server webservice calls* for other servers.

Handler parameters

SecureToken parameters are the following:

- **Memcached servers**: addresses of Memcached servers, separated with spaces.
- **Token expiration**: time in seconds for token expiration (remove from Memcached server).
- **Attribute to store**: the session key that will be stored in Memcached.
- **Protected URLs**: Regexp of URLs for which the secure token will be sent, separated by spaces
- **Header name**: name of the HTTP header carrying by the secure token.

- **Allow requests in error**: allow a request that has generated an error in token generation to be forwarded to the protected application without secure token (default: yes)

Attention: Due to Handler API change in 1.9, you need to set these attributes in `lemonldap-ng.ini` and not in Manager, for example:

```
[handler]
secureTokenMemcachedServers = 127.0.0.1:11211
secureTokenExpiration = 60
secureTokenAttribute = uid
secureTokenUrls = .*
secureTokenHeader = Auth-Token
secureTokenAllowOnError = 1
```

6.6 DevOps Handler

This handler is designed to read vhost configuration from the website itself not from LL:NG configuration. Rules and headers are set in a **rules.json** file stored at the website root directory (ie `http://website/rules.json`). This file looks like:

```
{
  "rules": {
    "^/admin": "$uid eq 'admin'",
    "default": "accept"
  },
  "headers": {
    "Auth-User": "$uid"
  }
}
```

If this file is not found, the default rule "accept" is applied and just "Auth-User" header is sent (Auth-User => $uid).

No specific configuration is required except that:

- you have to choose this specific handler (directly by using `VHOSTTYPE` environment variable)
- you can set the loopback URL needed by the DevOps handler to get `/rules.json` or use `RULES_URL` parameter to set JSON file path (see *SSO as a Service*). Default to `http://127.0.0.1:<server-port>`

Attention: Note that DevOps handler will refuse to compile rules.json if *Safe Jail* isn't enabled.

See *SSO as a Service* for more

6.7 DevOps+ServiceToken Handler

This handler enables both:

- *DevOps Handler*, base of *SSO as a service (SSOaaS)*
- *Service token handler*, used to control web-api sub requests

CHAPTER

SEVEN

LEMONLDAP::NG DATABASES

7.1 Configuration database

7.1.1 How to change configuration backend

LemonLDAP::NG provides a script to change configuration backend easily keeping history. It is set in LemonLDAP::NG utilities directory (`convertConfig`).

How it works

The `convertConfig` utility reads 2 LL::NG configuration files (`lemonldap-ng.ini`):

- **Current**: to extract all configuration history
- **New**: to write all configuration history

Let's go

- Prepare your new lemonldap-ng.ini file
- Configure your new backend (create SQL database,...)
- Launch the following command:

```
convertConfig --current=/etc/lemonldap-ng/lemonldap-ng.ini --new=/new/lemonldap-ng.ini
```

- Install the new lemonldap-ng.ini file at the place of the old file in all LL::NG servers
- Restart all your Apache servers

Note: Since LemonLDAP 2.0.9, you don't need the `--current` and `--new` options when migrating from the default file-based backend. Simply run *convertConfig* to migrate from the default configuration backend to the currently configured backend.

See also

Documentation is available for configuration backends :

- *SQL*
- *File*
- *LDAP*
- *SOAP proxy mechanism*

7.1.2 File configuration backend

This is the default configuration backend. Configuration is stored as JSON.

Tip: This configuration storage can be shared between different hosts using:

- *SOAP configuration backend proxy*
- any files sharing system (NFS, NAS, SAN,...)

Configuration

You just have to configure a directory writable by Apache user and set it in [configuration] section in your lemonldap-ng.ini file:

```
[configuration]
type  = File
dirName = /var/lib/lemonldap-ng/conf
prettyPrint = 1
```

Parameters

- *dirName*: directory under which the configuration files will be stored. It must be writable by your webserver account
- *prettyPrint*: store files in a readable. Set it to 0 to get a small performance increase when loading/saving configuration

7.1.3 YAMLFile configuration backend

Same as *File configuration backend* except that configuration is stored in YAML format.

Configuration

You just have to configure a directory writable by Apache user and set it in [configuration] section in your lemonldap-ng.ini file:

```
[configuration]
type  = YAMLFile
dirName = /var/lib/lemonldap-ng/conf
```

7.1.4 SQL configuration backends

There is 2 types of SQL configuration backends for LemonLDAP::NG:

- **CDBI**: very simple storage (recommended)
- **RDBI**: triple store storage

Tip: You can use any database engine if it provides a Perl Driver. You will find here examples for MySQL and PostgreSQL, but other engines may also work.

See *how to change configuration backend*.

MySQL

Perl Driver

You need DBD::MySQL Perl module:

- Debian:

```
apt install libdbd-mysql-perl
```

- Red Hat:

```
yum install perl-DBD-MySQL
```

Database and table creation

Create database:

```
CREATE DATABASE lemonldap-ng CHARACTER SET utf8;
```

Use database to create table:

```
use lemonldap-ng
```

RDBI

```
CREATE TABLE lmConfig (
    cfgNum int(11) NOT NULL,
    field varchar(255) NOT NULL DEFAULT '',
    value longtext,
    PRIMARY KEY (cfgNum,field)
    );
```

CDBI

```
CREATE TABLE lmConfig (
    cfgNum int not null primary key,
    data longtext
);
```

Grant access

You have to grant read/write access for the manager component. Other components needs just a read access. You can also use the same user for all.

Tip: You can use different dbiUser strings:

- one with read/write rights for servers hosting the manager
- one with just read rights for other servers

For example (suppose that our servers are in 10.0.0.0/24 network):

```
GRANT SELECT,INSERT,UPDATE,DELETE,LOCK TABLES ON lemonldap-ng.lmConfig
  TO lemonldaprw@manager.host IDENTIFIED BY 'mypassword';
GRANT SELECT ON lemonldap-ng.lmConfig
  TO lemonldapro@'10.0.0.%' IDENTIFIED BY 'myotherpassword';
```

Connection settings

Change configuration settings in `/etc/lemonldap-ng/lemonldap-ng.ini` file (section configuration):

```
[configuration]
type = RDBI
dbiChain    = DBI:mysql:database=lemonldap-ng;host=1.2.3.4
dbiUser     = lemonldaprw
dbiPassword = mypassword
; optional
dbiTable    = mytablename
```

PostGreSQL

Perl Driver

You need DBD::Pg Perl module:

- Debian:

```
apt install libdbd-pg-perl
```

- Red Hat:

```
yum install perl-DBD-Pg
```

Database and table creation

Create database:

```
CREATE DATABASE lemonldap-ng;
```

Use database to create table:

```
use lemonldap-ng
```

RDBI

```
CREATE TABLE lmconfig (
    cfgnum integer NOT NULL,
    field text NOT NULL,
    value text,
    PRIMARY KEY (cfgNum,field)
    );
```

CDBI

```
CREATE TABLE lmConfig (
    cfgnum integer not null primary key,
    data text
);
```

Connection settings

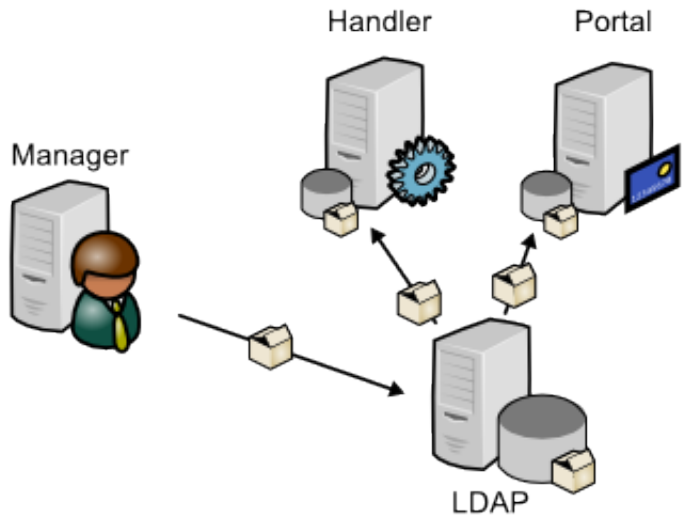
Change configuration settings in `/etc/lemonldap-ng/lemonldap-ng.ini` file (section configuration):

```
[configuration]
type = RDBI
dbiChain    = DBI:Pg:database=lemonldap-ng;host=1.2.3.4
dbiUser     = lemonldaprw
dbiPassword = mypassword
; optional
dbiTable    = mytablename
```

7.1.5 LDAP configuration backend

Presentation

You can choose to store LemonLDAP::NG configuration in an LDAP directory.



Advantages:

- Easy to share between servers with remote LDAP access
- Easy to duplicate with LDAP synchronization services (like SyncRepl in OpenLDAP)
- Security with SSL/TLS
- Access control possible by creating one user for Manager (write) and another for portal and handlers (read)
- Easy import/export through LDIF files

The configuration will be store under a specific branch, for example `ou=conf,ou=applications,dc=example,dc=com`.

Each configuration will be represented as an entry, which structural objectClass is by default `applicationProcess`. The configuration name is the same that files, so lmConf-1, lmConf-2, etc. This name is used in entry DN, for example `cn=lmConf-1,ou=conf,ou=applications,dc=example,dc=com`.

Then each parameter is one value of the attribute `description`, prefixed by its key. For example `{ldapPort}389`.

The LDIF view of such entry can be:

```
dn: cn=lmConf-1,ou=conf,ou=applications,dc=example,dc=com
objectClass: top
objectClass: applicationProcess
cn: lmConf-1
description: {globalStorage}'Apache::Session::File'
description: {cookieName}'lemonldap'
description: {whatToTrace}'$uid'
...
```

Configuration

LDAP server

Configuration objects use standard object class: `applicationProcess`. This objectClass allow attributes `cn` and `description`. If your LDAP server do not manage this objectClass, configure other objectclass and attributes (see below).

We advice to create a specific LDAP account with write access on configuration branch.

Next create the configuration branch where you want. Just remember its DN for LemonLDAP::NG configuration.

LemonLDAP::NG

Configure LDAP configuration backend in `lemonldap-ng.ini`, section `[configuration]`:

```
type = LDAP
ldapServer = ldap://localhost
ldapConfBase = ou=conf,ou=applications,dc=example,dc=com
ldapBindDN = cn=manager,dc=example,dc=com
ldapBindPassword = secret
ldapObjectClass = applicationProcess
ldapAttributeId = cn
ldapAttributeContent = description
```

Parameters:

- **ldapServer**: LDAP URI of the server
- **ldapConfBase**: DN of configuration branch
- **ldapBindDN**: DN used to bind LDAP
- **ldapBindPassword**: password used to bind LDAP
- **ldapObjectClass**: structural objectclass of configuration entry (optional)
- **ldapAttributeId**: RDN attribute of configuration entry (optional)
- **ldapAttributeContent**: attribute used to store configuration values, must be multivalued (optional)
- **ldapVerify**: When using a LDAPS or TLS server, whether or not to validate the server certificate. Possible values: `require`, `optional` or `none`.
- **ldapCAFile**: This allows you to override the default system-wide certificate authorities by giving a single file containing the CA used by the LDAP server.

- **ldapCAPath**: This allows you to override the default system-wide certificate authorities by giving the path of a directory containing your trusted certificates.

7.1.6 MongoDB configuration backends

MongoDB is a NoSQL database that can be used both for storing configuration and *sessions*. You need to install Perl MongoDB module to be able to use this backend.

See *how to change configuration backend* to change your configuration database.

Configuration

To use a MongoDB backend, configure your `lemonldap-ng.ini` file (section configuration) :

- Choose MongoDB as type
- Set dbName and collectionName parameters if different than default values (llConfDB and configuration)
- Set host and if needed db_name username, password and ssl fields as follow.

Example :

```
[configuration]
type = MongoDB
dbName = llConfDB
collectionName = configuration
; using a single server
host = 127.0.0.1:27017
; using a replicaSet
; host = mongodb://mongo1.example.com,mongo2.example.com/?replicaSet=myset
ssl = 1
; authentication parameters
db_name = admin
user = lluser
password = llpassword
```

Optional parameters (see MongoDB::MongoClient man page)		
Name	Comment	Example
db_name	Admin database (default: admin)	admin
auth_mechanism	Authentication mechanism	PLAIN
auth_mechanism_properties		
connect_timeout	Connection timeout	10000
ssl	Boolean or hash ref (default: 0)	1
username	Username to use to connect	lluser
password	Password	llpassword

7.1.7 Mini MongoDB howto

Just some commands needed to create collection and user:

```
$ mongo
connecting to: test
> use configuration
switched to db configuration
> db.createCollection("configuration")
...
> db.createUser({user:"lluser",pwd:"llpassword",roles:["readWrite"]})
...
> exit
bye
$
```

7.1.8 SOAP configuration backend (deprecated)

You can share your configuration over the network using SOAP proxy system.

Tip: Note that SOAP is not a real configuration backend, but just a proxy system to access to your configuration over the network

Attention: SOAP has been deprecated. Prefer to use *REST configuration backend*

Configuration

First, configure your real backend

- On your main server, configure a *File*, *SQL* or *LDAP* backend
- Set SOAP parameter to true in the configuration using the manager: the portal will become a SOAP server
- Configure your web server to allow remote access. Remote SOAP access is disabled by default. You must change it as follow :
- in `portal-apache2.conf` :

```
# SOAP functions for configuration access (disabled by default)
<Location /index.fcgi/config>
    Require ip 192.168.2.0/24
</Location>
```

- in `portal-nginx.conf` :

```
# SOAP functions for configuration access (disabled by default)
location /index.psgi/config {
  allow 192.168.2.0/24;
}
```

Next, configure SOAP for your remote servers

Change configuration in lemonldap-ng.ini :

```
type         = SOAP
; Apache
proxy        = https://auth.example.com/index.fcgi/config
; Nginx
proxy        = https://auth.example.com/index.pcgi/config
```

You can also add some other parameters

```
User         = lemonldap
Password     = mypassword
# LWP::UserAgent parameters
proxyOptions = { timeout => 5 }
```

7.1.9 REST configuration backend

You can share your configuration over the network using REST proxy system:

- GET /config/latest: get the last config metadata
- GET /config/<cfgNum>: get the metadata for config n° <cfgNum>
- GET /config/<latest|cfgNum>/<key>: get conf key value
- GET /config/<latest|cfgNum>?full=1: get the full configuration

You can retrieve "human readable" error messages:

- GET /error/<lang>/<errNum>: get <errNum> error reference and <lang> errors file.

If no <lang> provided, 'en' errors file is returned.

Tip: Note that REST is not a real configuration backend, but just a proxy system to access to your configuration over the network

---

Configuration

First, configure your real backend

- On your main server, configure a *File*, *SQL* or *LDAP* backend
- Enable REST server in the configuration using the manager (in portal plugins)
- Configure your web server to allow remote access. Remote REST access is disabled by default. Change it as follow:

* In `portal-apache2.conf`:

```
# REST functions for configuration access (disabled by default)
<Location /index.fcgi/config>
    Require ip 192.168.2.0/24
</Location>
```

* In `portal-nginx.conf`:

```
# REST functions for configuration access (disabled by default)
location /index.psgi/config {
  allow 192.168.2.0/24;
}
```

Next, configure REST for your remote servers

Change configuration in lemonldap-ng.ini :

```
type         = REST
; Apache
baseUrl      = https://auth.example.com/index.fcgi/config
; Nginx
baseUrl      = https://auth.example.com/index.psgi/config
```

You can also add some other parameters

```
User         = lemonldap
Password     = mypassword
# LWP::UserAgent parameters
proxyOptions = { timeout => 5 }
```

7.1.10 Local configuration backend

Some admins wants to deploy configuration using lemonldap-ng.ini only. This backend just return an empty configuration.

Attention: Advanced use only !

7.2 Sessions database

7.2.1 How to change session backend

LemonLDAP::NG provides a script to change session backend. This script will help you transfer existing persistent sessions (or offline sessions) when migrating from one backend to another, or when adding indexes to a *browseable session backend*. It is available in LemonLDAP::NG utilities directory (`convertSessions`).

How it works

The `convertSessions` utility requires you to create a job configuration file with the following content:

```
# This example migrates psessions from the default File backend to a PostgreSQL database
[sessions_from]
storageModule = Apache::Session::File
storageModuleOptions = {    \
      'Directory' => '/var/lib/lemonldap-ng/psessions',     \
      'LockDirectory' => '/var/lib/lemonldap-ng/psessions/lock', \
}
# Only convert some session types
# sessionKind = Persistent, SSO

[sessions_to]
storageModule = Apache::Session::Browseable::Postgres
storageModuleOptions = {    \
    'DataSource' => 'DBI:Pg:database=lemonldapdb;host=pg.example.com', \
    'UserName' => 'lemonldaplogin', \
    'Password' => 'lemonldappw', \
    'Commit' => 1, \
    'Index' => 'ipAddr _whatToTrace user', \
    'TableName' => 'psessions', \
}
```

Invocation

```
convertSessions -c job.ini
```

Options:

- `-c`: job configuration file (mandatory)
- `-r oldkey=newkey`: rename session keys during conversion (optional, can be given multiple times)
- `-i`: ignore errors. By default errors will stop the script execution
- `-d`: print debugging output

7.2.2 File session backend

File session backend is the more simple session database. Sessions are stored as files in a single directory. Lock files are stored in another directory. It can not be used to share sessions between different servers except if you share directories (with NFS,...) or if you use *SOAP proxy*.

Setup

In the manager: set "Apache::Session::File" in "General parameters » Sessions » Session storage » Apache::Session module" and add the following parameters (case sensitive):

Required parameters		
Name	Comment	Example
Directory	The path to the main directory	/var/lib/lemonldap-ng/sessions
LockDirectory	The path to the lock directory	/var/lib/lemonldap-ng/sessions/lock

Security

Restrict access to the directories only to the Apache server. Example:

```
chmod 750 /var/lib/lemonldap-ng/sessions /var/lib/lemonldap-ng/sessions/lock
chown www-data:www-data /var/lib/lemonldap-ng/sessions /var/lib/lemonldap-ng/sessions/
↪lock
```

7.2.3 SQL session backend

SQL session backend can be used with many SQL databases such as:

- MariaDB / MySQL
- PostgreSQL
- Oracle
- Informix
- Sybase
- …

Setup

Prepare the database

Your database must have a specific table to host sessions. Here are some examples for main databases servers.

Attention: If your database doesn't accept UTF-8 characters in 'text' field, use 'blob' instead of 'text'.

MySQL

Create a database if necessary:

```
mysqladmin create lemonldap-ng
```

Create sessions table:

```
CREATE TABLE sessions (
    id char(32) not null primary key,
    a_session text
    );
```

Attention: Change `char(32)` by `varchar(64)` if you use the now recommended SHA256 hash algorithm. See *Sessions* for more details

Tip: You can change table name `sessions` to whatever you want, just adapt the parameter `TableName` in module options.

Attention: For a better UTF-8 support, use DBD::MariaDB with Apache::Session*::MySQL instead of DBD::mysql

PostgreSQL

Create user and role:

```
su - postgres
createuser lemonldap-ng -P
```

```
Entrez le mot de passe pour le nouveau rôle : <PASSWORD>
Entrez-le de nouveau : <PASSWORD>
Le nouveau rôle est-il un super-utilisateur ? (o/n) n
Le nouveau rôle doit-il être autorisé à créer des bases de données ? (o/n) n
Le nouveau rôle doit-il être autorisé à créer de nouveaux rôles ? (o/n) n
```

Create database:

```
createdb -O lemonldap-ng lemonldap-ng
```

Create table:

```
psql -h 127.0.0.1 -U lemonldap-ng -W lemonldap-ng
```

```
Mot de passe pour l'utilisateur lemonldap-ng :
[...]
lemonldap-ng=> create unlogged table sessions ( id char(32) not null primary key, a_
↪session text );
lemonldap-ng=> q
```

Attention: Change `char(32)` by `varchar(64)` if you use the now recommended SHA256 hash algorithm. See *Sessions* for more details

Manager

Go in the Manager and set the session module (for example Apache::Session::Postgres for PostgreSQL) in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the following parameters (case sensitive):

Required parameters		
Name	Comment	Example
DataSource	The DBI string	dbi:Pg:dbname=sessions;host=10.2.3.1
UserName	The database username	lemonldap-ng
Password	The database password	mysuperpassword
Commit	Required for PostgreSQL	1
TableName	*(Optional)* Name of the table	sessions

You must read the man page corresponding to your database (Apache::Session::MySQL, ...) to learn more about parameters. You must also install the database connector (https://metacpan.org/pod/DBD::Oracle, DBD::Pg,...)

Attention: For MySQL, you need to set additional parameters:

- LockDataSource
- LockUserName
- LockPassword

Tip: For better performances, you can use specific *browseable session backend*.

Learn more at *how to increase Data Base performances*.

UTF8 support

If you may store some non-ASCII characters, you must add the parameter corresponding to your database.

Database	Parameter name	Value
MySQL	mysql_enable_utf8	1
PostgreSQL	pg_enable_utf8	1
SQLite	sqlite_unicode	1

Security

Restrict network access to the database.

You can also use different user/password for your servers by overriding parameters `globalStorage` and `globalStorageOptions` in lemonldap-ng.ini file.

7.2.4 LDAP session backend

An Apache session module was created by LL::NG team to store sessions in an LDAP directory.

Attention: This module is not part of LL::NG distribution, and can be found on CPAN: Apache::Session::LDAP.

Tip: This module is also available on GitHub.

Sessions will be stored as LDAP entries, like this:

```
dn: cn=6fb7c4a170a04668771f03b0a4747f46,ou=sessions,dc=example,dc=com
objectClass: applicationProcess
cn: 6fb7c4a170a04668771f03b0a4747f46
description: [Base64 serialized data]
```

Setup

Go in the Manager and set the LDAP session module (Apache::Session::LDAP) in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the following parameters (case sensitive):

Required parameters		
Name	Comment	Example
ldapServer	URI of the server	ldap://localhost
ldapConfBase	DN of sessions branch	ou=sessions,dc=example,dc=com
ldapBindDN	Connection login	cn=admin,dc=example,dc=password
ldapBindPassword	Connection password	secret

Optional parameters		
Name	Comment	Default value
ldapObjectClass	Objectclass of the entry	applicationProcess
ldapAttributeId	Attribute storing session ID	cn
ldapAttributeContent	Attribute storing session content	description
ldapVerify	Perform certificate validation	require (use none to disable)
ldapCAFile	Path of CA file bundle	(system CA bundle)
ldapCAPath	Perform CA directory	(system CA bundle)

Security

Restrict network access to the LDAP directory, and add specific ACL to session branch.

You can also use different user/password for your servers by overriding parameters `globalStorage` and `globalStorageOptions` in lemonldap-ng.ini file.

7.2.5 Redis session backend

Apache::Session::Browseable::Redis is the faster shareable session backend

Setup

Install and launch a Redis server. Install Apache::Session::Browseable::Redis Perl module.

In the manager: set Apache::Session::Browseable::Redis in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the connection parameters for your Redis server(s).

This backend uses the perl bindings for Redis database provided by the Redis perl module. A complete list of supported constructor/connection options can be found in the module documentation.

E.g., Parameters (case sensitive):

Name	Comment	Example
server	Redis server @ IP:PORT	127.0.0.1:6379
sock	Redis server @ unix socket	unix:/path/to/redis.sock
sentinels	Redis sentinels list	127.0.0.1:26379,127.0.0.2:26379,127.0.0.3:26379
password	password (== requirepass)	ChangeMe
select	Redis DB	1
Index	Fields to index	refer to *List of fields to index by session type*

Security

Restrict network access to the redis server. For remote servers, you can use *SOAP session backend* in cunjunction to increase security for remote server that access through an unsecure network

7.2.6 MongoDB session backend

Apache::Session::MongoDB is a faster shareable session backend.

> **Attention:** Use an up-to-date version of Apache::Session::MongoDB, at least 1.8.1.

Setup

Install and launch a MongoDB server. Install Apache::Session::MongoDB Perl module (version 0.15 required). You also need a recent version of Perl MongoDB client (version 1.00 required).

In the manager: set Apache::Session::MongoDB in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the following parameters (case sensitive):

Optional parameters		
Name	Comment	Example
host	MongoDB server URI	127.0.0.1:27017
db_name	Session database (default: sessions)	llconfdb
collection	Collection (default: sessions)	sessions
auth_mechanism	Authentication mechanism	PLAIN
auth_mechanism_properties		
connect_timeout	Connection timeout	10000
ssl	Boolean or hash ref (default: 0)	1
username	Username to use to connect	lluser
password	Password	llpassword

Advanced connection parameters (Replica Sets, timeouts…) may be specified in the `host` parameter. Refer to the perl MongoDB documentation for details

Security

Restrict network access to the MongoDB server. For remote servers, you can use *SOAP session backend* in cunjunction to increase security for remote server that access through an unsecure network

7.2.7 Browseable session backend

Presentation

Browseable session backend (Apache::Session::Browseable) works exactly like Apache::Session::* corresponding module but add index that increase the speed of some operations. It is recommended in production deployments.

Note: Without index, LL::NG will have to retrieve all sessions stored in backend and deserialize then filter each of them.

The following table list fields to index for each session type:

List of fields to index by session type

Session Type	Fields to index
Sessions (global)	_whatToTrace _session_kind _utime ipAddr _httpSessionType user
Persistent sessions	_session_kind _httpSessionType _session_uid ipAddr _whatToTrace
CAS sessions	_cas_id pgtIou
SAML sessions	_session_kind _utime _saml_id ProxyID _nameID _assert_id _art_id
OpenID Connect sessions	_session_kind _utime

Note: If you have configured LemonLDAP::NG to use something other than *_whatToTrace* as the main session identifier, you must replace *_whatToTrace* with the new session field in the previous list

See Apache::Session::Browseable man page to see how use indexes.

Tip: It is advised to use separate session backends for standard sessions, SAML sessions and CAS sessions, in order to avoid unused indexes.

Available backends

PgJSON session backend

This backend is the recommended one for production installations of LemonLDAP::NG.

Prerequisites

First, make sure you have installed the `DBD::Pg` perl module.

On Debian-based distributions

```
apt install libdbd-pg-perl
```

On Fedora-based distributions

```
yum install 'perl(DBD::Pg)'
```

The minimum required version of PostgreSQL is 9.3 with support for JSON column types

Make sure you are using at least version 1.2.9 of `Apache::Session::Browseable`, this might require installing it from Debian Backports or CPAN.

Create database schema

Create the following tables. You may skip the session types you are not going to use, but you need at least `sessions` and `psessions`

```
CREATE TABLE sessions (
    id varchar(64) not null primary key,
    a_session jsonb
);

CREATE INDEX i_s__whatToTrace     ON sessions ((a_session ->> '_whatToTrace'));
CREATE INDEX i_s__session_kind    ON sessions ((a_session ->> '_session_kind'));
CREATE INDEX i_s__utime           ON sessions ((cast (a_session ->> '_utime' as␣
↪bigint)));
CREATE INDEX i_s_ipAddr           ON sessions ((a_session ->> 'ipAddr'));
CREATE INDEX i_s__httpSessionType ON sessions ((a_session ->> '_httpSessionType'));
CREATE INDEX i_s_user             ON sessions ((a_session ->> 'user'));
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE psessions (
    id varchar(64) not null primary key,
    a_session jsonb
);
CREATE INDEX i_p__session_kind    ON psessions ((a_session ->> '_session_kind'));
CREATE INDEX i_p__httpSessionType ON psessions ((a_session ->> '_httpSessionType'));
CREATE INDEX i_p__session_uid     ON psessions ((a_session ->> '_session_uid'));
CREATE INDEX i_p_ipAddr           ON psessions ((a_session ->> 'ipAddr'));
CREATE INDEX i_p__whatToTrace     ON psessions ((a_session ->> '_whatToTrace'));


CREATE TABLE samlsessions (
    id varchar(64) not null primary key,
    a_session jsonb
);
CREATE INDEX i_a__session_kind ON samlsessions ((a_session ->> '_session_kind'));
CREATE INDEX i_a__utime        ON samlsessions ((cast(a_session ->> '_utime' as␣
↪bigint)));
CREATE INDEX i_a_ProxyID       ON samlsessions ((a_session ->> 'ProxyID'));
CREATE INDEX i_a__nameID       ON samlsessions ((a_session ->> '_nameID'));
CREATE INDEX i_a__assert_id    ON samlsessions ((a_session ->> '_assert_id'));
CREATE INDEX i_a__art_id       ON samlsessions ((a_session ->> '_art_id'));
CREATE INDEX i_a__saml_id      ON samlsessions ((a_session ->> '_saml_id'));

CREATE TABLE oidcsessions (
    id varchar(64) not null primary key,
    a_session jsonb
);
CREATE INDEX i_o__session_kind ON oidcsessions ((a_session ->> '_session_kind'));
CREATE INDEX i_o__utime        ON oidcsessions ((cast(a_session ->> '_utime' as bigint␣
↪)));

CREATE TABLE cassessions (
    id varchar(64) not null primary key,
    a_session jsonb
);
CREATE INDEX i_c__session_kind ON cassessions ((a_session ->> '_session_kind'));
CREATE INDEX i_c__utime        ON cassessions ((cast(a_session ->> '_utime' as bigint)));
CREATE INDEX i_c__cas_id       ON cassessions ((a_session ->> '_cas_id'));
CREATE INDEX i_c_pgtIou        ON cassessions ((a_session ->> 'pgtIou'));
```

LemonLDAP::NG configuration

Go in the Manager and set the session module to `Apache::Session::Browseable::PgJSON` for each session type you intend to use:

- `General parameters` » `Sessions` » `Session storage` » `Apache::Session module`
- `General parameters` » `Sessions` » `Persistent sessions` » `Apache::Session module`
- `CAS Service` » `CAS sessions module name`
- `OpenID Connect Service` » `Sessions` » `Sessions module name`
- `SAML2 Service` » `Advanced` » `SAML sessions module name`

Then, set the following module options:

Required parameters		
Name	Comment	Example
DataSource	The DBI string	dbi:Pg:database=lemonldap-ng
UserName	The database username	lemonldapng
Password	The database password	mysuperpassword
TableName	Table name (optional)	sessions
Commit	1	This setting is mandatory for PostgreSQL to work

Tip: Unlike other browseable modules, Pg::JSON does not require an `Index` parameter

Browseable MySQL session backend

Prerequisites

First, make sure you have installed the `DBD::mysql` perl module.

On Debian-based distributions

```
apt install libdbd-mysql-perl
```

On Fedora-based distributions

```
yum install 'perl(DBD::mysql)'
```

Create database schema

Create the following tables. You may skip the session types you are not going to use, but you need at least `sessions` and `psessions`

```
CREATE TABLE sessions (
    id varchar(64) not null primary key,
    a_session text,
    _whatToTrace varchar(64),
    _session_kind varchar(15),
```

(continues on next page)

(continued from previous page)

```
    ipAddr varchar(64),
    _utime bigint,
    _httpSessionType varchar(64),
    user varchar(64)
) DEFAULT CHARSET utf8;
CREATE INDEX i_s__whatToTrace ON sessions (_whatToTrace);
CREATE INDEX i_s__session_kind ON sessions (_session_kind);
CREATE INDEX i_s__utime ON sessions (_utime);
CREATE INDEX i_s_ipAddr ON sessions (ipAddr);
CREATE INDEX i_s__httpSessionType ON sessions (_httpSessionType);
CREATE INDEX i_s_user ON sessions (user);

CREATE TABLE psessions (
    id varchar(64) not null primary key,
    a_session text,
    _session_kind varchar(15),
    _httpSessionType varchar(64),
    _whatToTrace varchar(64),
    ipAddr varchar(64),
    _session_uid varchar(64)
)  DEFAULT CHARSET utf8;
CREATE INDEX i_p__session_kind ON psessions (_session_kind);
CREATE INDEX i_p__httpSessionType ON psessions (_httpSessionType);
CREATE INDEX i_p__session_uid ON psessions (_session_uid);
CREATE INDEX i_p_ipAddr ON psessions (ipAddr);
CREATE INDEX i_p__whatToTrace ON psessions (_whatToTrace);

CREATE TABLE samlsessions (
    id varchar(64) not null primary key,
    a_session text,
    _session_kind varchar(15),
    _utime bigint,
    ProxyID varchar(64),
    _nameID varchar(128),
    _assert_id varchar(64),
    _art_id varchar(64),
    _saml_id varchar(64)
)  DEFAULT CHARSET utf8;
CREATE INDEX i_a__session_kind ON samlsessions (_session_kind);
CREATE INDEX i_a__utime ON samlsessions (_utime);
CREATE INDEX i_a_ProxyID ON samlsessions (ProxyID);
CREATE INDEX i_a__nameID ON samlsessions (_nameID);
CREATE INDEX i_a__assert_id ON samlsessions (_assert_id);
CREATE INDEX i_a__art_id ON samlsessions (_art_id);
CREATE INDEX i_a__saml_id ON samlsessions (_saml_id);

CREATE TABLE oidcsessions (
    id varchar(64) not null primary key,
    a_session text,
    _session_kind varchar(15),
    _utime bigint
)  DEFAULT CHARSET utf8;
```

(continues on next page)

(continued from previous page)

```
CREATE INDEX i_o__session_kind ON oidcsessions (_session_kind);
CREATE INDEX i_o__utime ON oidcsessions (_utime);


CREATE TABLE cassessions (
    id varchar(64) not null primary key,
    a_session text,
    _session_kind varchar(15),
    _utime bigint,
    _cas_id varchar(128),
    pgtIou varchar(128)
) DEFAULT CHARSET utf8
CREATE INDEX i_c__session_kind ON cassessions (_session_kind);
CREATE INDEX i_c__utime        ON cassessions (_utime);
CREATE INDEX i_c__cas_id       ON cassessions (_cas_id);
CREATE INDEX i_c_pgtIou        ON cassessions (pgtIou);
```

LemonLDAP::NG configuration

Go in the Manager and set the session module to `Apache::Session::Browseable::PgJSON` for each session type you intend to use:

- `General parameters` » `Sessions` » `Session storage` » `Apache::Session module`
- `General parameters` » `Sessions` » `Persistent sessions` » `Apache::Session module`
- `CAS Service` » `CAS sessions module name`
- `OpenID Connect Service` » `Sessions` » `Sessions module name`
- `SAML2 Service` » `Advanced` » `SAML sessions module name`

Then, set the following module options:

Required parameters		
Name	Comment	Example
DataSource	The DBI string	dbi:mysql:database=lemonldap-ng
UserName	The database username	lemonldapng
Password	The database password	mysuperpassword
TableName	Table name (optional)	sessions
Index	Fields to index	refer to *List of fields to index by session type*

Browseable LDAP session backend

LemonLDAP::NG configuration

Go in the Manager and set the session module to `Apache::Session::Browseable::LDAP` for each session type you intend to use:

- `General parameters` » `Sessions` » `Session storage` » `Apache::Session module`
- `General parameters` » `Sessions` » `Persistent sessions` » `Apache::Session module`
- `CAS Service` » `CAS sessions module name`

- `OpenID Connect Service` » `Sessions` » `Sessions module name`
- `SAML2 Service` » `Advanced` » `SAML sessions module name`

The fill out the corresponding module parameters:

Required parameters		
Name	Comment	Example
ldapServer	URI of the server	ldap://localhost
ldapConfBase	DN of sessions branch	ou=sessions,dc=example,dc=com
ldapBindDN	Connection login	cn=admin,dc=example,dc=password
ldapBindPassword	Connection password	secret
Index	Fields to index	refer to *List of fields to index by session type*
Optional parameters		
Name	Comment	Default value
ldapObjectClass	Objectclass of the entry	applicationProcess
ldapAttributeId	Attribute storing session ID	cn
ldapAttributeContent	Attribute storing session content	description
ldapAttributeIndex	Attribute storing index	ou
ldapVerify	Perform certificate validation	require (use none to disable)
ldapCAFile	Path of CA file bundle	(system CA bundle)
ldapCAPath	Perform CA directory	(system CA bundle)

7.2.8 REST session backend

Session <type> can be 'global' for SSO sessions or 'persistent' for persistent sessions.

LL::NG portal provides REST end points for sessions management:

- GET /sessions/<type>/<session-id> : get session datas
- GET /sessions/<type>/<session-id>/<key> : get a session key value
- GET /sessions/<type>/<session-id>/[k1,k2] : get some session key value
- POST /sessions/<type> : create a session
- PUT /sessions/<type>/<session-id> : update some keys
- DELETE /sessions/<type>/<session-id> : delete a session

Sessions for connected users (used by *LLNG Proxy*):

- GET /session/my/<type> : get session datas
- GET /session/my/<type>/key : get session key
- DELETE /session/my : ask for logout

Authorizations for connected users (always enabled):

- GET /mysession/?authorizationfor=<base64-encoded-url>: ask if url is authorizated

This session backend can be used to share sessions stored in a non-network backend (like *file session backend*) or in a network backend protected with a firewall that only accepts HTTP flows.

Most of the time, REST session backend is used by Handlers installed on external servers.

To configure it, REST session backend will be set through Manager in global configuration (used by all Handlers), and the real session backend will be configured for local components in lemonldap-ng.ini.

Setup

Manager

First, activate REST in `General parameters` » `Plugins` » `Portal servers` » `REST session server`.

Then, set `Lemonldap::NG::Common::Apache::Session::REST` in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the following parameters (case sensitive):

Required parameters		
Name	Comment	Example
baseUrl	URL of sessions REST end point	http://auth.example.com/index.fcgi/sessions/global

Optional parameters		
Name	Comment	Example
user	Username to use for auth basic mechanism	
password	Password to use for auth basic mechanism	

Attention: By default, user password and other secret keys are hidden by LLNG REST server. You can force REST server to export their real values by selecting "Export secret attributes in REST" in the manager. This less secure option is disabled by default.

Apache

Sessions REST end points access must be allowed in Apache portal configuration (for example, access by IP range):

```
# REST/SOAP functions for sessions access (disabled by default)
<Location /index.fcgi/sessions>
    Require 192.168.2.0/24
</Location>
```

Real session backend

Real session backend will be configured in `lemonldap-ng.ini`, in `portal` section (the portal hosts the REST service for sessions, and will do the link between REST requests and real sessions).

For example, if real sessions are stored in *files*:

```
[portal]
globalStorage = Apache::Session::File
globalStorageOptions = { 'Directory' => '/var/lib/lemonldap-ng/sessions/', 'LockDirectory
↪' => '/var/lib/lemonldap-ng/sessions/lock/', }
```

Tip: Session explorer and "single session" features can't be used using this backend. Session explorer and portal must be launched with real backend.

By default, only few sessions keys are shared by REST (authenticationLevel, groups, ipAddr, _startTime, _utime, _lastSeen, _session_id), you need to define which other keys you want to share in `General parameters` » `Plugins` » `Portal servers` » `SOAP/REST exported attributes`.

You must start with + to keep default keys, else they will not be shared. For example:

```
+ uid cn mail
```

To share only the listed attributes:

```
authenticationLevel groups ipAddr _startTime _utime _lastSeen _session_id uid cn mail
```

7.2.9 SOAP session backend

LL::NG portal provides SOAP end points for sessions management:

- **sessions/**: read only access to sessions (enough for distant Handlers)
- **adminSessions/**: read/write access to sessions (required for distant Portal, distant Manager or distant Handlers which modify sessions)

This session backend can be used to share sessions stored in a non-network backend (like *file session backend*) or in a network backend protected with a firewall that only accepts HTTP flows.

Most of the time, SOAP session backend is used by Handlers installed on external servers.

To configure it, SOAP session backend will be set through Manager in global configuration (used by all Hanlders), and the real session backend will be configured for local components in lemonldap-ng.ini.

Setup

Manager

First, active SOAP in `General parameters` » `Advanced parameters` » `SOAP`.

Then, set `Lemonldap::NG::Common::Apache::Session::SOAP` in `General parameters` » `Sessions` » `Session storage` » `Apache::Session module` and add the following parameters (case sensitive):

Required parameters		
Name	Comment	Example
proxy	URL of sessions SOAP end point	http://auth.example.com/index.fcgi/sessions

Tip: Use /adminSessions if the Handler need to modify the session, for example if you configured an idle timeout.

By default, only few sessions keys are shared by SOAP (authenticationLevel, groups, ipAddr, _startTime, _utime, _lastSeen, _session_id), you need to define which other keys you want to share in `General parameters` » `Plugins` » `Portal servers` » `SOAP/REST exported attributes`.

You must start with + to keep default keys, else they will not be shared. For example:

```
+ uid cn mail
```

To share only the listed attributes:

```
_utime _session_id uid cn mail
```

Apache

Sessions SOAP end points access must be allowed in Apache portal configuration (for example, access by IP range):

```
# SOAP functions for sessions management (disabled by default)
<Location /index.fcgi/adminSessions>
    Require 192.168.2.0/24
</Location>

# SOAP functions for sessions access (disabled by default)
<Location /index.fcgi/sessions>
    Require 192.168.2.0/24
</Location>
```

Real session backend

Real session backend will be configured in `lemonldap-ng.ini`, in `portal` section (the portal hosts the SOAP service for sessions, and will do the link between SOAP requests and real sessions).

For example, if real sessions are stored in *files*:

```
[portal]
globalStorage = Apache::Session::File
globalStorageOptions = { 'Directory' => '/var/lib/lemonldap-ng/sessions/', 'LockDirectory
↪' => '/var/lib/lemonldap-ng/sessions/lock/', }
```

Tip: If your sessions explorer is on the same server that the portal, either use the **adminSessions** end point in Manager configuration, or override the `globalStorage` and `globalStorageOptions` parameters in section all (and not portal) of `lemonldap-ng.ini`.

CHAPTER

EIGHT

WRITING RULES AND HEADERS

Lemonldap::NG manages applications by their hostname (Apache's virtualHosts). Rules are used to protect applications, headers are HTTP headers added to the request to give datas to the application (for logs, profiles,...).

Attention: Note that variables designed by $xx correspond to the name of the *exported variables* or *macro names* except for `$ENV{<cgi-header>}` which correspond to CGI header (`$ENV{REMOTE_ADDR}` for example).

8.1 Available $ENV variables

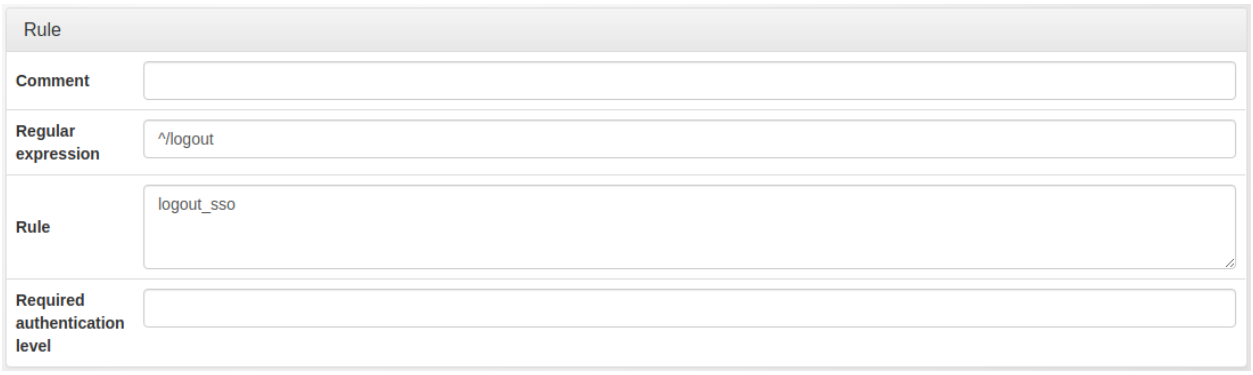
The %ENV table provides:

- all headers in CGI format (`User-Agent` becomes `HTTP_USER_AGENT`)
- some CGI variables depending on the context:
  - For portal: all CGI standard variables (you can add custom headers using `fastcgi_param` with Nginx),
  - For Apache handler: REMOTE_ADDR, QUERY_STRING, REQUEST_URI, SERVER_PORT, REQUEST_METHOD,
  - For Nginx handler: all variables given by `fastcgi_param` commands.
- For portal:
  - $ENV{urldc} : Origin URL before Handler redirection, in cleartext
  - $ENV{_url} : Origin URL before Handler redirection, base64 encoded

See also *extended functions*.

8.2 Rules

A rule associates a regular expression to a Perl boolean expression or a keyword.



Examples:

Goal	Regular expression	Rule
Restrict /admin/ directory to user bart.simpson	^/admin/	
Restrict /js/ and /css/ directory to authenticated users	^/(css|js)/	accept
Deny access to /config/ directory	^/config/	deny
Do not restrict /public/	^/public/	skip
Do not restrict /skip/ and restrict other to authenticated users	^/skip/	$ENV{REQUEST_URI} =~ /skip/ ? skip : 1
Makes authentication optional, but authenticated users are seen as such (that is, user data are sent to the app through HTTP headers)	^/forum/	unprotect
Restrict access to the whole site to users that have the LDAP description field set to "LDAP administrator" (must be set in exported variables)	default	

The "**default**" access rule is used if no other access rule match the current URL.

Tip: See *the rules examples page* for a few common use cases

Tip:

- Comments can be used to order your rules: rules are applied in the alphabetical order of comment (or regexp in there is no comment). See *security chapter* to learn more about writing good rules.
- See *performances* to know how to use macros and groups in rules.

Rules can also be used to intercept logout URL:

Goal	Regular expression	Rule
Logout user from Lemonldap::NG and redirect it to http://intranet/	^/index.php?logout	logout_sso http://intranet/
Logout user from current application and redirect it to the menu **(Apache only)**	^/index.php?logout	logout_app https://auth.example.com/
Logout user from current application and from Lemonldap::NG and redirect it to http://intranet/ **(Apache only)**	^/index.php?logout	logout_app_sso http://intranet/

Danger: `logout_app` and `logout_app_sso` rules are not available on Nginx, only on Apache.

By default, user will be redirected on portal if no URL defined, or on the specified URL if any.

Attention: Only current application is concerned by logout_app* targets. Be careful with some applications which doesn't verify Lemonldap::NG headers after having created their own cookies. If so, you can redirect users to a HTML page that explain that it is safe to close browser after disconnect.

8.2.1 Rules based on authentication level

LLNG set an "authentication level" during authentication process. This level depends on authentication backend used by this user. Default values are:

- 0 for *Null*
- 1 for *CAS*, *old OpenID-2*, *Facebook*,...
- 2 for web-form based authentication (*LDAP*, *DBI*,...)
- 3 for *Yubikey*
- 4 for *Kerberos*
- 5 for *SSL*

There are three ways to impose users a higher authentication level:

- writing a rule based on authentication level: `$authenticationLevel > 3`
- since 2.0, set a minimum level in virtual host options (default value for ALL access rules)
- since 2.0.7, a minimum authentication level can be set for each URI access rule. Useful if URI are protected by different types of handler (AuthBasic -> level 2, Main -> level set by authentication backend).

Tip: Instead of returning a 403 code, "minimum level" returns user to a form that explain that a higher level is required and propose to reauthenticate himself.

8.3 Headers

Headers are associations between an header name and a perl expression that returns a string. Headers are used to give user data to the application.

Examples:

Goal	Header name	Header value
Give the uid (for accounting)	Auth-User	$uid
Give a static value	Some-Thing	"static-value"
Give display name	Display-Name	$givenName." ".$surName
Give a non ascii data	Display-Name	

As described in *performances chapter*, you can use macros, local macros,...

Attention:

- Since many HTTP servers refuse non ascii headers, it is recommended to use encode_base64() function to transmit those headers
- Don't forget to add an empty string as second argument to encode_base64 function to avoid a "newline" characters insertion in result
- Header names must contain only letters and "-" character. With Nginx, you can bypass this restriction by using `underscores_in_headers on;` directive

Tip: By default, SSO cookie is hidden. So protected applications cannot retrieve SSO session key. But you can forward this key if absolutely needed:

```
Session-ID => $_session_id
```

8.4 Available functions

In addition to macros and name, you can use some functions in rules and headers:

- *LLNG extended functions*
- *Your custom functions*

8.5 Wildcards in hostnames

NEW Since 2.0, a wildcard can be used in virtualhost name (not in aliases !): `*.example.com` matches all hostnames that belong to `example.com` domain. Version 2.0.9 improves this and allows better wildcards such as `test-*.example.com` or `test-%.example.com`. The `%` wilcard doesn't match subdomains.

Even if a wildcard exists, if a virtualhost is explicitly declared, this rule is applied. Example with precedence order for test.sub.example.com:

1. test.sub.example.com
2. test%.sub.example.com
3. test*.sub.example.com
4. %.sub.example.com
5. *.sub.example.com
6. *.example.com (`%.example.com` does not match test.sub.example.com)

CHAPTER

NINE

VARIABLES

9.1 Presentation

Variables can be used in rules and headers. All rules are concerned:

- Access rule in virtual host
- SAML IDP preselection
- Session opening
- …

Variables are stored in the user session. We can distinguish several kind of variables:

- internal variables, managed by LemonLDAP::NG
- *exported variables* collected from UserDB backend
- *macro and groups*

When you know the key of the variable, you just have to prefix it with the dollar sign to use it, for example to test if `uid` variable match `coudot` :

```
$uid eq "coudot"
```

Tip: You can inspect a user session with the sessions explorer (in Manager)

Below are documented internal variables.

9.2 Modules

Register what module was used for authentication, user data, password, …

Key	Description
_auth	Authentication module
_userDB	User module
_passwordDB	Password module
_issuerDB	Issuer module (can be multivalued)
_authChoice	User choice done if *authentication choice* was used
_authMulti	Full name of authentication module (with `#label`) used in Multi
_userDBMulti	Full name of user module (with `#label`) used in Multi

9.3 Connection

Datas concerning the first connection to the portal

Key	Description
ipAddr	IP of the user (special care must be taken is you run the portal *behind a reverse proxy*)
_time-zone	Timezone of the user, set with javascript from standard login form (will be empty if other authentication methods are used)
_url	URL used before being redirected to the portal (empty if portal was used as entry point)

9.4 Authentication

Datas around the authentication process.

Key	Description
_session_id	Session identifier (carried in cookie)
_user	User found from login process
_password	Password found from login process (only if *password store in session* is configured)
authenticationLevel	Authentication level

9.5 Dates

Key	Description
_utime	Timestamp of session creation
_startTime	Date of session creation
_updateTime	Date of session last modification
_lastAuthnUTime	Timestamp of last authentication time

9.6 SAML

Datas related to SAML protocol

Key	Description
_idp	Name of IDP used for authentication
_idpConfKey	Configuration key of IDP used for authentication
_samlToken	SAML token
_lassoSessionDump	Lasso session dump
_lassoIdentityDump	Lasso identity dump

9.7 Notifications

Key	Description
_notification_*id*	Date of validation of the notification *id*

9.8 Login history

Key	Description
_loginHistory	HASH of login success and failures

9.9 LDAP

Only with UserDB LDAP.

Key	Description
_dn	Distinguished name

9.10 OpenID

Key	Description
_openid_*id*	Consent to share attribute *id* through OpenID

9.11 OpenID Connect

Key	Description
_oidc_id_token	ID Token
_oidc_OP	Configuration key of OP used for authentication
_oidc_access_token	OAuth2 Access Token used to get UserInfo data
_oidc_consent_scope_*rp*	Scope for which consent was given for RP *rp*
_oidc_consent_time_*rp*	Time when consent was given for RP *rp*

9.12 Other

Key	Description
_appsListOrder	Order of categories in the menu
_session_kind	Type of session (SSO, Persistent, ...)

CHAPTER

TEN

PROTECT YOUR APPLICATION

10.1 Presentation

Your application can know the connected user using:

- REMOTE_USER environment variable (with local Handler or SetEnvIf trick)
- HTTP header (in all cases)

To get more information on user (name, mail, etc.), you have to read *HTTP headers*.

Tip: If your application is based on Perl CGI package, you can simply replace CGI by *Lemonldap::NG::Handler::CGI*

10.2 Code snippet

Examples with a *configured header* named 'Auth-User':

10.2.1 Perl

```
print "Connected user: ".$ENV{HTTP_AUTH_USER};
```

10.2.2 PHP

```
print "Connected user: ".$_SERVER["HTTP_AUTH_USER"];
```

10.3 Perl auto-protected CGI

LL::NG now uses FastCGI instead of CGI, but you still can write your own protected CGI.

First create a PSGI module based on Lemonldap::NG::Handler:

```
package My::PSGI;

use base "Lemonldap::NG::Handler::PSGI"; # or Lemonldap::NG::Handler::PSGI::OAuth2,␣
↪etc...
```

(continues on next page)

(continued from previous page)

```
sub init {
    my ($self,$args) = @_;
    $self->protection('manager');
    $self->SUPER::init($args) or return 0;
    $self->staticPrefix("/static");
    $self->templateDir("/usr/share/lemonldap-ng/portal/templates");
    # See Lemonldap::NG::Common::PSGI for more
    #...
    # Return a boolean. If false, then error message has to be stored in
    # $self->error
    return 1;
}

sub handler {
    my ( $self, $req ) = @_;

    # Will be called only if authorisated
    my $userId = $self->userId($req);
    #...

    # Return JSON
    # $self->sendJSONresponse(...);

    # or Return HTML
    $self->sendHtml($req, "myskin/mytemplate", ( params => { 'userId' => $userId }) );
}
```

They create a FCGI script like this:

```
#!/usr/bin/env perl

use My::PSGI;
use Plack::Handler::FCGI;

Plack::Handler::FCGI->new->run( My::PSGI->run() );
```

See our LLNG Nginx/Apache configurations to see how to launch it or read PSGI/Plack documentation.

The protection parameter must be set when calling the init() method:

- `none`: no protection
- `authenticate`: check authentication but do not manage authorization
- `manager`: rely on virtual host configuration in Manager
- `rule: xxx`: apply a specific rule

CHAPTER

ELEVEN

FORM REPLAY

11.1 Presentation

Form replay allows you to open a session on a protected application by filling a HTML POST login form and autosubmitting it, without asking anything to the user.

Danger: This kind of SSO mechanism is not clean, and can lead to problems, like local password blocking, local session not well closed, etc.

Please always try to find another solution to protect your application with LL::NG. At least, check if it is not a *known application*, or *try to adapt its source code*.

If you configure form replay with LL::NG, the Handler will detect forms to fill, add a javascript in the html page to fill form fields with dummy datas and submit it, then intercept the POST request and add POST data in the request body.

POST data can be static values or computed from user's session.

Tip: To post user's password, you must enable *password storing*. In this case you will be able to use `$_password` to fill any password POST field.

11.2 Configuration

You should grab some information:

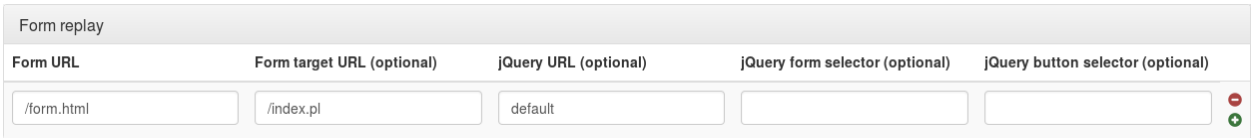
- URI of the html page which contains the form
- URI the html form is sent to
- Does the html page load jQuery ? If not, grab a jQuery URL reachable by user (any version over jQuery 1.0 is suitable)
- are there several html forms in the page ? If so, get a jQuery selector for the form you want to post
- is user required to click on a button, for example in order to perform some script ? If so, get a jQuery selector for that button
- names and values of the fields you want to control

If you don't know jQuery selector, just be aware that they are similar to css selectors: for example, button#foo points to the html button whose id is "foo", and .bar points to all html elements of css class "bar".

For example:

- Form page URI: /login.php
- Target URI: /process.php (if you let this parameter empty, target URI is supposed to be the same as form page URI)
- jQuery URL: http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js (if you let this parameter empty, jQuery is supposed to be already loaded; you can also set `default` to point to jQuery URL of LL::NG portal)
- jQuery form selector: #loginForm (if you let this parameter empty, browser will fill and submit any html form)
- jQuery button selector: button.validate (if you let this parameter empty, the form will be submitted but no button will be clicked; if you set it to "none", no button will be clicked and the form will be filled but not submitted)
- Fields:
  - postuid: $uid
  - postmail: $mail
  - poststatic: 'static'

Go in Manager, "Virtual Hosts" » *virtualhost* » "Form replay" and click on "New form replay".



Fill values here:

- **Form URL**: /login.php
- **Target URL**: /process.php
- **jQuery URL**: http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js
- **jQuery form selector**: #loginForm
- **jQuery button selector**: button.validate

Then click on `New variable` and add all data with their values, for example:

Tip: You can define more than one form replay URL per virtual host.

CHAPTER

TWELVE

CUSTOM HANDLERS

LLNG provides Perl libraries that can be easily used by inheritance. So you can write your own handlers but you need first to understand *Handler architecture*

12.1 Add a new handler type

1. Write your new Module (in Lemonldap/NG/Handler/Lib for example) that overload some Lemonldap::NG::Handler::Main methods
2. Write a wrapper in each platform directory *(see Lemonldap::NG::Handler::Apache2::AuthBasic or Lemonldap::NG::Handler::Server::AuthBasic for examples)*

Wrapper usually look at this:

```
package Lemonldap::NG::Handler::ApacheMP2::MyType;

use base 'Lemonldap::NG::Handler::ApacheMP2::Main', 'Lemonldap::NG::Handler::Lib::MyType
↪';

1;
```

12.1.1 Enable it

Your wrappers must be named "Lemonldap::NG::Handler::<platform>::<type>" where <platform> is the target (ApacheMP2 or Server) and <type> is the name you've chosen.

You can enable it either:

- by setting a `PerlSetVar VHOSTTYPE <type>` in the Apache configuration file
- by setting a `fastcgi_param VHOSTTYPE <type>` in the Nginx configuration file
- by adding it to the menu: add its name in `vhostType` "select" declaration (file `lemonldap-ng-manager/lib/Lemonldap/NG/Build/Attributes`) and rebuild LLNG

Note that configuration parameter can be set only in lemonldap-ng.ini configuration file *(section Handler)*.

12.2 Add a new platform

LLNG provides 3 platforms:

- ApacheMP2
- FastCGI server *(Nginx is build from there)*
- Auto-protected PSGI

If you want to add another, you must write:

- the platform launcher file that launch the required type (see `lemonldap-ng-handler/lib/Lemonldap/NG/Handler/ApacheMP2` file for example)
- write the main platform file (`Lemonldap::NG::Handler::MyPlatform::Main`) that provides required method (see `lemonldap-ng-handler/lib/Lemonldap/NG/Handler/*/Main` for examples) and inherits from `Lemonldap::NG::Handler::Main`
- write the "type" wrapper files (AuthBasic,...).

Wrapper usually look at this:

```
package Lemonldap::NG::Handler::MyPlatform::AuthBasic;

use base 'Lemonldap::NG::Handler::MyPlatform::Main',
↪'Lemonldap::NG::Handler::Lib::AuthBasic';

1;
```

12.3 Old fashion Nginx handlers

Attention: There is no need to use this feature now. It is kept for compatibility.

Three actions are needed:

- declare your own module in the manager "General Parameters >> Advanced Parameters >> Custom handlers (Nginx)". Key is the name that will be used below and value is the name of the custom package,
- in your Nginx configuration file, add `LLTYPE=<name>;` in the `location = /lmauth {...}` paragraph
- restart FastCGI server(s) *(reload is not enough here)*

CHAPTER

THIRTEEN

WEBSERVICES / API

13.1 Presentation

WebServices and API are mostly requested by an application, and not the end-user itself. In this case, you can not rely on LL::NG standard Handler to protect the webservice, as it will expect a cookie, which is not defined in the application requesting the service.

LL::NG offers several solutions to protect this kind of service.

13.2 ServiceToken Handler

Two Handlers will be used:

- The frontal Handler that will protect the web application, and will forge a specific token
- The backend Handler that will protect the web service, and will consume the token

See *ServiceToken Handler documentation*.

13.3 OAuth2 endpoints

We suppose here that LL::NG is acting as *OpenID Connect provider*. The web application will then be able to get an access token from LL::NG. This token could be sent to the webservice that can then validate it against LL::NG OAuth2 endpoints.

13.3.1 UserInfo

You can use the UserInfo endpoint, which requires the access token to deliver user attributes.

For example:

```
curl \
  -H "Authorization: Bearer a74d504ec9e784785e70a1da2b95d1d2" \
  https://auth.example.ccom/oauth2/userinfo | json_pp
```

```
{
  "family_name" : "OUDOT",
  "name" : "Clément OUDOT",
```

(continues on next page)

(continued from previous page)

```
  "email" : "clement@example.com",
  "sub" : "coudot"
}
```

13.3.2 Introspection

Introspection endpoint is defined in RFC 7662. It requires an authentication (same as the authentication for the token endpoint) and takes to access token as parameter.

For example:

```
curl \
  -H "Authorization: Basic bGVtb25sZGFwOnNlY3JldA==" \
  -X POST -d "token=a74d504ec9e784785e70a1da2b95d1d2" \
  https://auth.example.com/oauth2/introspect | json_pp
```

```
{
   "client_id" : "lemonldap",
   "sub" : "coudot",
   "exp" : 1572446485,
   "active" : true,
   "scope" : "openid profile address email phone"
}
```

13.4 OAuth2 Handler

We also suppose here that LL::NG is acting as *OpenID Connect provider*. But the webservice will be protected by the OAuth2 Handler and will just have to read the HTTP headers to know which user is connected.

```
curl \
   -H "Authorization: Bearer a74d504ec9e784785e70a1da2b95d1d2" \
   https://oauth2.example.ccom/rest/myapi
```

```
{
   "check" : "true",
   "user" : "coudot"
}
```

See *OAuth2 Handler documentation*.

CHAPTER

FOURTEEN

HTTP BASIC AUTHENTICATION

14.1 Presentation

Attention: For now, this feature is only supported by Apache handler.

Extract from the Wikipedia article:

In the context of an HTTP transaction, the basic access authentication is a method designed to allow a web browser, or other client program, to provide credentials – in the form of a user name and password – when making a request.

Before transmission, the username and password are encoded as a sequence of base-64 characters. For example, the user name Aladdin and password open sesame would be combined as Aladdin:open sesame – which is equivalent to QWxhZGRpbjpvcGVuIHNlc2FtZQ== when encoded in Base64. Little effort is required to translate the encoded string back into the user name and password, and many popular security tools will decode the strings "on the fly".

So HTTP Basic Authentication is managed through an HTTP header (`Authorization`), that can be forged by LL::NG, with this precautions:

- Data should not contains accents or special characters, as HTTP protocol only allow ASCII values in header (but depending on the HTTP server, you can use ISO encoded values)
- You need to forward the password, which can be the user main password (if *password is stored in session*, or any user attribute (if you keep secondary passwords in users database).

14.2 Configuration

The Basic Authentication relies on a specific HTTP header, as described above. So you have just to declare this header for the virtual host in Manager.

For example, to forward login (`$uid`) and password (`$_password` if *password is stored in session*):

```
Authorization => "Basic ".encode_base64("$uid:$_password", "")
```

LL::NG provides a special function named *basic* to build this header.

So the above example can also be written like this:

```
Authorization => basic($uid,$_password)
```

Tip: The `basic` function will also force conversion from UTF-8 to ISO-8859-1, which should be accepted by most of HTTP servers.

CHAPTER

FIFTEEN

APPLICATIONS

15.1 Active Directory Federation Services

15.1.1 Presentation

Microsoft ADFS (Active Directory Federation Services) is an Identity/Service Provider, compatible with several protocols, including SAML 2.0.

Attention: This documentation does not explains how to setup ADFS, but give only tricks to make it works with LL::NG

15.1.2 ADFS as Identity Provider

When ADFS is declared as an Identity Provider in LemonLDAP::NG, you need to take care of the following items:

- HTTPS is mandatory on LL::NG portal
- You need to use a certificate in LL::NG SAML metadata instead of a raw public key
- Activate option `Use specific query_string method` in SAML Service
- Use SHA1 instead of SHA256 as signature algorithm on ADFS if using a Lasso version < 2.5.0
- Force SAML response to be sent by POST and not Artifact (signature verification fails with Artifact)
- Enable `Allow proxy authentication` in IDP options on LL::NG side

15.2 Alfresco

15.2.1 Presentation

Alfresco is an ECM/BPM software.

Since 4.0 release, it offers an easy way to configure SSO thanks to authentication subsystems.

Authentication against LL::NG can be done through:

- HTTP headers (LL::NG Handler)
- SAML 2 (LL::NG as SAML2 IDP)

Tip: Alfresco now recommends SAML2 method

15.2.2 HTTP headers

Alfresco

Tip: The official documentation can be found here: http://docs.alfresco.com/4.0/tasks/auth-alfrescoexternal-sso.html

You need to find the following files in your Alfresco installation:

- `alfresco-global.properties` (ex: `tomcat/shared/classes/alfresco-global.properties`)
- `share-config-custom.xml` (ex: `tomcat/shared/classes/alfresco/web-extension/share-config-custom.xml`)

The first will allow one to configure SSO for the alfresco webapp, and the other for the share webapp.

Edit first `alfresco-global.properties` and add the following:

```
### SSO ###
authentication.chain=external1:external
external.authentication.enabled=true
external.authentication.defaultAdministratorUserNames=
external.authentication.proxyUserName=
external.authentication.proxyHeader=Auth-User
external.authentication.userIdPattern=
```

Edit then `share-config-custom.xml` and uncomment the last part. In the `<endpoint>`, change `<connector-id>` value to `alfrescoHeader` and change the `<userHeader>` value to `Auth-User`:

```
<config evaluator="string-compare" condition="Remote">
   <remote>
       <keystore>
          <path>alfresco/web-extension/alfresco-system.p12</path>
          <type>pkcs12</type>
          <password>alfresco-system</password>
      </keystore>

      <connector>
         <id>alfrescoCookie</id>
         <name>Alfresco Connector</name>
         <description>Connects to an Alfresco instance using cookie-based authentication
↪</description>
         <class>org.alfresco.web.site.servlet.SlingshotAlfrescoConnector</class>
      </connector>

      <connector>
         <id>alfrescoHeader</id>
         <name>Alfresco Connector</name>
         <description>Connects to an Alfresco instance using header and cookie-based␣
↪authentication</description>
         <class>org.alfresco.web.site.servlet.SlingshotAlfrescoConnector</class>
         <userHeader>Auth-User</userHeader>
      </connector>

      <endpoint>
         <id>alfresco</id>
         <name>Alfresco - user access</name>
         <description>Access to Alfresco Repository WebScripts that require user␣
↪authentication</description>
         <connector-id>alfrescoHeader</connector-id>
         <endpoint-url>http://localhost:8080/alfresco/s</endpoint-url>
         <identity>user</identity>
         <external-auth>true</external-auth>
      </endpoint>
   </remote>
</config>
```

You need to restart Tomcat to apply changes.

Danger: Now you can log in with a simple HTTP header. You need to restrict access to Alfresco to LL::NG.

LL::NG

Headers

Just set the `Auth-User` header with the attribute that carries the user login, for example `$uid`.

Rules

Set the default rule to what you need.

Other rules:

- Unprotect access to some resources: `^/share/res => unprotect`
- Catch logout: `^/share/page/dologout => logout_app_sso`

15.2.3 SAML2

Alfresco

Install SAML Alfresco module package:

```
cp alfresco-saml-repo-1.0.1.amp <ALFRESCO_HOME>/amps
cp alfresco-saml-share-1.0.1.amp <ALFRESCO_HOME>/amps_share
./bin/apply_amp.sh
```

Generate SAML certificate:

```
keytool -genkeypair -alias my-saml-key -keypass change-me -storepass change-me -keystore␣
↪my-saml.keystore -storetype JCEKS
```

Export the keystore:

```
mv my-saml.keystore alf_data/keystore
cat <<EOT > alf_data/keystore/my-saml.keystore-metadata.properties
aliases=my-saml-key
keystore.password=change-me
my-saml-key.password=change-me
EOT
cat <<EOT >> tomcat/shared/classes/alfresco-global.properties

saml.keystore.location=\${dir.keystore}/my-saml.keystore
saml.keystore.keyMetaData.location=\${dir.keystore}/my-saml.keystore-metadata.properties
EOT
```

Edit then `share-config-custom.xml`:

```
...
    <config evaluator="string-compare" condition="CSRFPolicy" replace="true">



    <!--
```

(continues on next page)

(continued from previous page)

```
        If using https make a CSRFPolicy with replace="true" and override the properties␣
↪section.
        Note, localhost is there to allow local checks to succeed.



        I.e.
        <properties>
            <token>Alfresco-CSRFToken</token>
            <referer>https://your-domain.com/.*|http://localhost:8080/.*</referer>
            <origin>https://your-domain.com|http://localhost:8080</origin>
        </properties>
    -->



        <filter>



            <!-- SAML SPECIFIC CONFIG -  START -->



            <!--
             Since we have added the CSRF filter with filter-mapping of "/*" we will␣
↪catch all public GET to avoid them
             having to pass through the remaining rules.
             -->
            <rule>
                <request>
                    <method>GET</method>
                    <path>/res/.*</path>
                </request>
            </rule>



            <!-- Incoming posts from IDPs do not require a token -->
            <rule>
                <request>
                    <method>POST</method>
                    <path>/page/saml-authnresponse|/page/saml-logoutresponse|/page/saml-
↪logoutrequest</path>
                </request>
            </rule>



            <!-- SAML SPECIFIC CONFIG -  STOP -->
```

(continues on next page)

(continued from previous page)

```
            <!-- EVERYTHING BELOW FROM HERE IS COPIED FROM share-security-config.xml -->



            <!--
             Certain webscripts shall not be allowed to be accessed directly form the␣
↪browser.
             Make sure to throw an error if they are used.
             -->
            <rule>
                <request>
                    <path>/proxy/alfresco/remoteadm/.*</path>
                </request>
                <action name="throwError">
                    <param name="message">It is not allowed to access this url from your␣
↪browser</param>
                </action>
            </rule>



            <!--
             Certain Repo webscripts should be allowed to pass without a token since␣
↪they have no Share knowledge.
             TODO: Refactor the publishing code so that form that is posted to this URL␣
↪is a Share webscript with the right tokens.
             -->
            <rule>
                <request>
                    <method>POST</method>
                    <path>/proxy/alfresco/api/publishing/channels/.+</path>
                </request>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
            </rule>



            <!--
             Certain Surf POST requests from the WebScript console must be allowed to␣
↪pass without a token since
             the Surf WebScript console code can't be dependent on a Share specific␣
↪filter.
             -->
            <rule>
                <request>
                    <method>POST</method>
```

(continues on next page)

(continued from previous page)

```
                    <path>/page/caches/dependency/clear|/page/index|/page/surfBugStatus|/
↪page/modules/deploy|/page/modules/module|/page/api/javascript/debugger|/page/console</
↪path>
                </request>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
            </rule>



            <!-- Certain Share POST requests does NOT require a token -->
            <rule>
                <request>
                    <method>POST</method>
                    <path>/page/dologin(\?.+)?|/page/site/[^/]+/start-workflow|/page/
↪start-workflow|/page/context/[^/]+/start-workflow</path>
                </request>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
            </rule>



            <!-- Assert logout is done from a valid domain, if so clear the token when␣
↪logging out -->
            <rule>
                <request>
                    <method>POST</method>
                    <path>/page/dologout(\?.+)?</path>
                </request>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
                <action name="clearToken">
                    <param name="session">{token}</param>
                    <param name="cookie">{token}</param>
                </action>
            </rule>
```

(continues on next page)

(continued from previous page)

```
            <!-- Make sure the first token is generated -->
            <rule>
                <request>
                    <session>
                        <attribute name="_alf_USER_ID">.+</attribute>
                        <attribute name="{token}"/>
                        <!-- empty attribute element indicates null, meaning the token␣
↪has not yet been set -->
                    </session>
                </request>
                <action name="generateToken">
                    <param name="session">{token}</param>
                    <param name="cookie">{token}</param>
                </action>
            </rule>



            <!-- Refresh token on new "page" visit when a user is logged in -->
            <rule>
                <request>
                    <method>GET</method>
                    <path>/page/.*</path>
                    <session>
                        <attribute name="_alf_USER_ID">.+</attribute>
                        <attribute name="{token}">.+</attribute>
                    </session>
                </request>
                <action name="generateToken">
                    <param name="session">{token}</param>
                    <param name="cookie">{token}</param>
                </action>
            </rule>



            <!--
             Verify multipart requests from logged in users contain the token as a␣
↪parameter
             and also correct referer & origin header if available
             -->
            <rule>
                <request>
                    <method>POST</method>
                    <header name="Content-Type">multipart/.+</header>
                    <session>
                        <attribute name="_alf_USER_ID">.+</attribute>
                    </session>
                </request>
                <action name="assertToken">
                    <param name="session">{token}</param>
                    <param name="parameter">{token}</param>
```

(continues on next page)

(continued from previous page)

```
                </action>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
            </rule>



            <!--
             Verify that all remaining state changing requests from logged in users'␣
↪requests contains a token in the
             header and correct referer & origin headers if available. We "catch" all␣
↪content types since just setting it to
             "application/json.*" since a webscript that doesn't require a json request␣
↪body otherwise would be
             successfully executed using i.e."text/plain".
             -->
            <rule>
                <request>
                    <method>POST|PUT|DELETE</method>
                    <session>
                        <attribute name="_alf_USER_ID">.+</attribute>
                    </session>
                </request>
                <action name="assertToken">
                    <param name="session">{token}</param>
                    <param name="header">{token}</param>
                </action>
                <action name="assertReferer">
                    <param name="referer">{referer}</param>
                </action>
                <action name="assertOrigin">
                    <param name="origin">{origin}</param>
                </action>
            </rule>
        </filter>
    </config>
...
```

Configure SAML service provider using the Alfresco admin console (/alfresco/s/enterprise/admin/admin-saml).

Set the following parameters:

- Enable SAML Authentication (SSO): on
- Authentication service URL: https://auth.example.com/saml/singleSignOn
- Single Logout URL: https://auth.example.com/saml/singleLogout
- Single logout return URL: https://auth.example.com/saml/singleLogoutReturn
- Entity identification: http://alfresco.myecm.org:8080/share

- User ID mapping: Subject/NameID

To finish with Alfresco configuration, tick the "Enable SAML authentication (SSO)" box.

LL::NG

Configure SAML service and set a certificate as signature public key in metadata.

Export Alfresco SAML Metadata from admin console and import them in LL::NG.

In the authentication response option, set:

- Default NameID Format: Unspecified
- Force NameID session key: uid

And you can define these exported attributes:

- GivenName
- Surname
- Email

15.2.4 Other resources

- DevCon 2012: Unlocking the Secrets of Alfresco Authentication, Mehdi Belmekki
- Setting up Alfresco SAML authentication with LemonLDAP::NG

15.3 Amazon Web Services

Amazon Web Services allows one to delegate authentication through SAML2.

15.3.1 SAML

- Make sure you have followed the steps here.
- Go to https://your.portal.com/saml/metadata and save the resulting file locally.
- In each AWS account, go to IAM -> Identity providers -> Create Provider.
- Select `SAML` as the provider type
- Choose a name (best if kept consistent between accounts), and then choose the metadata file you saved above.
- Looking again at the links on the left side of the page, go to Roles -> Create role
- Choose `SAML / Saml 2.0 federation`
- Select the provider you just configured, click `Allow programmatic and AWSManagement Console access` which will fill in the rest of the form for you, then click next.
- Set whatever permissions you need to and then click `Review`.
- Choose a name for the role. These will shown to people when they log in, so make them descriptive. We have different accounts for different regions of the world, so I put the region into the role name so people know which account is which.

Attention: If you have only one role, the configuration is simple. If you have multiple roles for different people, it is a little trickier. As you will see, the SAML attributes are not dynamic, so you have to set them in the session when a user logs in or use a custom function. In this example, I wanted to avoid managing custom functions on all the servers, so the SAML attributes are set in the session. We also use LDAP for user information, so I will describe that. In our LDAP tree, each user has attributes which are used quite heavily for dynamic groups and authorisation. You will want something similar, using whatever attribute makes sense to you. For example:

```
dn: uid=user,ou=people,dc=your,dc=com
...
ou: sysadmin
ou: database
ou: root
```

- Assuming you use the web interface to manage lemonldap, go to General Parameters -> Authentication parameters -> LDAP parameters -> Exported variables. Here set the key to the LDAP attribute and the value to something sensible. I keep them the same to make it easy.
- Now go to *Variables -> Macros*. Here set up variables which will be computed based on the attributes you exported above. You will need to emit strings in this format `arn:aws:iam::account-number:role/role-name1,arn:aws:iam::account-number:saml-provider/provider-name`. The parts you need to change are `account-number`, `role-name1` and `provier-name`. The last two will be the provider name and role names you just set up in AWS.
- Perl works in here, so something like this is valid: `aws_eu_role -> $ou =~ sysadmin ? "arn:aws..." : "arn:..."`
- If it easier, split multiple roles into different macros. Then tie all the variables you define together into one string concatenating them with whatever is in General Parameters -> Advanced Parameters -> Separator. Actually click into this field and move around with the arrow keys to see if there is a space, since spaces can be part of the separator.
- Remember macros are defined alphanumerically, so you want one right at the end, like `z_aws_roles -> join("; ", $role_name1, $role_name2, ...)`
- On the left again, click `SAML service providers`, then `Add SAML SP`.
- Enter a name, click ok, then select it on the left. Select `Metadata`, then enter `https://signin.aws.amazon.com/static/saml-metadata.xml` in the `URL` field, then click load.
- Click `Exported attributes` on the left, then `Add attribute` twice to add two attributes. The first field is the name of a variable set in the user's session:
  - `_whatToTrace` -> `https://aws.amazon.com/SAML/Attributes/RoleSessionName` (leave the rest)
  - `z_aws_roles` (the macro name you defined above) -> `https://aws.amazon.com/SAML/Attributes/Role` (leave the rest)
- On the left, select Options -> Security -> Enable use of IDP initiated URL -> On
- Select General Parameters -> Portal -> Menu -> Categories and applications
- Select a category or create a new one if you need to. Then click `New application`.
- Enter a name etc. For the URL, use `https://your.portal.com/saml/singleSignOn?IDPInitiated=1&sp=urn:amazon:webservices`
- Display application should be set to `Enabled`
- Go to your portal, click on the link, and check that it works!

15.4 AWX (Ansible Tower)

15.4.1 Presentation

AWX is the upstream version for Ansible Tower.

This documentation explains how to interconnect LemonLDAP::NG and AWX using SAML 2.0 protocol.

You can find the Official AWX documentation about this topic here : https://docs.ansible.com/ansible-tower/latest/html/administration/ent_auth.html#saml-authentication-settings Please read it before the LLNG doc.

15.4.2 Configuration

This page assumes you already have configured the SAML Service in LemonLDAP::NG, if not please follow : *SAML service configuration*

AWX SAML Key & Certificate

You'll need a private key and the corresponding certificate to setup saml in AWX, you can do it with your pki or with openssl on your machine :

```
openssl req -x509 -newkey rsa:4096 -keyout saml-awx.key -out saml-awx.crt -days 3650 -
↪nodes
```

LLNG SAML Certificate

AWX need a certificate for the IDP signature, a public key won't work. You can either just generate a certificate from the private key and put it in AWX conf, or you can do it globally.

Generate Certificate from Key

You can find your private key in : SAML2 Service -> Security Parameters -> Signature -> Private Key

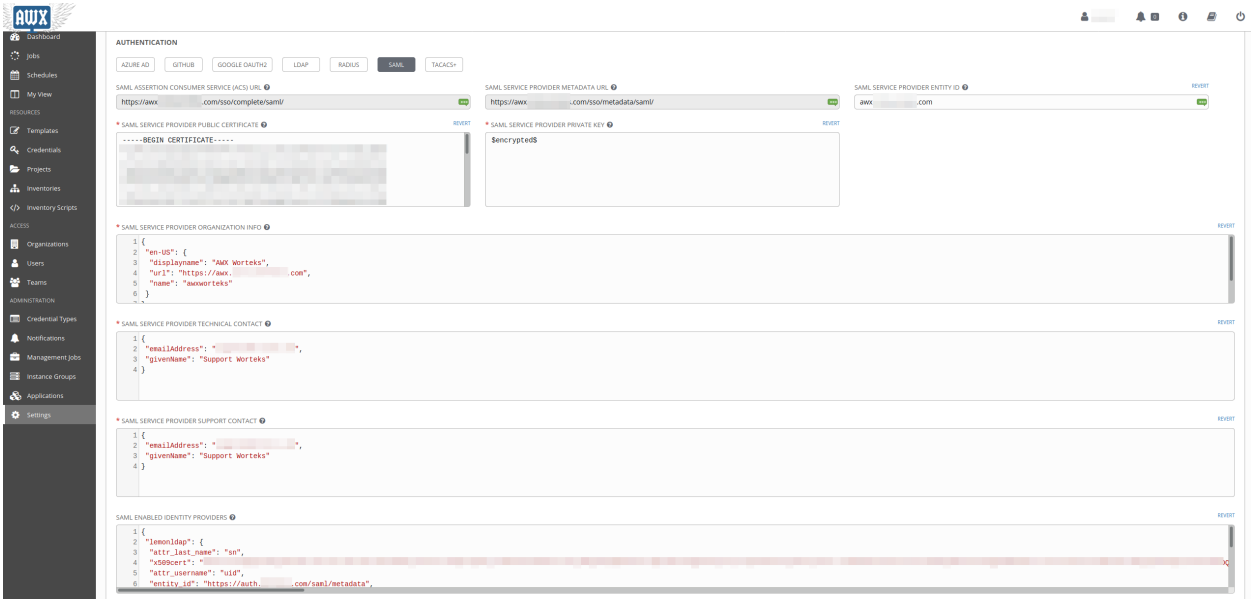
Copy it somewhere secure as lemonldap.key, and then generate the certificate with this command :

```
openssl req -new -x509 -days 3650 -key lemonldap.key > lemonldap.crt
```

After that, if you want, you can replace your SAML public key with this certificate in LLNG configuration, this is not mandatory.

AWX

You'll need an administrator account, then go to Settings -> Authentication -> SAML



There is a few settings :

SAML Service Provider Entity ID

This is the entityID for awx, lets put the fqdn : `awx.example.com`

SAML Service Provider Public Certificate

Put the content of `saml-awx.crt`

```
-----BEGIN CERTIFICATE-----
cert
-----END CERTIFICATE-----
```

SAML Service Provider Private Key

Put the content of `saml-awx.key`

```
-----BEGIN RSA PRIVATE KEY-----
key
-----END RSA PRIVATE KEY-----
```

It will be replaced with `$encrypted$` after you save the settings.

SAML Service Provider Organization Info

Organization Info for The SP, this is purely "for looks"

```
{
 "en-US": {
  "displayname": "AWX ACME",
  "url": "https://awx.example.com",
  "name": "awxacme"
 }
}
```

SAML Service Provider Technical Contact

Technical Contact for the SP

```
{
 "emailAddress": "support@example.com",
 "givenName": "Support ACME"
}
```

SAML Service Provider Support Contact

Support Contact for the SP

```
{
 "emailAddress": "support@example.com",
 "givenName": "Support ACME"
}
```

SAML Enabled Identity Providers

This is the configuration of the IdP :

```
{
 "lemonldap": {
  "attr_last_name": "sn",
  "x509cert": "SOXGp.....",
  "attr_username": "uid",
  "entity_id": "https://auth.example.com/saml/metadata",
  "attr_first_name": "givenName",
  "attr_email": "mail",
  "attr_user_permanent_id": "uid",
  "url": "https://auth.example.com/saml/singleSignOn"
 }
}
```

- "attr_last_name": "sn" SAML Attribute for the user last name
- "x509cert": "SOXGp....." the content of `lemonldap.crt` generated in the "LLNG SAML Certificate" section
- "attr_username": "uid" SAML Attribute for the user username
- "entity_id": "https://auth.example.com/saml/metadata" entityID of the IdP
- "attr_first_name": "givenName" SAML Attribute for the user first name
- "attr_email": "mail" SAML Attribute user for the user email
- "attr_user_permanent_id": "uid" SAML Attribute for the user unique id inside AWX
- "url": "https://auth.example.com/saml/singleSignOn" SAML SSO Url

SAML Security Config

```
{
   "requestedAuthnContext": false,
   "authnRequestsSigned": true
}
```

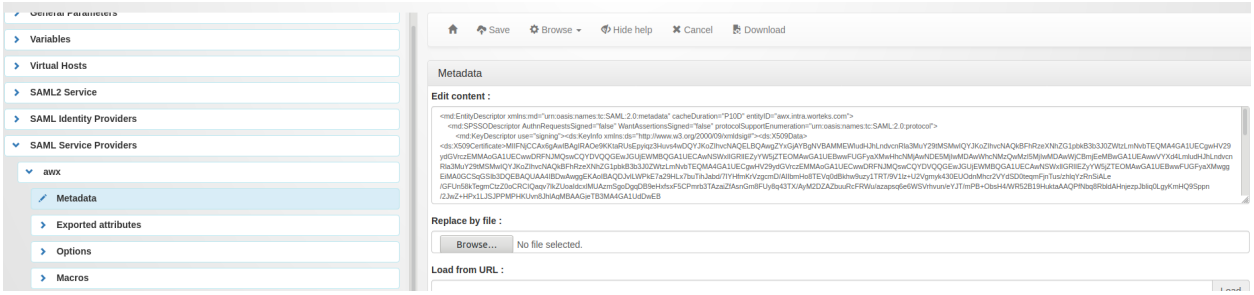
Save your configuration.

LemonLDAP:NG

We now have to define a service provider in LL:NG.

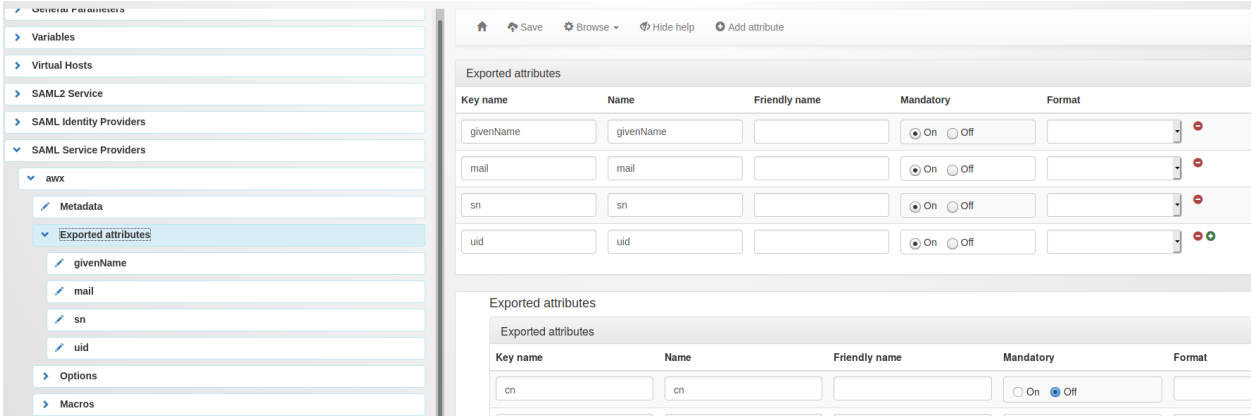
Go to "SAML service providers", click on "Add SAML SP" and name it as you want (example : 'AWX')

In the new subtree 'AWX', open 'Metadata' and paste the content of the AWX Metadatas, wich can be found at the `SAML Service Provider Metadata URL` in AWX : https://awx.example.com/sso/metadata/saml/



Now go in "Exported attributes" and add, the 'uid', 'sn', 'givenName', 'mail'.

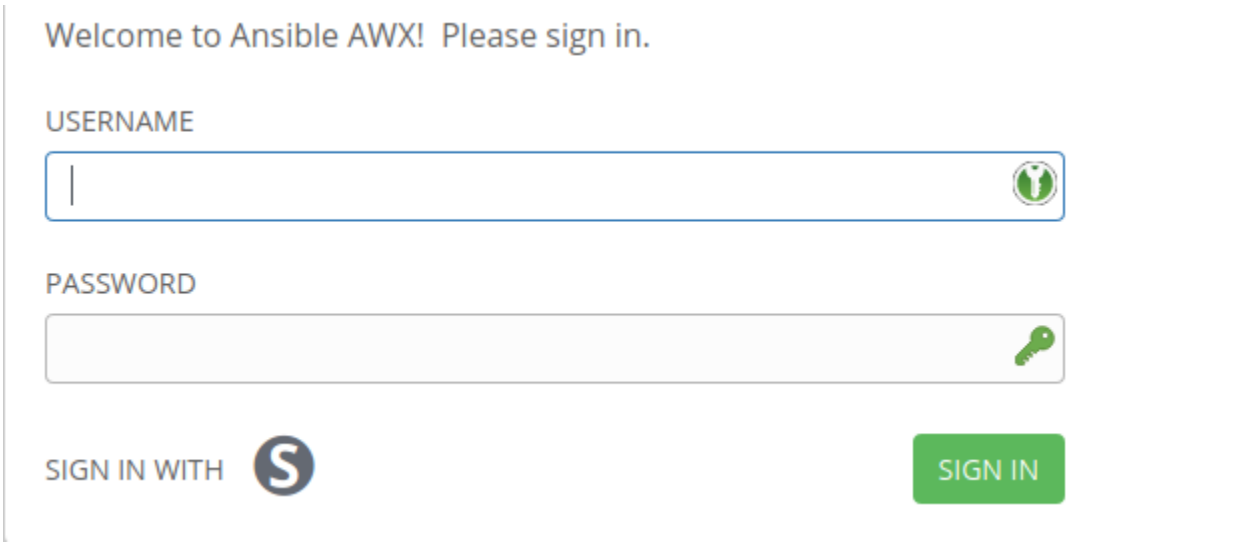
All four attributes are mandatory for AWX. Make sure they match the names of the attributes available in your LemonLDAP sessions.



Don't forget to save your configuration.

You are now good to go, and you can add the application in *your menu* and *your virtual hosts*.

You should now have a SAML button on the login page :



15.5 Bugzilla

15.5.1 Presentation

Bugzilla is server software designed to help you manage software development.

Bugzilla can authenticate a user with HTTP headers, and auto-create its account with a few information:

- User ID
- Email
- Real name

15.5.2 Configuration

Bugzilla administration

In Bugzilla administration interface, go in `Parameters` » `User authentication`

Then set:

- **auth_env_id**: HTTP_AUTH_USER
- **auth_env_email**: HTTP_AUTH_MAIL
- **auth_env_realname**: HTTP_AUTH_CN
- **user_info_class**: Env or Env,CGI

Bugzilla virtual host

Configure Bugzilla virtual host like other *protected virtual host*.

- For Apache:

```
<VirtualHost *:80>
       ServerName bugzilla.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name bugzilla.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Bugzilla virtual host in Manager

Go to the Manager and *create a new virtual host* for Bugzilla.

Configure the *rules*.

Configure the following *header*.

- **Auth-User**: $uid
- **Auth-Mail**: $mail
- **Auth-Cn**: $cn

15.6 BigBlueButton

15.6.1 Presentation

BigBlueButton is a web conferencing system designed for online learning. It offers audio/video sharing, presentations with extended whiteboard capabilities - such as a pointer, zooming and drawing - public and private chat, breakout rooms, screen sharing, integrated VoIP using FreeSWITCH, and support for presentation of PDF documents and Microsoft Office documents.

Its user-facing interface, *Greenlight*, can be configured to authenticate users with *OpenID Connect* since version 2.7.17.

15.6.2 Configuration

LL:NG

Make sure you have already *enabled OpenID Connect* on your LemonLDAP::NG server

Make sure you have generated a set of signing keys in `OpenID Connect Service` » `Security` » `Keys`

You also need to set a Signing key ID to a non-empty value of your choice.

Then, add a Relaying Party with the following configuration

- Options » Authentification » Client ID : choose a client ID, such as `my_client_id`
- Options » Authentification » Client Secret : choose a client secret, such as `my_client_secret`
- Options » Allowed redirection address : `https://my_greenlight_server/b/auth/openid_connect/callback`
- Options » ID Token Signature Algorithm : `RS256`
- Adjust your Exported Attributes to send the correct session variables in the `email` and `name` claims.

Greenlight

Configure the following environment variables in your greenlight *.env* file

```
OPENID_CONNECT_CLIENT_ID=my_client_id
OPENID_CONNECT_CLIENT_SECRET=my_client_secret
OPENID_CONNECT_ISSUER=https://auth.example.com
OPENID_CONNECT_UID_FIELD=sub
OAUTH2_REDIRECT=https://my_greenlight_server/b/
```

Notes

- Your ID Token Signature Algorithm has to be RSxxx, symmetric algorithms seem broken as of Greenlight 2.7.17
- `OAUTH2_REDIRECT` must match the URL you use to access Greenlight. the `auth/openid_connect/callback` suffix must be omitted
- Greenlight requires your LemonLDAP::NG server to be served over HTTPS using a publically recognized certificate authority (such as Let's Encrypt)

15.7 Cornerstone On Demand

15.7.1 Presentation

CornerStone On Demand (CSOD) allows one to use SAML to authenticate users. It works by default with IDP intiated mechanism, but can works with the standard SP initiated cinematic.

To work with LL::NG it requires:

- An enterprise account
- LL::NG configured as *SAML Identity Provider*
- Registered users on CSOD with the same email than those used by LL::NG (email will be the NameID exchanged between CSOD and LL::NG)

15.7.2 Configuration

New Service Provider

You should have configured LL::NG as an *SAML Identity Provider*,

Now we will add CSOD as a new SAML Service Provider:

1. In Manager, click on SAML service providers and the button `New service provider`.
2. Set csod as Service Provider name.
3. Set `Email` in `Options` » `Authentication Response` » `Default NameID format`
4. Select `Metadata`, and unprotect the field to paste the following value:

```
<md:EntityDescriptor entityID="mycompanyid.csod.com" xmlns="urn:oasis:names:tc:SAML:2.
↪0:metadata" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:md=
↪"urn:oasis:names:tc:SAML:2.0:metadata">
  <SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <KeyDescriptor use="signing">
```

(continues on next page)

(continued from previous page)

```
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
     <ds:X509Data>
      <ds:X509Certificate>
Base64 encoded CSOD certificate
        </ds:X509Certificate>
      </ds:X509Data>
      </ds:KeyInfo>
    </KeyDescriptor>
    <AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"␣
↪Location="https://mycompanyid.csod.com/samldefault.aspx" index="1" />
    <NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</NameIDFormat>
  </SPSSODescriptor>
</md:EntityDescriptor>
```

Attention: Change **mycompanyid** (in `AssertionConsumerService` markup, parameter `Location`) into your CSOD company ID and put the certificate value inside the ds:X509Certificate markup

CSOD control panel

CSOD needs two things to configure LL::NG as an IDP:

- Certificate
- SAML assertion

Certificate

See *SAML security parameters* to know how generate a certificate from you SAML private key.

SAML assertion

You need to use the IDP initiated feature of LL::NG. Just call this URL:

```
https://auth.example.com/saml/singleSignOn?IDPInitiated=1&sp=mycompanyid.csod.com
```

15.8 Discourse

15.8.1 Presentation

Discourse is a conversation-oriented forum engine

Discourse supports its own Single-Sign-On scheme but is also compatible with standard protocols such as SAML and OpenID Connect, through plugins.

This documentation illustrates the OpenID Connect plugin.

First, make sure you have set up LemonLDAP::NG 's *OpenID Connect service* and added *a Relaying Party for your Discourse instance*

Discourse can use the following OpenID Connect attributes to fill the user's profile:

```
* name
* email
* given_name
* family_name
* preferred_username
* picture
```

Make sure you create a username and password for the Relying Party, and that the discourse callback URL is allowed : https://discourse.example.com/auth/oidc/callback

15.8.2 Discourse configuration

Plugin installation

Install the Discourse OpenID Connect Plugin according to these instructions

Plugin configuration

Browse to your Discourse admin interface, and to the plugin settings

- openid_connect_enabled: *Yes*
- openid_connect_discovery_document: https://auth.example.com/.well-known/openid-configuration
- openid_connect_client_id: *Client ID you chose when configuring the Relying Party*
- openid_connect_client_secret: *Client Secret you chose when configuring the Relying Party*
- openid_connect_authorize_scope: *openid email profile*

15.9 Django

15.9.1 Presentation

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

15.9.2 Connector

The Django connector is available on GitHub: https://github.com/rclsilver/django-lemonldap

See the README to know how install and configure it.

15.10 Dokuwiki

15.10.1 Presentation

DokuWiki is a standards compliant, simple to use Wiki, mainly aimed at creating documentation of any kind. It is targeted at developer teams, workgroups and small companies. It has a simple but powerful syntax which makes sure the data files remain readable outside the Wiki and eases the creation of structured texts. All data is stored in plain text files – no database is required.

15.10.2 HTTP headers

You need to install a Dokuwiki plugin, available on Dokuwiki plugins registry: https://www.dokuwiki.org/plugin:authlemonldap

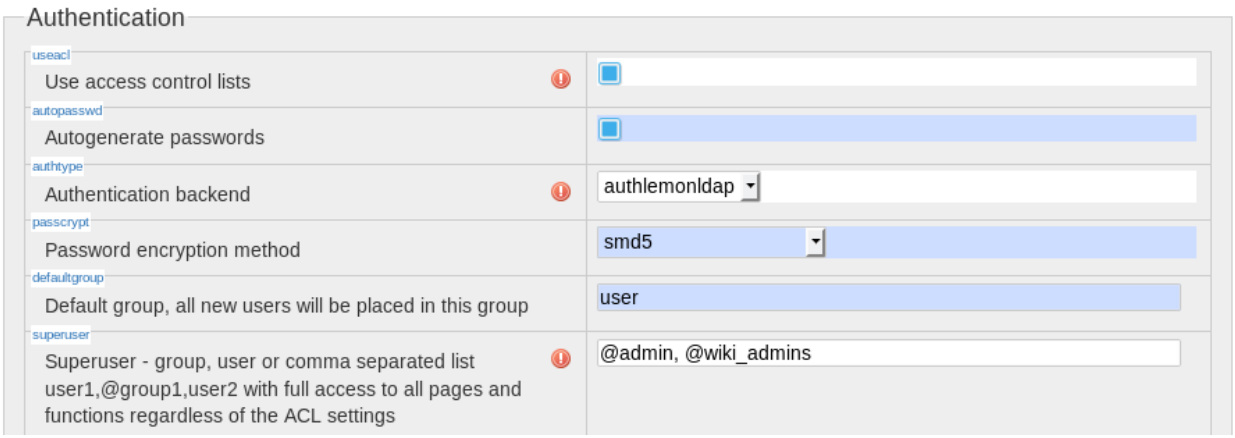
Plugin installation

Install the plugin using the Plugin Manager.

Dokuwiki configuration

As administrator, go in Dokuwiki parameters and set:

- Authentication backend: authlemonldap
- Manager: set which users and/or groups will be admin



Dokuwiki virtual host

Configure Dokuwiki virtual host like other *protected virtual host*.

- For Apache:

```
<VirtualHost *:80>
       ServerName dokuwiki.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name dokuwiki.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
```

(continues on next page)

(continued from previous page)

```
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Dokuwiki virtual host in Manager

Go to the Manager and *create a new virtual host* for Dokuwiki.

Configure the *access rules*.

Configure the *headers*:

- Auth-User $uid
- Auth-Cn: $cn
- Auth-Mail: $mail
- Auth-Groups: encode_base64($groups,””)

Attention: To allow execution of encode_base64() method, you must deactivate the *Safe jail*.

15.11 Drupal

15.11.1 Presentation

Drupal is a CMS written in PHP. It can works with external modules to extends its functionalities. One of this module can be used to delegate authentication server to the web server: Webserver Auth.

15.11.2 Installation

Install Webserver Auth module, by downloading it, and unarchive it in the drupal modules/ directory.

15.11.3 Configuration

Drupal module activation

Go on Drupal administration interface and enable the Webserver Auth module.

Drupal virtual host

Configure Drupal virtual host like other *protected virtual host*.

Attention: If you are protecting Drupal with LL::NG as reverse proxy, *convert header into REMOTE_USER environment variable*.

- For Apache:

```
<VirtualHost *:80>
       ServerName drupal.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name drupal.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
```

(continues on next page)

(continued from previous page)

```
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Drupal virtual host in Manager

Go to the Manager and *create a new virtual host* for Drupal.

Just configure the *access rules*.

If using LL::NG as reverse proxy, configure the `Auth-User` *header*, else no headers are needed.

Protect only the administration pages

With the above solution, all the Drupal site will be protected, so no anonymous access will be allowed.

Attention: You cannot use the `unprotect` rule because Drupal navigation is based on query strings (?q=admin, ?q=user, etc.), and unprotect rule only works on URL patterns.

You can create a special virtual host and use Apache rewrite module to switch between open and protected hosts:

```
<VirtualHost *:80>
    ServerName drupal.example.com

    # DocumentRoot
    DocumentRoot /var/www/html/drupal/
    DirectoryIndex index.php

    # Redirect admin pages
    RewriteEngine On
    RewriteCond  %{QUERY_STRING} q=(admin|user)
    RewriteRule ^/(.*)$ http://admindrupal.example.com/$1 [R]

    LogLevel warn
    ErrorLog /var/log/httpd/drupal-error.log
    CustomLog /var/log/httpd/drupal-access.log combined
</VirtualHost>
<VirtualHost *:80>
    ServerName admindrupal.example.com
```

(continues on next page)

(continued from previous page)

```
    # SSO protection
    PerlHeaderParserHandler Lemonldap::NG::Handler

    # DocumentRoot
    DocumentRoot /var/www/html/drupal/
    DirectoryIndex index.php

    LogLevel warn
    ErrorLog /var/log/httpd/admindrupal-error.log
    CustomLog /var/log/httpd/admindrupal-access.log combined
</VirtualHost>
```

15.12 FusionDirectory

15.12.1 Presentation

FusionDirectory provides a solution to daily management of data stored in an LDAP directory.

15.12.2 Configuration

FusionDirectory

Go in Configuration and in Login and Session panel. Set:

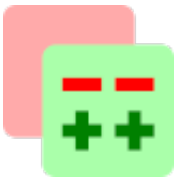
- **HTTP Header authentication**: Activate
- **Header name**: Auth-User

See also https://documentation.fusiondirectory.org/en/documentation/admin_installation/core_configuration#login-and-session

LL::NG

Just set the `Auth-User` header with the attribute that carries the user login, for example `$uid`.

15.13 Gerrit



15.13.1 Presentation

Gerrit allows to review commits before they are integrated into a target branch.

With the OAuth2 provider plugin Gerrit can use OAuth2 protocol for authentication.

15.13.2 Configuration

15.13.3 Gerrit

Install the OAuth Provider plugin. A prebuilt package of the plugin can be found on the Gerrit CI.

Then, configure Gerrit:

In `/var/gerrit/etc/gerrit.config`

```
...
[auth]
  type = OAUTH
  gitBasicAuthPolicy = HTTP
...
[plugin "gerrit-oauth-provider-lemonldap-oauth"]
  root-url = https://auth.<LLNG_SERVER>
  client-id = <GERRIT_CLIENT_ID>
```

In `/var/gerrit/etc/secret.config`

```
...
[plugin "gerrit-oauth-provider-lemonldap-oauth"]
  client-secret = <GERRIT_CLIENT_SECRET>
```

15.13.4 LL::NG

Add an Open ID Connect Relying Party for Gerrit

```
# Exported attributes (the values must fit your LDAP schema)
lemonldap-ng-cli -yes 1 \
   addKey \
     oidcRPMetaDataExportedVars/gerrit preferred_username uid \
     oidcRPMetaDataExportedVars/gerrit name cn \
     oidcRPMetaDataExportedVars/gerrit email mail \
     oidcRPMetaDataExportedVars/gerrit sub email

# Options > Basic > Allowed redirection addresses for login
#         > Logout > Allowed redirection addresses for logout
lemonldap-ng-cli -yes 1 \
   addKey \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsRedirectUris 'http://<GERRIT_
↪SERVER>/oauth' \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsPostLogoutRedirectUris 'https://
↪<GERRIT_SERVER>/'

# Options > Basic > Client ID
#         > Basic > Client Secret
lemonldap-ng-cli -yes 1 \
   addKey \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsClientID '<GERRIT_OAUTH_ID>' \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsClientSecret '<GERRIT_OAUTH_
↪SECRET>'

# Timeout > ID Token expiration
#         > Access Token expiration
# Security > ID Token signature algorithm
lemonldap-ng-cli -yes 1 \
   addKey \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsIDTokenExpiration 3600 \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsAccessTokenExpiration 3600 \
     oidcRPMetaDataOptions/gerrit oidcRPMetaDataOptionsIDTokenSignAlg RS512
```

15.14 Gitlab

15.14.1 Presentation

See Gitlab page for product presentation.

Gitlab allows one to use SAML to authenticate users, see official documentation

15.14.2 SAML

For this example, we use these sample values:

- Gitlab URL : https://gitlab.example.com
- LL::NG portal URL : https://auth.example.com

Gitlab configuration

Find the gitlab.rb file and add these settings:

```
vi /etc/gitlab/gitlab.rb
```

```
gitlab_rails['omniauth_enabled'] = true
gitlab_rails['omniauth_allow_single_sign_on'] = ['saml']
gitlab_rails['omniauth_auto_link_saml_user'] = true
gitlab_rails['omniauth_block_auto_created_users'] = false

gitlab_rails['omniauth_providers'] = [
  {
    name: 'saml',
    args: {
      assertion_consumer_service_url: 'https://gitlab.example.com/users/auth/saml/
↪callback',
      idp_cert_fingerprint: '99:BE:7B:68:3F:XX:7D:EF:6B:C3:XX:C0:0E:XX:D4:EA:02:XX:83:2A
↪',
      idp_sso_target_url: 'https://auth.example.com/saml/singleSignOn',
      issuer: 'https://gitlab.example.com',
      name_identifier_format: 'urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress'
    },
    label: 'Login with LL::NG' # optional label for SAML login button
  }
]
```

Tip: To get the fingerprint of IDP certificate, copy SAML certificate from LL::NG configuration in a file and use openssl:

```
openssl x509 -in CERT.pem -noout -fingerprint
```

You can force SAML by default with this option:

```
gitlab_rails['omniauth_auto_sign_in_with_provider'] = 'saml'
```

In this case, users won't be able to log directly on gitlab. Set it once you are sure the SAML configuration is valid.

To apply changes:

```
gitlab-ctl reconfigure
```

LL::NG configuration

We suppose LL::NG is configured as SAML IDP, and that you converted the public key into a certificate for SAML signature. You must enable the option to send certificates in response. If you don't want to, you need to copy the certificate value into Gitlab configuration, in `idp_cert` parameter.

You can get Gitlab SAML metadata on https://gitlab.example.com/users/auth/saml/metadata

Register them in LL::NG and send these SAML attributes:

- mail => email
- uid => uid
- cn => name

Attention: The value from LL::NG mail session attribute must be the email of the user in Gitlab database, in order to associate accounts.

Manage groups

You can pass groups to Gitlab. For this, declare groups attribute in gitlab.rb:

```
...
gitlab_rails['omniauth_providers'] = [
  {
    name: 'saml',
    groups_attribute: 'groups',
...
```

And in LL::NG, export the groups attribute:

- groups => groups

15.14.3 OpenID Connect

Alternatively to SAML, you can choose to configure Gitlab to use OpenID Connect.

Gitlab configuration

In `/etc/gitlab/gitlab.rb`

```
...
gitlab_rails['omniauth_allow_single_sign_on'] = ['openid_connect']
gitlab_rails['omniauth_block_auto_created_users'] = false

gitlab_rails['omniauth_providers'] = [
  { 'name' => 'openid_connect',
    'label' => 'LemonLDAP::NG',
```

(continues on next page)

(continued from previous page)

```
    'args' => {
      'name' => 'openid_connect',
      'issuer' => 'https://auth.example.com',
      'scope' => ['openid', 'profile', 'email'],
      'response_type' => 'code',
      'client_auth_method' => 'client_secret_post',
      'discovery' => true,
      'uid_field' => 'sub',
      'client_options' => {
        'redirect_uri' => 'http://gitlab.example.com/users/auth/openid_connect/callback',
        'identifier' => 'LEMONLDAP_CLIENT_ID',
        'secret' => 'LEMONLDAP_CLIENT_SECRET',
      }
    }
  }
];

...
```

LL::NG configuration

Add an OpenID Connect RP to LemonLDAP::NG

- Chose a client ID and a client secret, and write the same values in the `gitlab.rb` file above
- You need to chose an asymetrical signature algorithm for the ID Token (RS256 or above)
- You also need to set a key identifier on your LemonLDAP::NG server in `OpenID Connect service` » `Security` » `Signing key ID` (use something like `default` as the value).
- Make sure the attribute containing the user email in the LemonLDAP::NG session is mapped to the `email` claim.

Attention: You need to set a key identifier, or you will get a *JSON::JWK::Set::KidNotFound* error on Gitlab

15.15 GLPI

15.15.1 Presentation

GLPI is the Information Resource-Manager with an additional Administration- Interface. You can use it to build up a database with an inventory for your company (computer, software, printers...). It has enhanced functions to make the daily life for the administrators easier, like a job-tracking-system with mail-notification and methods to build a database with basic information about your network-topology.

15.15.2 Configuration

For GLPI >= 0.71, it is a simple configuration in GLPI: Setup → Authentication. In "External authentications" click "Others" and in "Field holding the login in the _SERVER array" select "REMOTE_USER"

For older version, check http://wiki.glpi-project.org/doku.php?id=en:authautoad

If you use Nginx, you need to add this in configuration:

```
proxy_set_header Host $http_host;
proxy_set_header X-Forwarded-Host $http_host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

15.16 Google Apps

15.16.1 Presentation

Google Apps can use SAML to authenticate users, behaving as an SAML service provider, as explained here.

To work with LL::NG it requires:

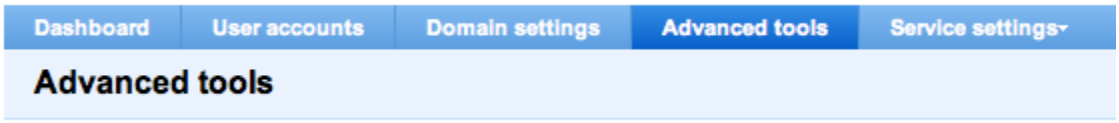
- An enterprise Google Apps account
- LL::NG configured as *SAML Identity Provider*
- Registered users on Google Apps with the same email than those used by LL::NG (email will be the NameID exchanged between Google Apps and LL::NG)

15.16.2 Configuration

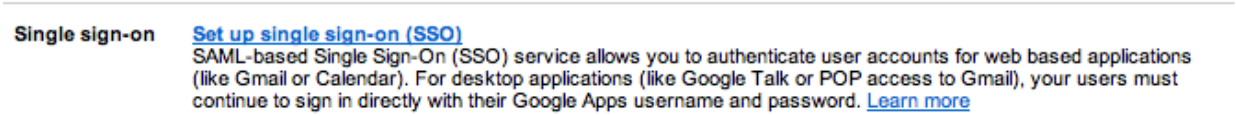
Google Apps control panel

Attention: This part is based on SimpleSAMLPHP documentation.

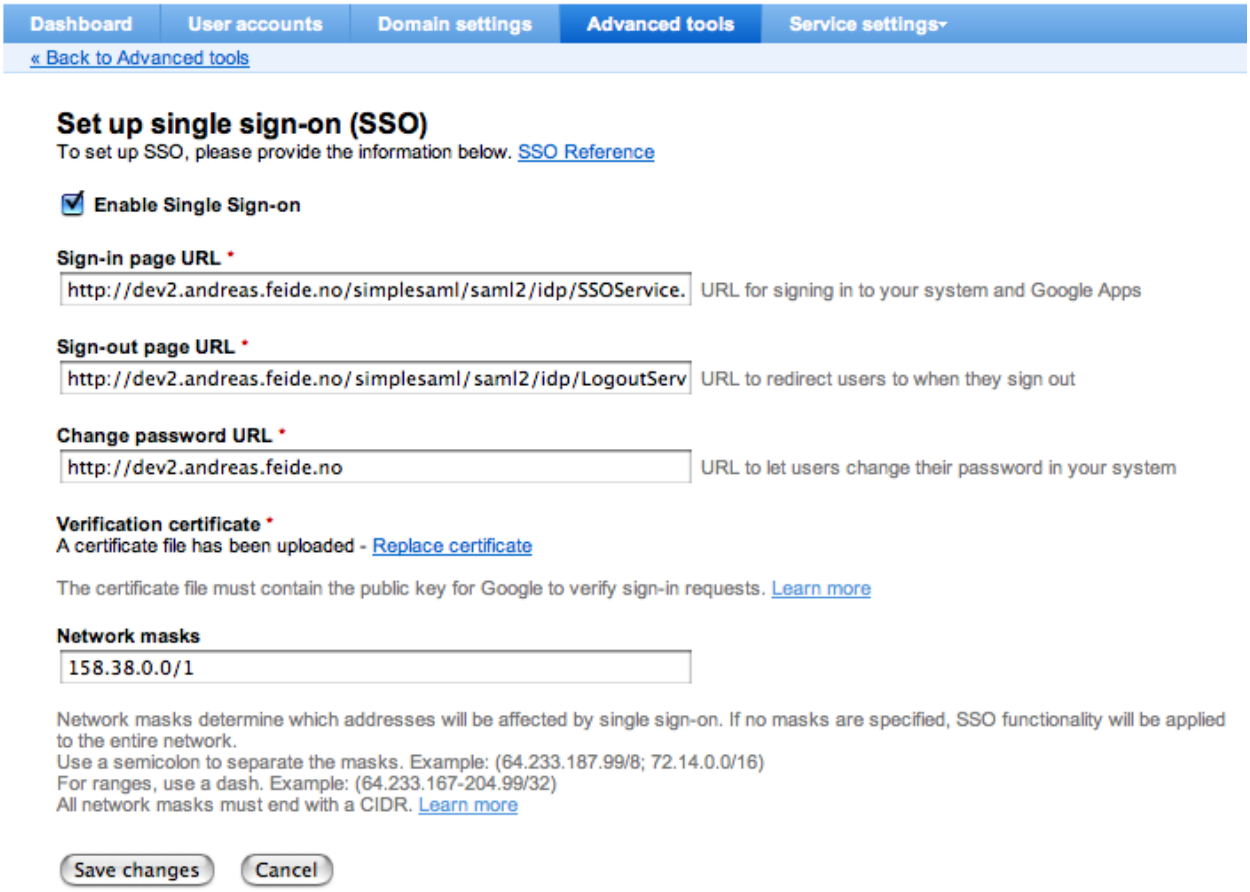
As administrator, go in Google Apps control panel and click on Advanced tools:



Then select `Set up single sign-on (SSO)`:



Now configure all SAML parameters:



- **Enable Single Sign-On**: check the box. Uncheck it to disable SAML authentication (for example, if your Identity Provider is down).
- **Sign-in page URL**: SSO access point (HTTP-Redirect binding). Example: http://auth.example.com/saml/singleSignOn
- **Sign-out page URL**: this in not the SLO access point (Google Apps does not support SLO), but the main logout page. Example: http://auth.example.com/?logout=1
- **Change password URL**: where users can change their password. Example: http://auth.example.com

Attention: You must check the option `Use a specific domain transmitter` to force Google Apps to send the full entityId.

Certificate

For the certificate, you can build it from the signing private key registered in Manager. Select the key, and export it (button `Download`). This will download the public and the private key.

Keep the private key in a file, for example lemonldap-ng-priv.key, then use openssl to generate an auto-signed certificate:

```
openssl req -new -key lemonldap-ng-priv.key -out cert.csr
openssl x509 -req -days 3650 -in cert.csr -signkey lemonldap-ng-priv.key -out cert.pem
```

You can now the upload the certificate (`cert.pem`) on Google Apps.

Tip: You can also use the certificate instead of public key in SAML metadata, see *SAML service configuration*

New Service Provider

You should have configured LL::NG as an *SAML Identity Provider*,

Now we will add Google Apps as a new SAML Service Provider:

1. In Manager, click on SAML service providers and the button `New service provider`.
2. Set GoogleApps as Service Provider name.
3. Set `Email` in `Options` » `Authentication Response` » `Default NameID format`
4. Disable all signature flags in `Options` » `Signature`, except `Sign SSO message` which should be to `On`
5. Select `Metadata`, and unprotect the field to paste the following value:

```
<md:EntityDescriptor entityID="google.com" xmlns="urn:oasis:names:tc:SAML:2.0:metadata"␣
↪xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:md="urn:oasis:names:tc:SAML:2.
↪0:metadata">
  <SPSSODescriptor protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"␣
↪Location="https://www.google.com/a/mydomain.org/acs" index="1" />
    <NameIDFormat>urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress</NameIDFormat>
  </SPSSODescriptor>
</md:EntityDescriptor>
```

Attention: Change **mydomain.org** (in `AssertionConsumerService` markup, parameter `Location`) into your Google Apps domain. Also adapt your entityID to match the Assertion issuer: google.com/a/mydomain.org

Application menu

You can add a link in *application menu* to display Google Apps to users.

You need to adapt some parameters:

- **Address**: set one of Google Apps URL (all Google Apps product a distinct URL), for example http://www.google.com/calendar/hosted/mydomain.org/render
- **Display**: As Google Apps is not a protected application, set to `On` to always display it

Attention: Change **mydomain.org** into your Google Apps domain

Logout

Google Apps does not support Single Logout (SLO).

Google Apps has a configuration parameter to redirect user on a specific URL after Google Apps logout (see *Google Apps control panel*).

To manage the other way (LL::NG → Google Apps), you can add a dedicated *logout forward rule*:

```
GoogleApps => http://www.google.com/calendar/hosted/mydomain.org/logout
```

Attention: Change **mydomain.org** into your Google Apps domain

15.17 Grafana

15.17.1 Presentation

Grafana is an Open Source dashboard for monitoring databases such as Prometheus, Graphite or Elasticsearch

Grafana offers social login through a generic OAuth 2 connector. Thankfully, it is close enough to OpenID Connect to work well with LemonLDAP::NG

15.17.2 Pre-requisites

Grafana configuration

You should start by following the generic OAuth2 documentation provided by Grafana: https://grafana.com/docs/grafana/latest/auth/generic-oauth/

Your configuration file will have to look something like this:

```
[auth.generic_oauth]
enabled = true
client_id = CHOOSE_A_CLIENT_ID
client_secret = CHOOSE_A_CLIENT_SECRET
scopes = openid email profile
auth_url = https://auth.example.com/oauth2/authorize
token_url = https://auth.example.com/oauth2/token
api_url = https://auth.example.com/oauth2/userinfo
allow_sign_up = true
name = LemonLDAP::NG
send_client_credentials_via_post = false
email_attribute_name = email
```

LL:NG

Make sure you have already *enabled OpenID Connect* on your LemonLDAP::NG server

Then, add a Relaying Party with the following configuration:

- Options » Authentification » Client ID : same as `client_id` above
- Options » Authentification » Client Secret : same as `client_secret` above
- Options » Allowed redirection address : `https://<grafana domain>/login/generic_oauth`

If you want to transmit extra user attributes to Grafana, you also need to configure:

- Extra Claims »
  - add a key named `profile`
  - set a value of `name username display_name upn`
- Exported Attributes (not all of them are mandatory)
  - replace the existing keys with the following 5 new keys:
    - `name`
    - `username`
    - `display_name`
    - `upn`
    - `email`
  - map them to your corresponding LemonLDAP::NG session attribute

Tip: To trigger OIDC authentication directly, you can register grafana in application menu and set as URL: `https://<grafana domain>/login/generic_oauth`

15.18 GRR

15.18.1 Presentation

GRR is a room booking software.

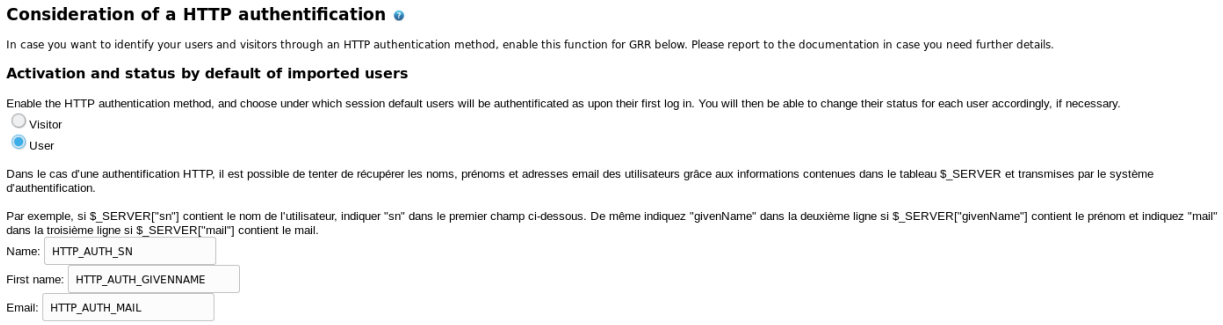
15.18.2 HTTP header

Configuration

GRR has a SSO configuration page in its administration panel.

Do not use Lemonldap mode, which is for a very old Lemonldap version, but HTTP authentication.

Set the default profile of connected users and which headers contains surname, firstname and mail.



GRR will check the username in REMOTE_USER, so use *remote header conversion* if you are in proxy mode.

GRR virtual host in LL::NG

Access rules:

- ^/index.php => accept
- default => unprotect

Headers:

- Auth-User $uid
- Auth-Sn: $sn
- Auth-GivenName: $givenName
- Auth-Mail: $mail

15.19 Guacamole

15.19.1 Presentation

Apache Guacamole is a web-based remote desktop gateway. It supports standard protocols like VNC, RDP, and SSH.

As of version 0.9.14, Guacamole can use *OpenID Connect* , *CAS* or *HTTP Headers* as authentication sources through plug-ins.

This document explains how to implement OpenID Connect

15.19.2 Pre-requisites

Guacamole

Refer to the official Guacamole documentation to install Guacamole, either manually or through Docker images

You need to be able to enable extensions. If you are using docker, you need to follow these instructions in order to provide your own extensions directory and Guacamole configuration file

Your Guacamole configuration directory will look something like this.

```
├── extensions
│   └── 00-guacamole-auth-openid-1.0.0.jar
└── guacamole.properties
```

Danger: Make sure to rename the JAR in a way that ensures that it will be loaded first

And `guacamole.properties` should contain at least

```
openid-authorization-endpoint: http://auth.example.com/oauth2/authorize
openid-jwks-endpoint: http://auth.example.com/oauth2/jwks
openid-issuer: http://auth.example.com
openid-client-id: guacamole
openid-redirect-uri: http://guacamole.example.com/guacamole/
openid-username-claim-type: sub
```

Tip: Remplace the `redirect uri` with your Guacamole server's URL

LL:NG

Make sure you have already *enabled OpenID Connect* on your LemonLDAP::NG server

You also need to allow the `Implicit Flow` under `OpenID Connect Service` » `Security`

Then, add a Relaying Party with the following configuration

- Options » Authentification » Client ID : same as `openid-client-id` in `guacamole.properties`
- Options » Allowed redirection address : same as `openid-redirect-uri` in `guacamole.properties`
- Options » ID Token Signature Algorithm : `RS512`

15.20 HumHub

15.20.1 Présentation

HumHub is a free and open-source social network written on top of the Yii2 PHP framework that provides an easy to use toolkit for creating and launching your own social network.

Unauthenticated users may connect using a login form against HumHub local database or a LDAP directory, or choose which authentication service they want to use.

Administrator can configure one or several OAuth, OAuth2 or OIDC authentication services to be displayed as buttons on the login page.

With *OpenID Connect* authentication service, users successfully authenticated by LemonLDAP::NG will be registered in HumHub upon their first login.

Danger: HumHub retrieves a user from his username and the authentication service he came through. As a result, a former local or LDAP user will be rejected when trying to authenticate using another authentication service. See *Migrate former local or ldap Humhub account to connect through SSO*

15.20.2 OpenID Connect

Note: This set-up works with option enablePrettyUrl activated in Humhub. If not activated, rewrite URL in Humhub HTTP server and allowed redirect URL in LemonLDAP needs to be adapted to work with the non pretty URL format.

Configuring HumHub

First disable LDAP (Administration > Users section) and delete (or *migrate*) any local users whose username or email are conflicting with the username or email of your OIDC users.

Then install and configure the OIDC connector for humhub extension using composer :

- Install composer.
- Consider using prestissimo, to speed up composer update command (4x faster):

```
composer global require hirak/prestissimo
```

- Go to {humhub_home} folder
- Check if composer.json file is present. If not, download it for your current version:

```
wget https://raw.githubusercontent.com/humhub/humhub/v1.3.15/composer.json
```

- Install the connector as a dependency:

```
composer require --no-update --update-no-dev worteks/humhub-auth-oidc
composer update worteks/humhub-auth-oidc  --no-dev --prefer-dist -vvv
```

Note: If you just need to update the connector, change its version in composer.json and run the above composer update command.

- Edit *{humhub_home}/protected/config/common.php* with the client configuration :

```
'components' => [
  'authClientCollection' => [
    'clients' => [
      // ...
      'lemonldapng' => [
        'class' => 'worteks\humhub\authclient\OIDC',
        'domain' => 'https://auth.example.com',
        'clientId' => 'myClientId', // Client ID for this RP in LemonLDAP
        'clientSecret' => 'myClientSecret', // Client secret for this RP in LemonLDAP
        'defaultTitle' => 'auth.example.com', // Text displayed in login button
        'cssIcon' => 'fa fa-lemon-o', // Icon displayed in login button
      ],
    ],
    // ...
]
```

- Edit {humhub_home}/protected/config/web.php to disconnect users from LemonLDAP::NG after they logged out of Humhub:

```
return [
 // ...
 'modules' => [
  'user' => [
   'logoutUrl' => 'https://auth.domain.com/?logout=1',
  ],
```

(continues on next page)

(continued from previous page)

```
 ]
];
```

User can now log in through SSO using a button on humhub logging page. If you want to remove this intermediate login page, so user are automatically logged in through SSO when they first access Humhub, you can set up a redirection in the http server in front of the application :

- Example in apache

```
RewriteEngine On
RewriteCond %{QUERY_STRING} !nosso [NC]
RewriteRule "^/user/auth/login$" "/user/auth/external?authclient=lemonldapng" [L,R=301]
```

- Example in nginx

```
if ($query_string !~ "nosso"){
  rewrite ^/user/auth/login$ /user/auth/external?authclient=lemonldapng permanent;
}
```

If the authentication was successful but the user could not be registered in Humhub (which often happen if there is a conflict between source, username or email), Humhub will redirect to the login page to display the error, which trigger a redirection to the portal, ultimately triggering a loop error while registration error is not displayed.

To change this behavior and display the registration error, AuthController.onAuthSuccess method needs to be adapted so redirect to SSO will be bypassed when a registration error occured. This works for version 1.3.15 :

- Go to {humhub_home} folder
- Execute

```
sed -i "s|return \$this->redirect(\['/user/auth/login'\]);|return \$this->redirect(['/
↪user/auth/login','nosso'=>'showerror']);|" protected/humhub/modules/user/controllers/
↪AuthController.php
```

Configuring LemonLDAP

If not done yet, configure LemonLDAP::NG as an *OpenID Connect service*.

Then, configure LemonLDAP::NG to recognize your HumHub instance as a valid *new OpenID Connect Relying Party* using the following parameters:

- **Client ID**: the same you set in HumHub configuration
- **Client Secret**: the same you set in HumHub configuration
- **Add the following exported attributes**
  - **given_name**: user's givenName attribute
  - **family_name**: user's sn attribute
  - **email**: user's mail attribute
- **Redirect URIs** containing your Yii2 auth client ID.

Configuration sample using CLI:

```
$ /usr/libexec/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
      oidcRPMetaDataExportedVars/humhub given_name givenName \
      oidcRPMetaDataExportedVars/humhub family_name sn \
      oidcRPMetaDataExportedVars/humhub email mail \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsClientID myClientId \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsClientSecret myClientSecret \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsRedirectUris 'https://humhub.
↪example.com/user/auth/external?authclient=lemonldapng'  \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsPostLogoutRedirectUris 'https://
↪humhub.example.com' \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsIDTokenSignAlg RS512 \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsIDTokenExpiration 3600 \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsAccessTokenExpiration 3600 \
      oidcRPMetaDataOptions/humhub oidcRPMetaDataOptionsBypassConsent 1 && \
```

Migrate former local or ldap Humhub account to connect through SSO

You need to manually update Humhub database to swith authentication mode to LemonLDAP::NG.

Table “user”:

- Columns “username” and “email” should match exactly OIDC sub and email attributes ;
- If former ldap user, change column “auth_mode” to “local”.

Table “user_auth”:

- Add an entry with user_id, username and “lemonldapng” as source (or the name you chose in your connector configuration) :

```
+---------+-------------+-------------+
| user_id | source      | source_id   |
+---------+-------------+-------------+
|       4 | lemonldapng | jdoe        |
```

Troubleshooting

If LemonLDAP login page freezes because of a browser security blockage, adapt security’s CSP Form Action to allow HumHub host :

```
$ /usr/libexec/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
     set \
       cspFormAction "'self' https://*.example.com"
```

15.21 i-Parapheur

15.21.1 Presentation

i-Parapheur is a web application allowing digital signature on documents. It was built around Alfresco.

It can use external authentication based on HTTP header.

15.21.2 Configuration

On i-Parapheur

Edit `/opt/iParapheur/tomcat/shared/classes/alfresco-global.properties` and add:

```
parapheur.auth.external.header.authorize=true
```

Edit `/opt/iParapheur/tomcat/shared/classes/iparapheur-global.properties` and add:

```
parapheur.auth.external.header.name=Auth-User
parapheur.auth.external.header.regexp=.*
```

On LemonLDAP::NG

Go to the Manager and *create a new virtual host* for iParapheur.

Just configure the *access rules*.

Create the `Auth-User` *header* to send the user login to iParapheur.

15.22 Jitsi Meet

15.22.1 Presentation

Jitsi Meet is a WEBRTC-based video conferencing application, powering the meet.jit.si online service.

Users may install their own instance of Jitsi Meet for private use, in which case, they may use authentication to control the creation of conference rooms.

The official documentation provides instructions on how to configure Jitsi Meet to use Shibboleth, but with a little adaptation, it can work just as fine with LemonLDAP::NG.

15.22.2 Configuration

Pre-requisites

This documentation assumes that you have already installed a *Nginx-based* LemonLDAP::NG Handler on your Jitsi server.

You need to install Nginx before Jitsi Meet. If you install Jitsi Meet first, the Jitsi Meet installer will not generate a Nginx configuration file.

We assume that you have followed the Jitsi Meet quick start

Jitsi Meet configuration

As with the Shibboleth guide, you need to configure `/etc/jitsi/jicofo/sip-communicator.properties`

```
org.jitsi.jicofo.auth.URL=shibboleth:default
org.jitsi.jicofo.auth.LOGOUT_URL=/logout/
```

This defines the login servlet as `/login/` and the logout URL as `/logout/`

Jitsi Meet Nginx configuration

In the Nginx configuration that the Jitsi Meet quickstart generated, you must add the following blocks, just like you would in a typical handler configuration file:

```
# This block lets Nginx know how to contact the local LLNG handler
# for authentication
location = /lmauth {
   internal;
   include /etc/nginx/fastcgi_params;
   fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
   fastcgi_pass_request_body  off;
   fastcgi_param CONTENT_LENGTH "";
   fastcgi_param HOST $http_host;
   fastcgi_param X_ORIGINAL_URI  $original_uri;
}

# Protect only the /login/ URL
# You may want to change this is your goal is to make the whole Jitsi Meet instance␣
↪private
location /login/ {
```

(continues on next page)

(continued from previous page)

```
    # Protect the current path with LLNG
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;

    # Transmis user information to Jitsi through HTTP headers
    auth_request_set $mail $upstream_http_mail;
    proxy_set_header mail $mail;
    auth_request_set $displayname $upstream_http_displayName;
    proxy_set_header displayName $displayname;
    auth_request_set $lmcookie $upstream_http_cookie;
    proxy_set_header Cookie: $lmcookie;

    # Proxy requests to Jitsi Meet
    proxy_pass http://127.0.0.1:8888/login;
}
```

Warning: Thoses 2 blocks should be append BEFORE the following block:

```
#Anything that didn't match above, and isn't a real file,
#assume it's a room name and redirect to /
location ~ ^/([^/?&:'"]+)/(.*)$ {
   set $subdomain "$1.";
   set $subdir "$1/";
   rewrite ^/([^/?&:'"]+)/(.*)$ /$2;
}
```

Jitsi Meet Virtual host in Manager

Go to the Manager and *create a new virtual host* for Jitsi Meet.

Configure the *access rules*.

- Don't forget to configure the /logout/ URL

Configure the following *headers*.

- **mail**: $mail
- **displayName**: $cn

Danger: Jitsi meet expects to find a `mail` HTTP header, it will ignore REMOTE_USER and only use the mail value to identify the user.

15.23 Liferay

15.23.1 Presentation

Liferay is an enterprise portal.

Liferay can use LL::NG as an SSO provider but you have to manage how users are created:

- By hand in Liferay administration screens
- Imported from an LDAP directory

Of course, integration will be full if you use the LDAP directory as users backend for LL::NG and Liferay.

> **Attention:** If the user is not created, or can not be created via LDAP import, the connection to Liferay will be refused. With LDAP, login, mail, first name and last name are required attributes. If one is missing, the user is not created.

This documentation just explains how to set up the SSO part. Please refer to Liferay documentation to enable LDAP provisionning.

15.23.2 Configuration

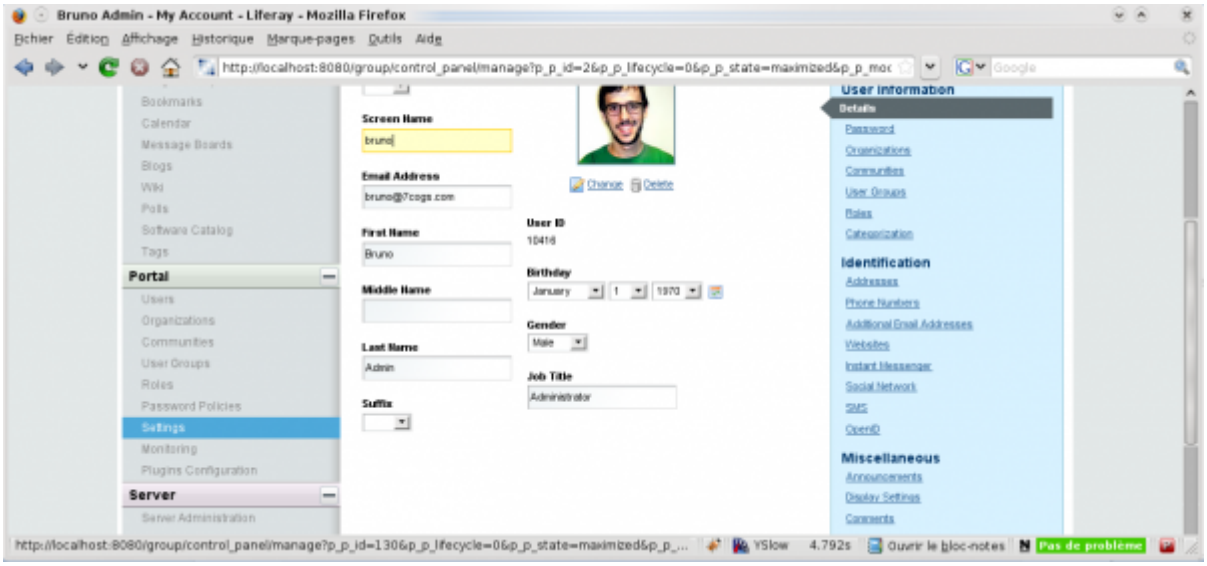
Liferay administration

Access to Liferay (first time):

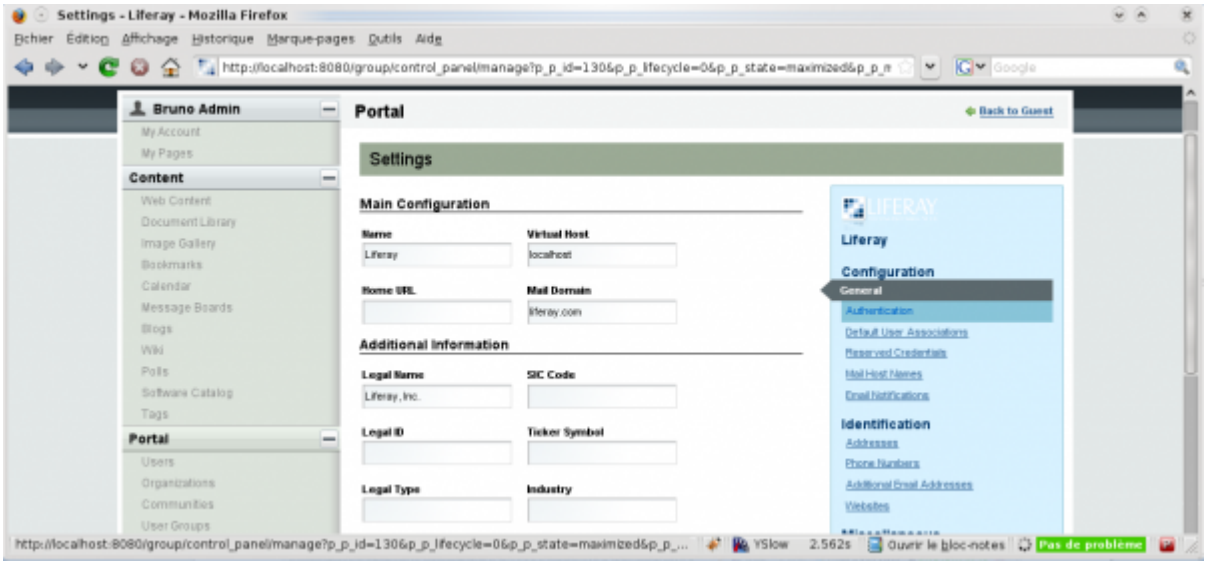
Login as administrator:

Go to `My Account`:

Go to `Portal` » `Settings`:



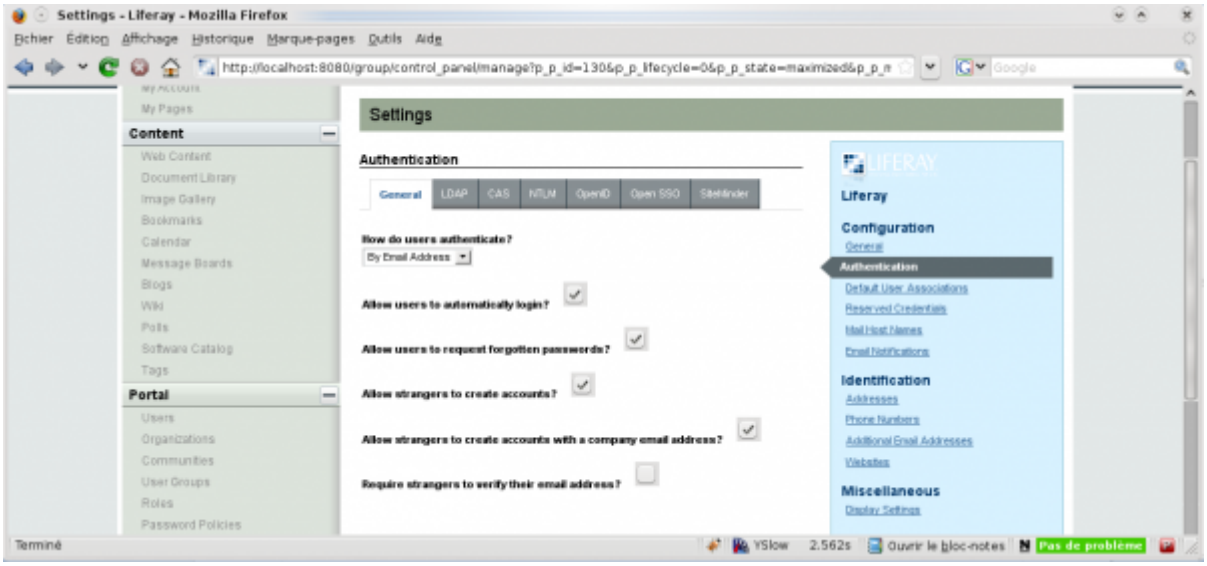
Go to `Configuration` » `Authentication`:



In `General`, fill at least the following information:

- **How do users authenticate?**: by login

Tip: We advice to deactivate other options, cause users will use LL::NG portal to modify or reset their password.

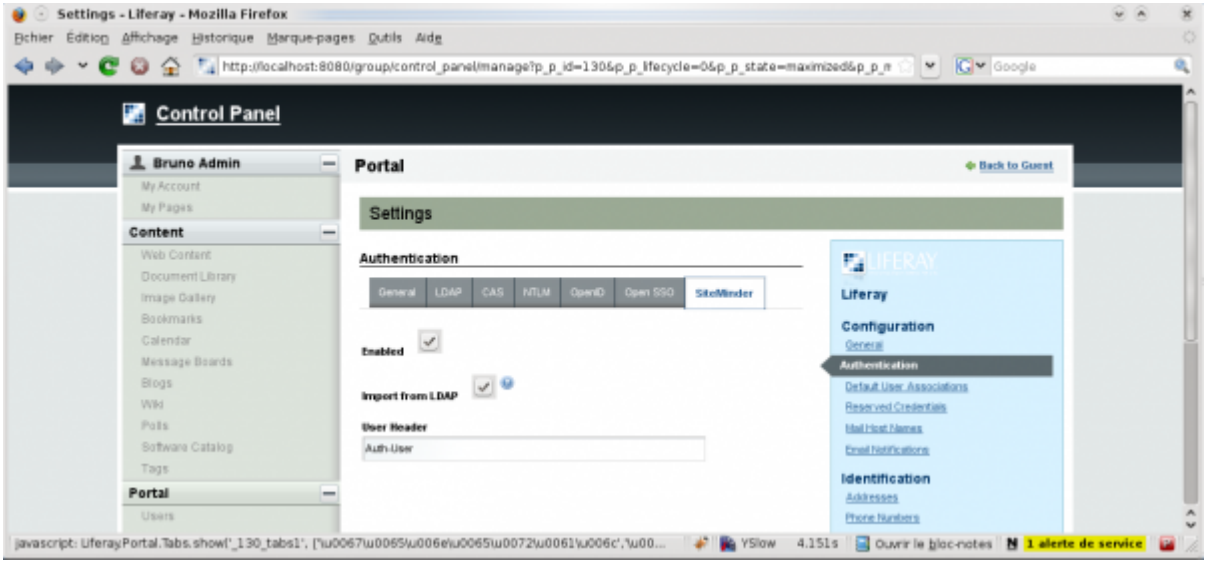


Attention: You need to activate LDAP authentication, else SSO authentication will not work. Do this in the control panel or in the configuration file:

```
ldap.auth.enabled=true
```

Then use the `SiteMinder` tab to configure SSO:

- **Enabled**: Yes
- **Import from LDAP**: Yes (see *presentation*)
- **User Header**: Auth-User (case sensitive)



Attention: Do not forget to save your changes!

Liferay virtual host

Configure Liferay virtual host like other *protected virtual host*.

- For Apache:

```
<VirtualHost *:80>
       ServerName liferay.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name liferay.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Liferay virtual host in Manager

Go to the Manager and *create a new virtual host* for Liferay.

Just configure the *access rules*. You can add a rule for logout:

```
^/c/portal/logout => logout_sso
```

Configure the `Auth-User` *header*.

15.24 LimeSurvey

15.24.1 Presentation

LimeSurvey is a web survey software written in PHP.

15.24.2 HTTP Headers

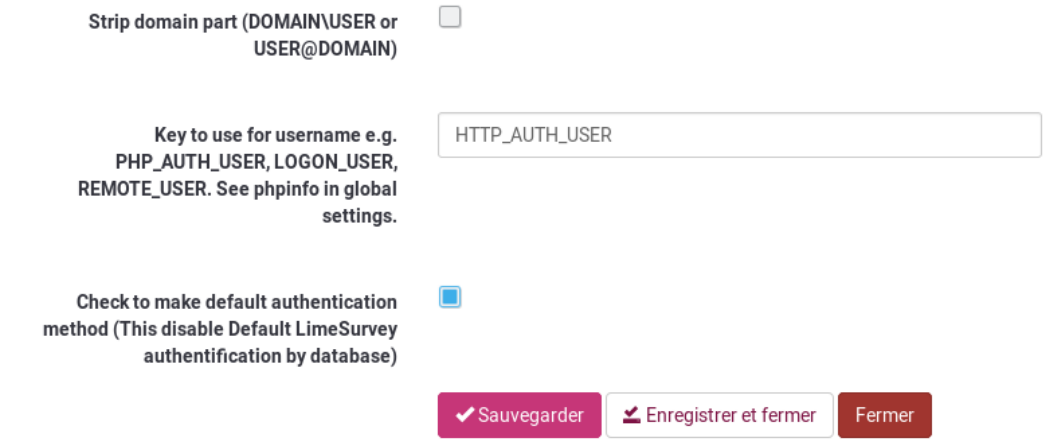
LimeSurvey has a webserver authentication mode that allows one to integrate it directly into LemonLDAP::NG.

To have a stronger integration, we will configure LimeSurvey to autocreate unknown users and use HTTP headers to fill name and mail.

> **Attention:** We suppose that LimeSurvey is installed in /var/www/html/limesurvey

LimeSurvey configuration

In Administration panel, go in Configuration > Parameters > Extensions manager. Select the WebServer module and configure it.



This is enough for the authentication part.

Tip: If you are blocked, you can deactivate the plugin with this request in database:

```
update lime_plugins SET active=0 where name="Authwebserver";
```

To configure account autocreation, you need to edit application/config/config.php: The configuration is done in config.php:

```
vi /var/www/html/limesurvey/application/config/config.php
```

```
'config'=>array(
// debug: Set this to 1 if you are looking for errors. If you still get no errors after␣
↪enabling this
// then please check your error-logs - either in your hosting provider admin panel or in␣
↪some /logs directory
// on your webspace.
// LimeSurvey developers: Set this to 2 to additionally display STRICT PHP error␣
↪messages and get full access to standard templates
        'debug'=>0,
        'debugsql'=>0, // Set this to 1 to enanble sql logging, only active when debug =␣
↪2
        // Update default LimeSurvey config here
        'auth_webserver_autocreate_user' => true,
        'auth_webserver_autocreate_profile' => Array('full_name' => $_SERVER['HTTP_AUTH_
↪CN'],'email' => $_SERVER['HTTP_AUTH_MAIL'],'lang'=>'en'),
        'auth_webserver_autocreate_permissions' => Array('surveys' => array('create'=>
↪true,'read'=>false,'update'=>false,'delete'=>false)),
        )
```

See also https://manual.limesurvey.org/Optional_settings#Authentication_delegation_with_automatic_user_import

LimeSurvey virtual host

Configure LimeSurvey virtual host like other *protected virtual host*.

LimeSurvey virtual host in Manager

Go to the Manager and *create a new virtual host* for LimeSurvey.

Headers

Header name	Description
Auth-User	user login
Auth-Cn	user full name
Auth-Mail	user email

Rules

Rule name	Expression	Description
Logout	/sa/logout$	Logout rule (for example logout_app_sso)
Admin		Allow only admin and superadmin users
Default	default	Allow only users with a LimeSurvey role

Tip: You can set the default access to:

- **accept**: all authenticated users will access surveys
- **unprotect**: no authentication will be asked to access surveys

---

15.25 Mattermost Team Edition

15.25.1 Presentation

Mattermost is a team-based instant messaging application.

See the official Mattermost website for a complete presentation.

Mattermost follows an Open Core development model. The freely available Team edition contains all the basic chat features, but lack the integration capabilities found in the Enterprise edition.

The Enterprise edition provides SAML integration out of the box, and you can configure it just like *any other SAML service in LemonLDAP::NG*

The Team edition, however, only provides SSO integration with Gitlab.

However, it is possible to configure LemonLDAP::NG to behave exactly like a Gitlab Oauth2 server, allowing Mattermost Team Edition to be integrated with LemonLDAP::NG without having to use a *Gitlab* server.

Danger: The following configuration requires your user database to expose a unique numeric identifier for every user.

15.25.2 Configuring Mattermost Team Edition

Configuring Mattermost through the *System Console* will not allow you to set the correct URLs. You need to edit the Mattermost configuration file, and avoid changing Gitlab integration settings in the *System Console*

Set the following settings in `/opt/mattermost/config/config.json`

```
"GitLabSettings": {
    "Enable": true,
    "Secret": "CHOOSE_A_CLIENT_SECRET",
    "Id": "CHOOSE_A_CLIENT_ID",
    "Scope": "",
    "AuthEndpoint": "https://auth.example.com/oauth2/gitlab_authorize",
    "TokenEndpoint": "https://auth.example.com/oauth2/token",
    "UserApiEndpoint": "https://auth.example.com/oauth2/userinfo"
},
```

Configuring your web server

Mattermost does not use OpenID Connect to communicate with Gitlab, but uses plain OAuth2 instead. Because of that, LemonLDAP::NG will not receive the `scope=` parameter and will display an error on the portal when trying to authenticate.

In order to fix this, we can add a fake OAuth2 authorize URL on the LemonLDAP::NG server that will automatically add this `scope=` parametrer, before sending the request to the correct OIDC URL

Here is an example configuration for Nginx, add it in your Portal virtualhost before any other rewrite rule:

```
rewrite ^/oauth2/gitlab_(authorize.*)$ https://auth.example.com/oauth2/$1?scope=openid
↪%20gitlab ;
```

And if you are using Apache

```
RewriteRule "^/oauth2/gitlab_authorize(.*)$" "https://auth.example.com/oauth2/authorize?
↪$1scope=openid gitlab" [QSA,NE]
```

Configuring LemonLDAP

We now have to configure LemonLDAP::NG to recognize Mattermost as a valid OAuth2 relaying party and send it the information it needs to recognize a user.

Add a *new OpenID Connect relaying party* with the following parameters:

- **Client ID**: the same you set in Mattermost configuration
- **Client Secret**: the same you set in Mattermost configuration
- **Add a new scope in "Extra claims"**

  - **Key**: `gitlab`
  - **Value**: `id username name email`

- **Add the following exported attributes**
  - `username`: set it to the session attribute containing the user login
  - `name`: session attribute containing the user's full name
  - `email`: session attribute containing the user's email
  - `id`: session attribute containing the user's numeric ID. You must set this claim type to *Integer*

Danger: Mattermost absolutely needs to receive a numerical value in the `id` claim. If you are using a LDAP server, you could use the `uidNumber` LDAP attribute. If you use something else, you will have to find a way to assign a unique numeric ID to each Mattermost user.

The `id` attribute has to be different for each user, since this is the field Mattermost will use internally to map Gitlab identities to Mattermost accouts.

Troubleshooting

If you see a HTTP code 500 when going back to mattermost, with a panic() in `(*GitLabUser).IsValid(...)` , it probably means that you are not exporting the correct attributes, but it can also mean that `id` is exported as a JSON string.

Note: An issue in version 2.0.9 prevented the `id` field from being sent correctly. Upgrade your LemonLDAP-NG installation to at least 2.0.10 and *set the claim type* to *Integer*

15.26 MediaWiki

15.26.1 Presentation

MediaWiki is a wiki software, used by the well known Wikipedia.

Several extensions allows one to configure SSO on MediaWiki:

- Automatic REMOTE_USER
- Siteminder Authentication

We will explain how to use Automatic REMOTE_USER extension.

15.26.2 Installation

The extension is presented here: http://www.mediawiki.org/wiki/Extension:AutomaticREMOTE_USER

You can download the code here: https://www.mediawiki.org/wiki/Special:ExtensionDistributor/Auth_remoteuser

You have to install '' Auth_remoteuser" in the `extensions/` directory of your MediaWiki installation:

```
cp -a Auth_remoteuser/ extensions/
```

15.26.3 Configuration

MediWiki local configuration

Then edit MediaWiki local settings

```
vi LocalSettings.php
```

```
require_once "$IP/extensions/Auth_remoteuser/Auth_remoteuser.php";
$wgAuth = new Auth_remoteuser();
```

Add then extension configuration, for example:

```
$wgAuthRemoteuserAuthz = true; /* Your own authorization test */
$wgAuthRemoteuserName = $_SERVER["HTTP_AUTH_CN"]; /* User's name */
$wgAuthRemoteuserMail = $_SERVER["HTTP_AUTH_MAIL"]; /* User's Mail */
$wgAuthRemoteuserNotify = false; /* Do not send mail notifications */
//$wgAuthRemoteuserDomain = "NETBIOSDOMAIN"; /* Remove NETBIOSDOMAIN\ from the beginning␣
↪or @NETBIOSDOMAIN at the end of a IWA username */
/* User's mail domain to append to the user name to make their email address */
//$wgAuthRemoteuserMailDomain = "example.com";

// see http://www.mediawiki.org/wiki/Manual:Hooks/SpecialPage_initList
// and http://www.mediawiki.org/w/Manual:Special_pages
// and http://lists.wikimedia.org/pipermail/mediawiki-l/2009-June/031231.html
// disable login and logout functions for all users
function LessSpecialPages(&$list) {
    unset( $list['Userlogout'] );
    unset( $list['Userlogin'] );
    return true;
}
$wgHooks['SpecialPage_initList'][]='LessSpecialPages';

// http://www.mediawiki.org/wiki/Extension:Windows_NTLM_LDAP_Auto_Auth
// remove login and logout buttons for all users
function StripLogin(&$personal_urls, &$wgTitle) {
    unset( $personal_urls["login"] );
    unset( $personal_urls["logout"] );
    unset( $personal_urls['anonlogin'] );
    return true;
}
$wgHooks['PersonalUrls'][] = 'StripLogin';
```

Danger: In last version of Auth_remoteuser and Mediawiki, empty passwords are not authorized, so you may need to patch the extension code if you get the error: "Unexpected REMOTE_USER authentication failure. Login Error was:EmptyPass". If necessary, use the code below to patch the extension:

```
sed -i "s/'wpPassword' => ''/'wpPassword' => 'none'/" extensions/Auth_remoteuser/Auth_
↪remoteuser.body.php
```

Danger: In last version of Auth_remoteuser and Mediawiki, auto-provisioning requires REMOTE_USER to match the normalized mediawiki username (for example: john_doe -> john doe), so you may need to patch the extension code if you get the error: "Unexpected REMOTE_USER authentication failure. Login Error was:WrongPluginPass" You can use the code below for normalizing logins containing "_" in the extension:

```
sed -i '/$usertest = $this->getRemoteUsername();/a\                $usertest = str_
↪replace( "_"," ", $usertest );' extensions/Auth_remoteuser/Auth_remoteuser.body.php
```

MediaWiki virtual host

Configure MediaWiki virtual host like other *protected virtual host*.

Attention: If you are protecting MediaWiki with LL::NG as reverse proxy, *convert header into REMOTE_USER environment variable*.

- For Apache:

```
<VirtualHost *:80>
       ServerName mediawiki.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name mediawiki.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
```

(continues on next page)

(continued from previous page)

```
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

MediaWiki virtual host in Manager

Go to the Manager and *create a new virtual host* for MediaWiki.

Just configure the *access rules*. You can also add a rule for logout:

```
Userlogout => logout_sso
```

You can create these two headers to fill user name and mail (see extension configuration):

```
Auth-Cn => $cn
Auth-Mail => $mail
```

If using LL::NG as reverse proxy, configure also the `Auth-User` *header*,

15.27 Mobilizon

15.27.1 Presentation

Mobilizon is an online tool to help manage your events, your profiles and your groups.

Mobilizon lets users authenticate with OpenID Connect through the same plugin used by Keycloak.

First, make sure you have set up LemonLDAP::NG 's *OpenID Connect service* and added *a Relaying Party for your Mobilizon instance*

The only options you need to configure are:

- *Client ID*: choose one
- *Client Secret*: choose one
- *Allowed redirection addresses for login*: `https://mobilizon.example.com/auth/keycloak/callback`

15.27.2 Mobilizon configuration

Edit `/etc/mobilizon/config.exs`, and adjust the Client ID, Client Secret and URLs to match your domain

```
config :ueberauth,
    Ueberauth,
    providers: [
      keycloak: {Ueberauth.Strategy.Keycloak, [default_scope: "openid profile email"]}
    ]

config :mobilizon, :auth,
  oauth_consumer_strategies: [
    {:keycloak, "LemonLDAP::NG"}
  ]

config :ueberauth, Ueberauth.Strategy.Keycloak.OAuth,
  client_id: "CHANGEME",
  client_secret: "CHANGEME",
  site: "https://auth.example.com",
  authorize_url: "https://auth.example.com/oauth2/authorize",
  token_url: "https://auth.example.com/oauth2/token",
  userinfo_url: "https://auth.example.com/oauth2/userinfo",
  token_method: :post
```

15.28 NextCloud

15.28.1 Presentation

NextCloud is a fork of Owncloud, suite of client-server software for creating file hosting services and using them.

This documentation explains how to interconnect LemonLDAP::NG and NextCloud using SAML 2.0 protocol.

15.28.2 Pre-requisites

NextCloud

You need to install the software.

Tip: If your NextCloud is behind a proxy (thus having a private IP), metadata generated by NextCloud won't work.

Consider changing the configuration of NextCloud to force the domain and the protocol, in **$nextcloud-rootwww/config/config.php**, add the following:

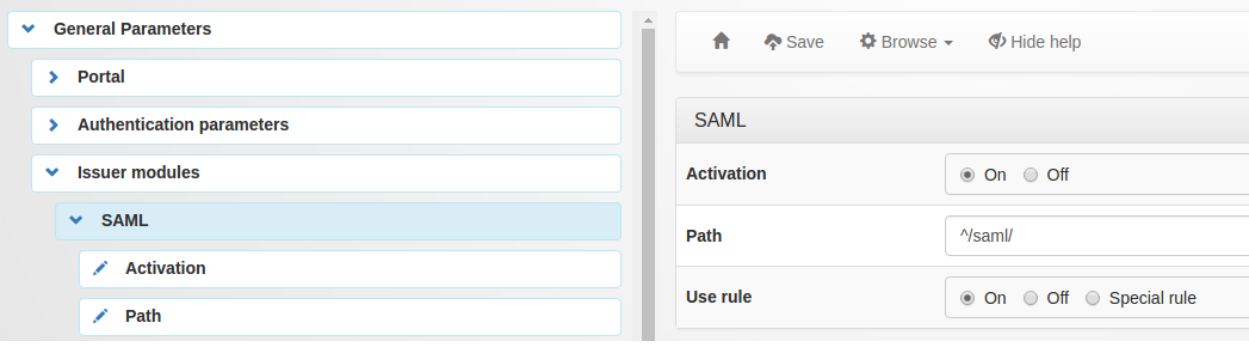
```
'overwritehost' => 'nextcloud.example.com',
'overwriteprotocol' => 'https',
```

You also need to enable the "SAML authentication" plugin in your NextCloud. <code> + Apps -> Not enabled -> SAML authentication</code>

LL:NG

You need to enable SAML 2.0 issuer module in LL:NG:

```
"General Parameters -> Issuer modules -> SAML -> Activation"
```



15.28.3 NextCloud, SAML 2.0 configuration

Configuration of SAML 2.0 in NextCloud is pretty straightforward.

```
Administration -> SAML authentication
```

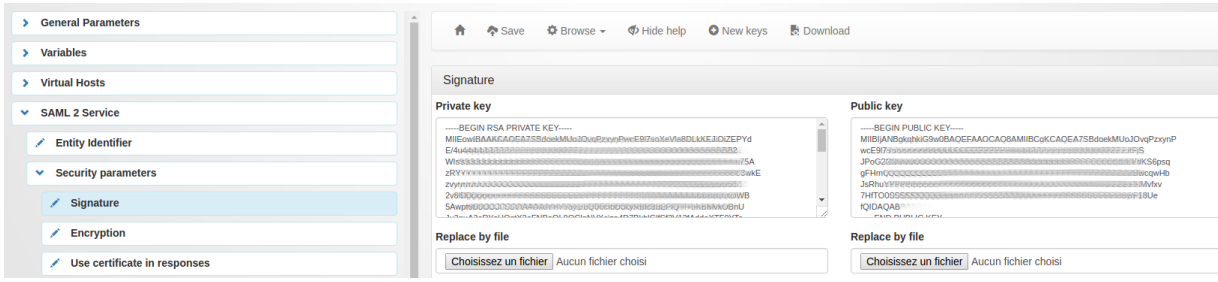
You will find the following fields:

- **Attribute to map the UID to**: Identity attribute provided by your LL:NG that will be used as UID in NextCloud.
- **Identity Provider Data**:

- **Identifier of the IdP entity**: SAML Metadata URL of your LL:NG
- **URL Target of the IdP where the SP will send the Authentication Request Message**: SingleSignOn URL of your LL:NG
- **URL Location of the IdP where the SP will send the SLO Request**: SingleLogOut URL of your LL:NG
- **Public X.509 certificate of the IdP**: Certificate of your LL:NG (see below for instructions)

We need a few steps to generate our LL:NG certificate (unless you already have one). You first need to create a pair of SSH Keys in LL:NG:

```
SAML 2 Service -> Security Parameters -> Signature
```

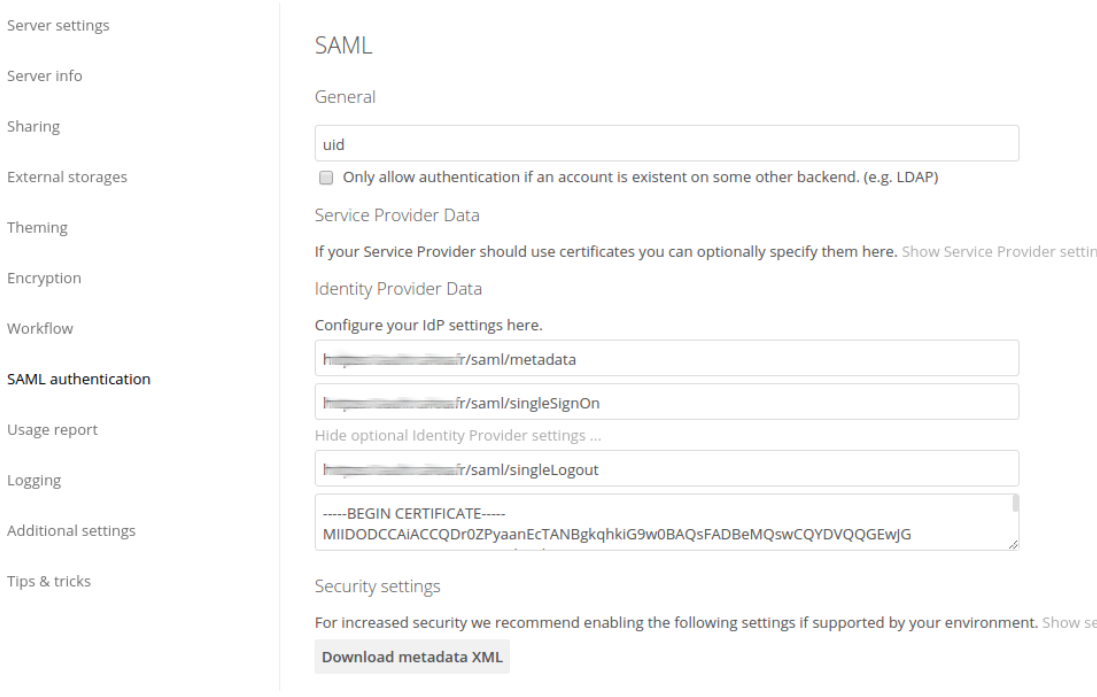


and click "New keys"

Take the private key in a private.key file, and run the following:

```
openssl req -new -key private.key -out cert.csr
openssl x509 -req -days 3650 -in cert.csr -signkey private.key -out cert.pem
```

Copy/Paste the content of your new cert.pem in the "Public X.509 certificate of the IdP" field of your NextCloud.



Your fields should look like this:

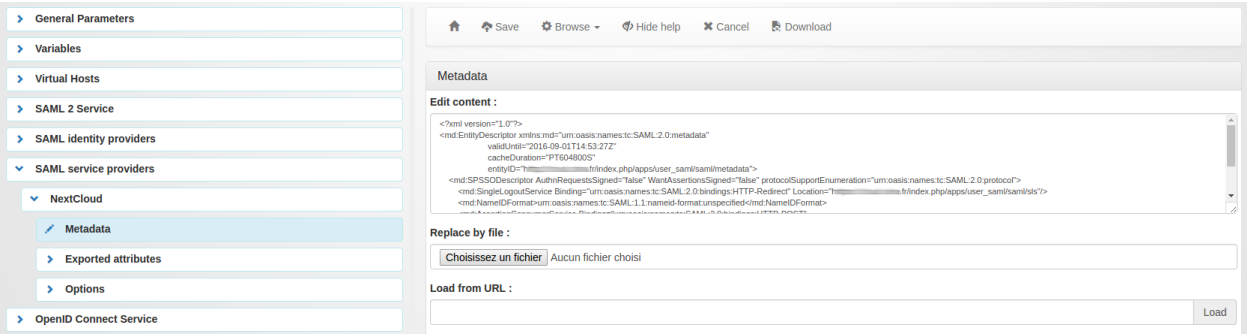
You can now download your metadata xml file.

15.28.4 LL:NG, SAML 2.0 Service Provider configuration

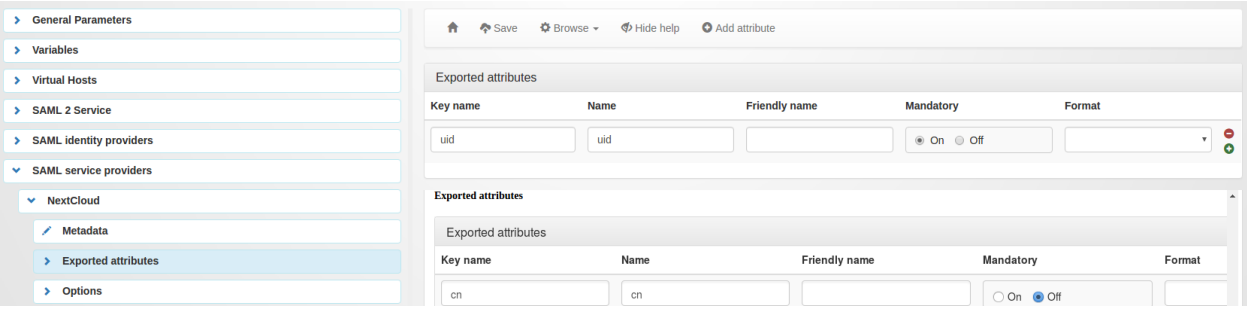
We now have to define a service provider (e.g our nextcloud) in LL:NG.

Go to "SAML service providers", click on "Add SAML SP" and name it as you want (example : 'NextCloud')

In the new subtree 'NextCloud', open 'Metadata' and paste the content of your previously downloaded file (or upload the file)



Now go in "Exported attributes" and add, at least, the 'uid'



Don't forget to save your configuration.

You are now good to go, and you can add the application in *your menu* and *your virtual hosts*.

15.29 OBM

15.29.1 Presentation

OBM is enterprise-class messaging and collaboration platform for workgroup or enterprises with many thousands users. OBM includes Groupware, messaging server, CRM, LDAP, Windows Domain, smartphone and PDA synchronization…

OBM is shipped with a LL::NG plugin with these features:

- SSO on OBM web interface
- Logout
- User provisioning (account auto creation at first connection)

15.29.2 Configuration

OBM

To enable LL::NG authentication plugin, go in `/etc/obm/obm_conf.inc`:

```
$auth_kind = 'LemonLDAP';

$lemonldap_config = Array(
                "auto_update"           => true,
                "auto_update_force_user" => true,
                "auto_update_force_group" => false,
                "url_logout"            => "https://OBMURL/logout",
                "server_ip_address"     => "localhost",
                "server_ip_check"       => false,
                "debug_level"           => "NONE",
//                "debug_header_name"     => "HTTP_OBM_UID",
//                "group_header_name"     => "HTTP_OBM_GROUPS",
                "headers_map"           => Array(
                        //"userobm_gid"                   => "HTTP_OBM_GID",
                        //"userobm_domain_id"           => ,
                        "userobm_login"                 => "HTTP_OBM_UID",
                        "userobm_password"              => "HTTP_OBM_USERPASSWORD",
                        //"userobm_password_type"       => ,
                        "userobm_perms"                 => "HTTP_OBM_PERMS",
                        //"userobm_kind"                => ,
                        "userobm_lastname"              => "HTTP_OBM_SN",
                        "userobm_firstname"             => "HTTP_OBM_GIVENNAME",
//                        "userobm_title"                 => "HTTP_OBM_TITLE",
                        "userobm_email"                 => "HTTP_OBM_MAIL",
                        "userobm_datebegin"             => "HTTP_OBM_DATEBEGIN",
                        //"userobm_account_dateexp"     => ,
                        //"userobm_delegation_target"   => ,
                        //"userobm_delegation"          => ,
                        "userobm_description"           => "HTTP_OBM_DESCRIPTION",
                        //"userobm_archive"             => ,
                        //"userobm_hidden"              => ,
                        //"userobm_status"              => ,
                        //"userobm_local"               => ,
                        //"userobm_photo_id"            => ,
```

(continues on next page)

(continued from previous page)

```
                        "userobm_phone"                 => "HTTP_OBM_TELEPHONENUMBER",
                        //"userobom_phone2"             => ,
                        //"userobm_mobile"              => ,
                        "userobm_fax"                   => "HTTP_OBM_
↪FACSIMILETELEPHONENUMBER",
                        //"userobm_fax2"                => ,
                        "userobm_company"               => "HTTP_OBM_O",
                        //"userobm_direction"           => ,
                        "userobm_service"               => "HTTP_OBM_OU",
                        "userobm_address1"              => "HTTP_OBM_POSTALADDRESS",
                        //"userobm_address2"            => ,
                        //"userobm_address3"            => ,
                        "userobm_zipcode"               => "HTTP_OBM_POSTALCODE",
                        "userobm_town"                  => "HTTP_OBM_L",
                        "userobm_zipcode"               => "HTTP_OBM_POSTALCODE",
                        "userobm_town"                  => "HTTP_OBM_L",
                        //"userobm_expresspostal"       => ,
                        //"userobm_host_id"             => ,
                        //"userobm_web_perms"           => ,
                        //"userobm_web_list"            => ,
                        //"userobm_web_all"             => ,
                        //"userobm_mail_perms"          => ,
                        //"userobm_mail_ext_perms"      => ,
                        //"userobm_mail_server_id"      => ,
                        //"userobm_mail_server_hostname" => ,
                        "userobm_mail_quota"            => "HTTP_OBM_MAILQUOTA",
                        //"userobm_nomade_perms"        => ,
                        //"userobm_nomade_enable"       => ,
                        //"userobm_nomade_local_copy"   => ,
                        //"userobm_email_nomade"        => ,
                        //"userobm_vacation_enable"     => ,
                        //"userobm_vacation_datebegin"  => ,
                        //"userobm_vacation_dateend"    => ,
                        //"userobm_vacation_message"    => ,
                        //"userobm_samba_perms"         => ,
                        //"userobm_samba_home"          => ,
                        //"userobm_samba_home_drive"    => ,
                        //"userobm_samba_logon_script"  => ,
                        // ---- Unused values ? ----
                        "userobm_ext_id"                => "HTTP_OBM_SERIALNUMBER",
                        //"userobm_system"              => ,
                        //"userobm_nomade_datebegin"    => ,
                        //"userobm_nomade_dateend"      => ,
                        //"userobm_location"            => ,
                        //"userobm_education"           => ,
                        ),
        );
```

Parameters:

- **url_logout**: URL used by OBM to logout, will be caught by LL::NG
- **headers_map**: map OBM internal field to LL::NG header

Edit also OBM configuration to enable LL::NG Handler:

- For Apache:

```
<VirtualHost *:80>
    ServerName obm.example.com

    # SSO protection
    PerlHeaderParserHandler Lemonldap::NG::Handler

    DocumentRoot /usr/share/obm/php

    ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name obm.example.com;
  root /usr/share/obm/php;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location ~ \.php$ {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

LL::NG

Attributes and macros

You will need to collect all attributes needed to create a user in OBM, this includes:

- First name
- Last name
- Login
- Mail
- …

To add these attributes, go in Manager, `Variables` » `Exported Variables`.

Attention: If you plan to forward user's password to OBM, then you have to *keep the password in session*.

You may also create these macros to manage OBM administrator account (`Variables` » `Macros`):

field	value
uidR	`($uid =~ /^admin0/i)[0] ? "admin0\@global.virt" :  $uid`
mailR	`($uid =~ /admin0/i)[0] ? "" :  ($mail =~ / ([ @]+)/)[0] . "\@example.com"`

Virtual host

Create OBM virtual host (for example obm.example.com) in LL::NG configuration: `Virtual Hosts` » `New virtual host`.

Then edit rules and headers.

Rules

Define at least:

- **Default rule**: who can access to the application
- **Logout rule**: catch OBM logout
- **Exceptions**: allow anonymous access for specific URLs (connectors, etc.)

field	value
^/logout	logout_sso
^/obm-sync	unprotect
^/minig	unprotect
^/Microsoft-Server-ActiveSync	unprotect
^/caldav	unprotect
default	accept (or whatever you want)

Headers

Define headers used in OBM mapping, for example:

field	valeur
OBM_GIVENNAME	$givenName
OBM_GROUPS	$groups
OBM_UID	$uidR
OBM_MAIL	$mailR
OBM_USERPASSWORD	$_password

Other

Do not forget to add OBM in *applications menu*.

15.30 Office 365

15.30.1 Presentation

Office 365 provides online access to Microsoft products like Office, Outlook or Yammer. Authentication is done on https://login.microsoftonline.com/ and can be forwarded to an SAML Identity Provider.

15.30.2 Configuration

Office 365

You first need to install AzureAD PowerShell to be able to run administrative commands.

Then run this script:

```
$dom = "mycompany.com"
$brand = "My Company"
$url = "https://auth.example.com/saml/singleSignOn"
$uri = "https://auth.example.com/saml/metadata"
$logouturl = "https://auth.example.com/?logout=1"
$cert = "xxxxxxxxxxxxxxxxxx"

Set-MsolDomainAuthentication -DomainName $dom -FederationBrandName $brand -
↪Authentication Federated  -PassiveLogOnUri $url -SigningCertificate $cert -IssuerUri
↪$uri  -LogOffUri $logouturl -PreferredAuthenticationProtocol SAMLP
```

Where parameters are:

- dom: Your Office 365 domain

- brand: Simple label
- url: The SAML SSO endpoint
- uri: The SAML metadata endpoint
- logouturl: Logout URL
- cert: The SAML certificate containing the signature public key

If you have several Office365 domains, you can't use the same URLs for each domains. To be able to have a single SAML IDP for several domains, you must add the 'domain' GET parameters at the end of SSO endpoint and metadata URLs, for example:

- domain 'mycompany.com':
  - url: https://auth.example.com/saml/singleSignOn?domain=mycompany
  - uri: https://auth.example.com/saml/metadata?domain=mycompany
- domain 'myfirm.com':
  - url: https://auth.example.com/saml/singleSignOn?domain=myfirm
  - uri: https://auth.example.com/saml/metadata?domain=myfirm

LemonLDAP::NG

Create a new SAML Service Provider and import Microsoft metadata from https://nexus.microsoftonline-p.com/federationmetadata/saml20/federationmetadata.xml

Set the NameID value to persistent, or any immutable value for the user.

Create a SAML attribute named IDPEmail which contains the user principal name (UPN).

15.31 Publik

15.31.1 Presentation

Publik is an open-source citizen relationship management tool.

See the official Publik website for a complete presentation.

It feature an OpenID Connect login that work with LemonLDAP::NG.

15.31.2 Configuring Publik

Connect to your publik instance authentic2 webui with an Admin user, in the admin panel, go to "Authentic2_Auth_Oidc" › "Oidc providers".

Click on "Add Oidc Provider".

- Name : LemonLDAP SSO
- Short id : lemonldap
- Provider : https://auth.example.com/
- Client id : clientid
- Client secret : secret
- Authorization endpoint : https://auth.example.com/oauth2/authorize
- Token endpoint : https://auth.example.com/oauth2/token
- Userinfo endpoint : https://auth.example.com/oauth2/userinfo
- End session endpont : https://auth.example.com/oauth2/logout
- WebKey JSON : Copy/Paste the content of https://auth.example.com/oauth2/jwks
- Claims Enabled : yes
- Show on connection page : yes

Strategy and Collectivity can be configured based to your needs.

OIDC Claim mappings can be configured based on your needs.

Configuring LemonLDAP

We now have to configure LemonLDAP::NG to recognize publik as a valid OIDC relying party.

Add a *new OpenID Connect relying party* with the following parameters (Options -> Basic) :

- **Client ID**: the same you set in Publik configuration.
- **Client Secret**: the same you set in Publik configuration.
- **Allowed redirection addresses for login**: The "Callback URL" for authentic2 : https://authentic2-instance/accounts/oidc/callback/

15.32 phpLDAPadmin

15.32.1 Presentation

phpLDAPadmin is an LDAP administration tool written in PHP.

phpLDAPadmin will connect to the directory with a static DN and password, and so will not request authentication anymore. The access to phpLDAPadmin will be protected by LemonLDAP::NG with specific access rules.

Danger: phpLDAPadmin will have no idea of the user connected to the WebSSO. So a simple user can have admin rights on the LDAP directory if your access rules are too lazy.

15.32.2 Configuration

phpLDAPadmin local configuration

Just set the authentication type to `config` and indicate DN and password inside the file `config.php`:

```
$servers->SetValue('login','auth_type','config');
$servers->SetValue('login','bind_id','cn=Manager,dc=example,dc=com');
$servers->SetValue('login','bind_pass','secret');
```

phpLDAPadmin virtual host

Configure phpLDAPadmin virtual host like other *protected virtual host*.

- For Apache:

```
<VirtualHost *:80>
       ServerName phpldapadmin.example.com

       PerlHeaderParserHandler Lemonldap::NG::Handler

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name phpldapadmin.example.com;
  root /path/to/application;
  # Internal authentication request
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
```

(continues on next page)

(continued from previous page)

```
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location / {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

phpLDAPadmin virtual host in Manager

Go to the Manager and *create a new virtual host* for phpLDAPadmin.

Just configure the *access rules*.

No *headers* are required.

15.33 RoundCube

15.33.1 Presentation

RoundCube webmail is a browser-based multilingual IMAP client with an application-like user interface. It provides full functionality you expect from an email client, including MIME support, address book, folder manipulation, message searching and spell checking.

15.33.2 Configuration

LemonLDAP::NG

- Add a new virtual host webmail.domain.tld
- Add a new rule:

```
"^/\?_task\=logout" -> "logout_app https://auth.domain.tld"
```

- in HTTP headers, you need Auth-User ($mail) and Auth-Pw ($_password).

Attention: To be able to forward password to RoundCube, see *how to store password in session*

- Configure *Apache or Nginx virtual host*

RoundCube

- install http_authentication plugin
- Patch it to replace `PHP_AUTH_*` by `HTTP_AUTH_*`
- enable http_authentication plugin in main.inc.php :

```
$rcmail_config['plugins'] = array('http_authentication');
```

15.34 SalesForce

15.34.1 Presentation

Salesforce Inc. is a cloud computing company. It is best known for their CRM products and social networking applications.

It allows one to use SAML to authenticate users. It can deal with both SP and IdP initiated modes.

This page presents the SP initiated mode.

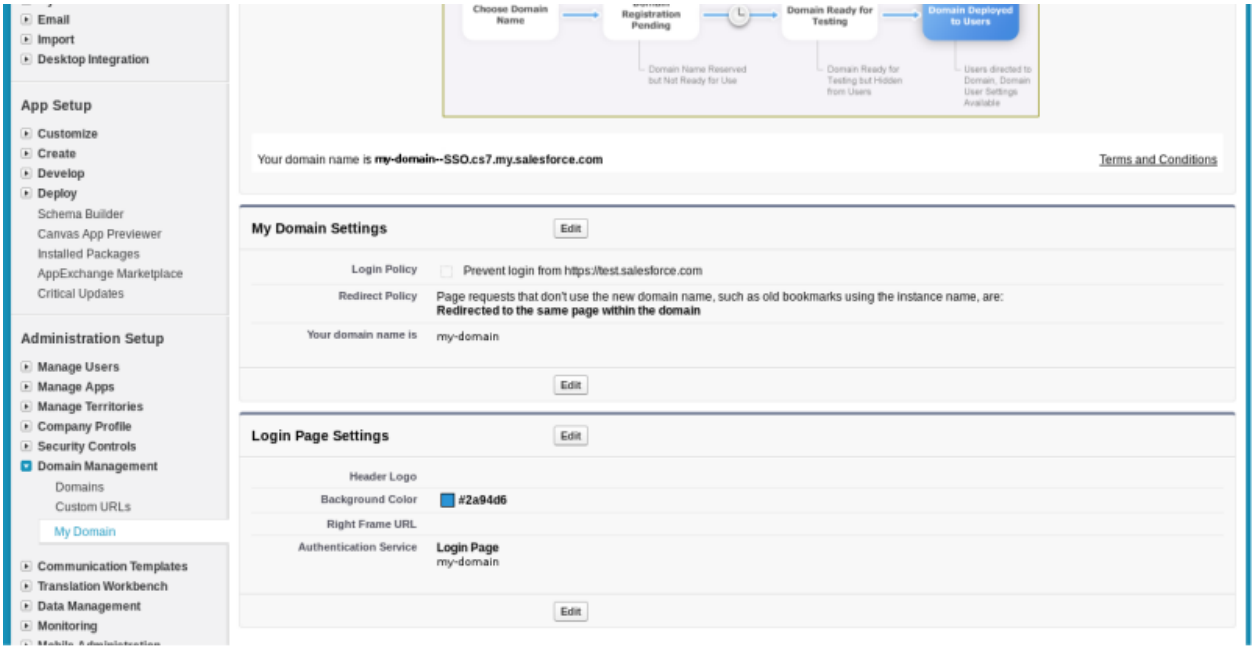
To work with LL::NG it requires:

- LL::NG configured as *SAML Identity Provider*

15.34.2 Configuration

You should have configured LL::NG as a *SAML Identity Provider*.

Create Salesforce domain



For using SP-initiated mode, you must create your salesforce domain. Creation can take up to 1 hour. (if it is superior to 1h, then there is a problem. Problems are generally resolved in up to 72 hours)

Then you must **deploy** this domain in order to go on with the configuration.

Finally, just ensure that at least:

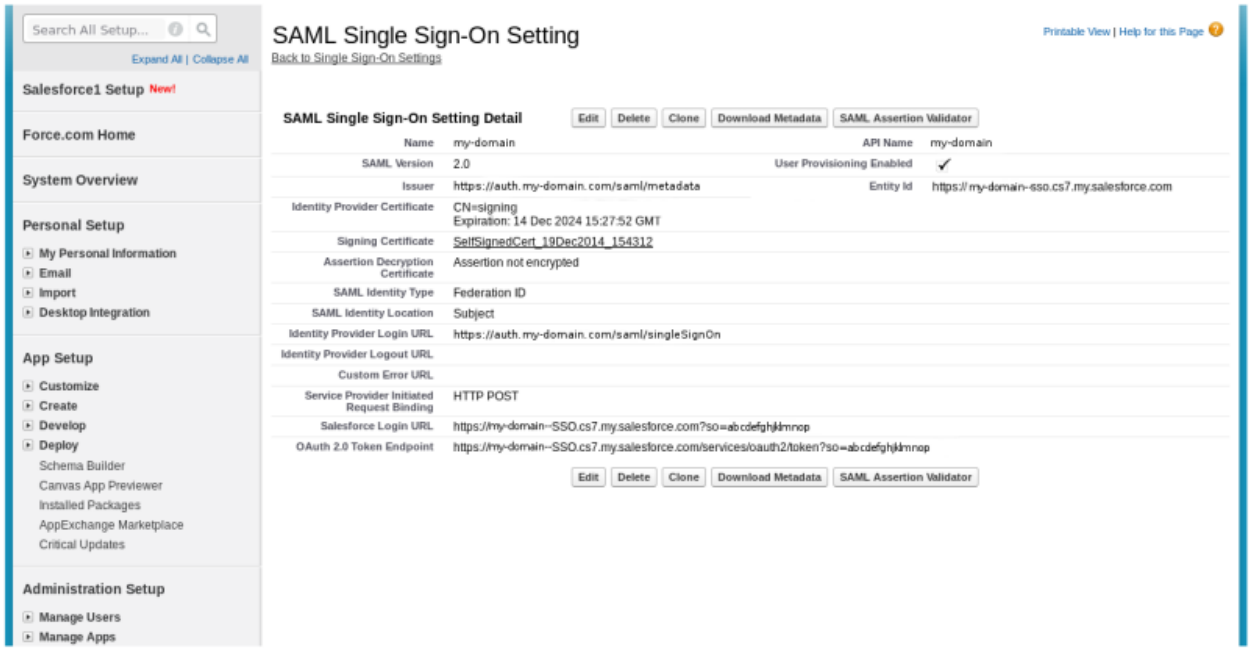
- Login policy
- Redirect policy
- domain name
- authentication service

match with the correct values. (adapt the domain if necessary)

> **Attention:** For now, the authentication service parameter has no domain available. You must come back later to fill this parameter. Once SAML cinematics are working, you can then put your domain, and delete the login form, and you'll have an automatic redirection to your Identity Provider (no need for the user to click). Note that you can always access Salesforce by the general login page: https://login.salesforce.com

SAML settings

Salesforce is not able to read metadata, you must fill the information into a form.

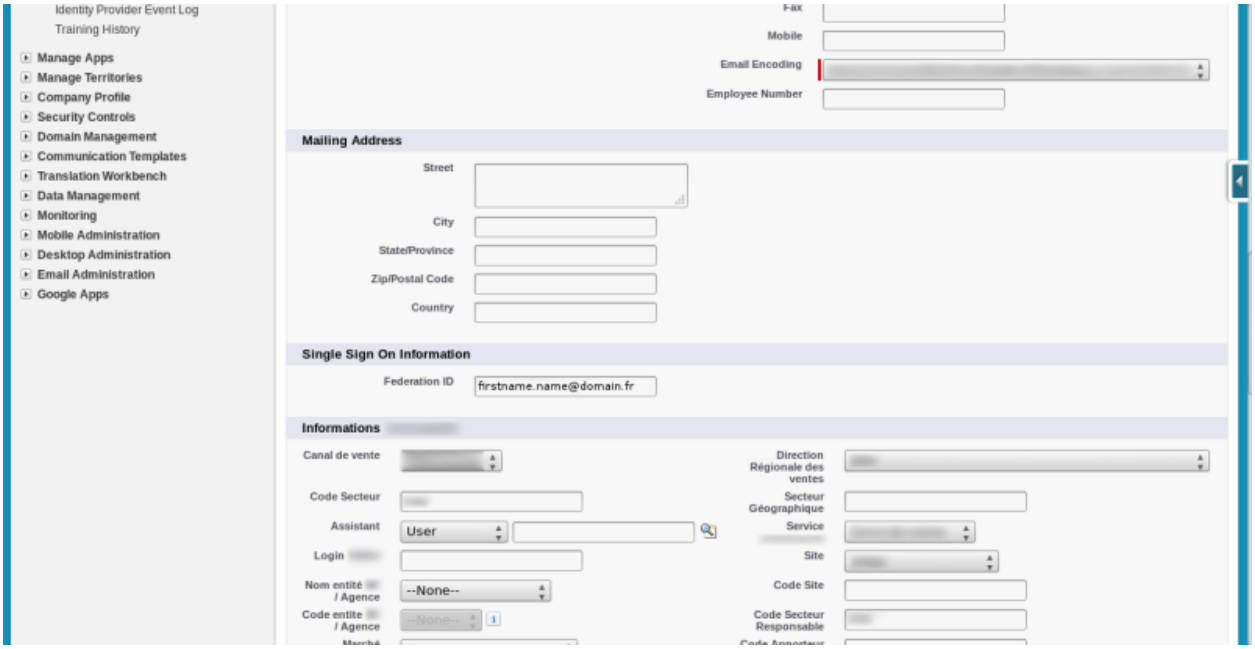


Go to the SAML Single Sign On settings, and fill these information:

- Name: should be filled automatically with your organization or domain
- SAML Version: check that version 2.0 is used
- Issuer: this is the LemonLDAP::NG (our IdP) Entity Id, which is by default #PORTAL#/saml/metadata
- Identity Provider Certificate: whereas it is mentioned that this is the authentication certificate, you must give your LemonLDAP::NG (IdP) signing certificate. If you don't have one, create it with the signing key pair already generated (you could do this with openssl). SSL authentication (https) does not seem to be checked anyway.
- Signing Certificate: choose a certificate for SP signature. (create one if none is present)
- Assertion decryption Certificate: choose a certificate only if you want to cipher your assertion. (default is not to cipher)
- SAML Identity Type: choose Federation ID. This means that the user Name ID will be mapped to the Federation ID field. (see next section)
- SAML Identity Location: choose if the user Name ID is held in the subject or in some attribute
- Identity Provider Login URL: the user/password SAML portal location on the IdP
- Identity Provider Logout URL: the logout location on the IdP
- Custom Error URL: you can redirect the user to a special page when an error is happening
- SP Initiated Binding: chose any of the supported binding (every one listed there is currently supported on LemonLDAP::NG) HTTP POST is a good choice
- Salesforce Login URL: generated automatically. This is the entry point of our login cinematic.
- OAuth 2.0 Token Endpoint: not used here
- API Name: filled automatically
- User Provisioning Enabled: should create automatically the user in Salesforce (not functionnal right now)
- EntityId: Salesforce (the SP) Entity ID. Fill this field accordingly. It should be the same value as the organization domain url, displayed on the previous section

Configure Federation ID

Finally, configure for each user his Federation ID value. It will be the link between the SAML assertion coming from LemonLDAP::NG (the IdP) and a given user in Salesforce. Here, the mail has been chosen as the user Name ID.



Once this is completed, click to export the Salesforce metadata and import them into LemonLDAP::NG, into the declaration of the Salesforce Service Provider.

See *Register partner Service Provider on LemonLDAP::NG* configuration chapter.

15.35 SAP

15.35.1 HTTP header

Read the following documentation: http://help.sap.com/saphelp_nw70/helpdata/en/d0/a3d940c2653126e10000000a1550b0/frameset.htm

15.35.2 SAML

Read the following documentation: https://help.sap.com/saphelp_nw70/helpdata/en/94/695b3ebd564644e10000000a114084/content.htm

15.36 simpleSAMLphp

15.36.1 Presentation

simpleSAMLphp is an identity/service provider written in PHP. It supports a lot of protocols like CAS, OpenID and SAML.

This documentation explains how to interconnect LemonLDAP::NG and simpleSAMLphp using SAML 2.0 protocol.

15.36.2 Pre-requisites

simpleSAMLphp

You need to install the software. If using Debian, just do:

```
apt-get install simplesamlphp
```

We suppose that configuration is done in `/etc/simplesamlphp` and that simpleSAMLphp is accessible at http://localhost/simplesamlphp.

To be able to sign SAML messages, you need to create a certificate. First set where certificates are stored:

```
vi /etc/simplesamlphp/config.php
```

```
'certdir' => '/etc/simplesamlphp/certs/',
```

Create directory and generate the certificate

```
mkdir /etc/simplesamlphp/certs/
cd /etc/simplesamlphp/certs/
openssl req -newkey rsa:2048 -new -x509 -days 3652 -nodes -out saml.crt -keyout saml.pem
```

Then associate this certificate to the default SP:

```
vi /etc/simplesamlphp/authsources.php
```

```
'default-sp' => array(
    'saml:SP',
    'privatekey' => 'saml.pem',
    'certificate' => 'saml.crt',
```

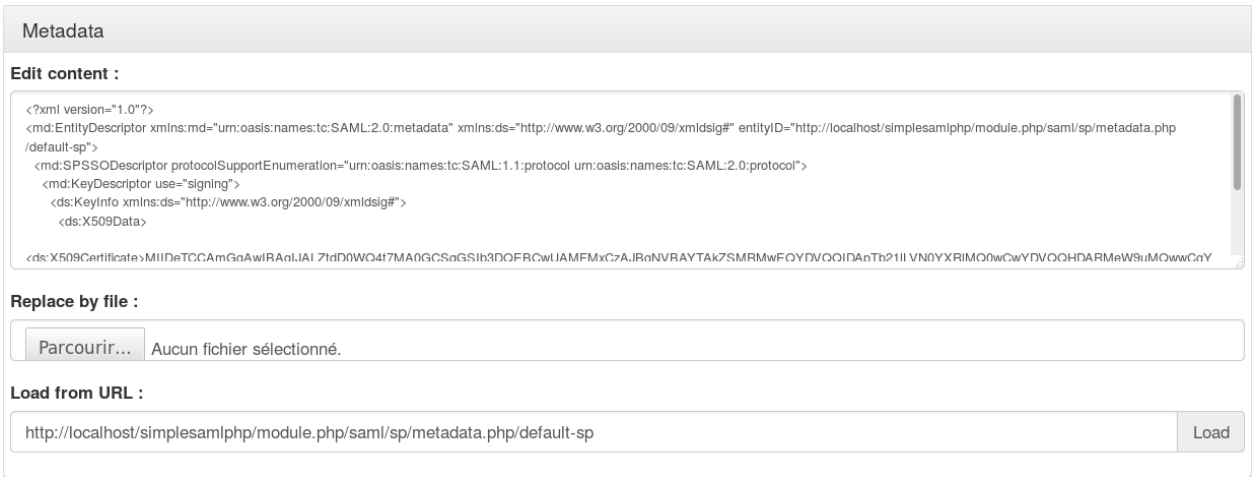
LemonLDAP::NG

You need to configure *SAML Service*. Be sure to convert public key in a certificate, as described in the *security chapter* as simpleSAMLphp can't use the public key.

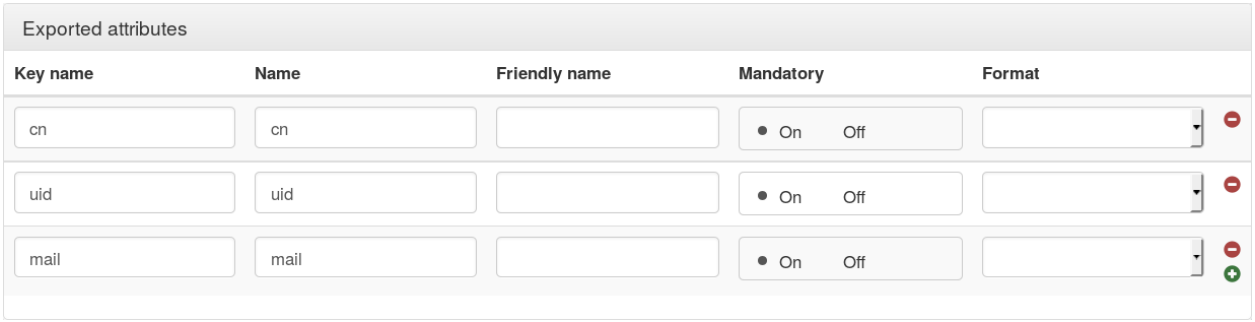
15.36.3 simpleSAMLphp as Service Provider

We suppose you configured LemonLDAP::NG as *SAML Identity Provider* and want to use simpleSAMLphp as Service Provider.

In LL::NG Manager, create an new SP and load simpleSAMLphp metadata through URL (by default: http://localhost/simplesamlphp/module.php/saml/sp/metadata.php/default-sp):



Then set some attributes that will be sent to simpleSAMLphp:



Tip: Set `Mandatory` to `On` to force attributes in authentication response.

You can also force all signatures:

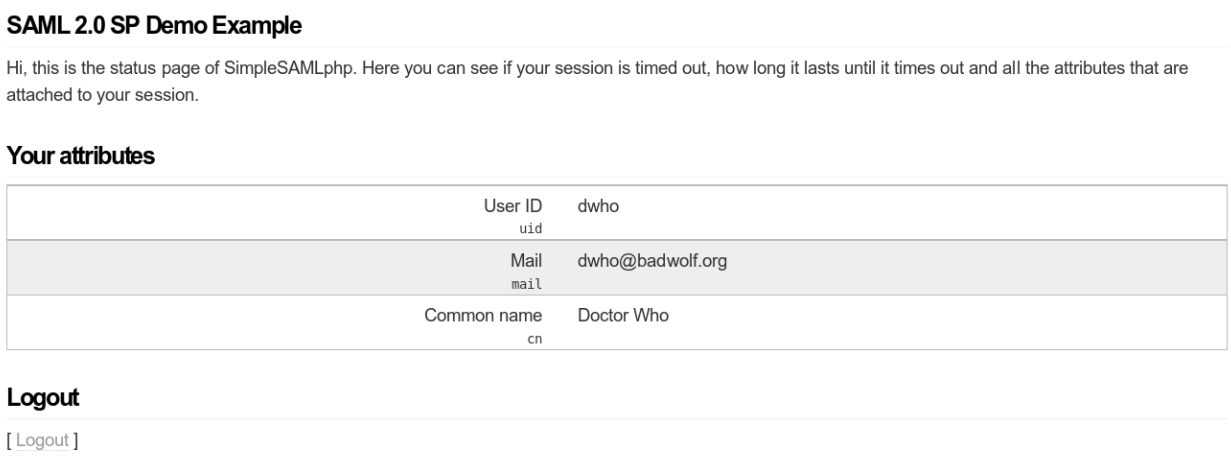
On simpleSAMLphp side, use the metadata converter (by default: http://localhost/simplesamlphp/admin/metadata-converter.php) to convert LL::NG metadata (by default: http://auth.example.com/saml/metadata) into internal PHP representation. Copy the `saml20-idp-remote` content:

```
vi /etc/simplesamlphp/metadata/saml20-idp-remote.php
```

```
<?php
$metadata['http://auth.example.com/saml/metadata'] = array (
  'entityid' => 'http://auth.example.com/saml/metadata',
...
   // Add this option to force SLO requests signature
   'sign.logout' => true,
);
?>
```

Tip: Don't forget PHP start and end tag to have a valid PHP file.

All is ready, you can now test the authentication (by default: http://localhost/simplesamlphp/module.php/core/authenticate.php). You should see something like that:



15.36.4 simpleSAMLphp as Identity Provider

We suppose you configured LemonLDAP::NG as *SAML Service Provider* and want to use simpleSAMLphp as Identity Provider.

First, you need to activate IDP feature in simpleSAMLphp:

```
vi /etc/simplesamlphp/config.php
```

```
'enable.saml20-idp' => true,
```

And create a default IDP configuration:

```
vi /etc/simplesamlphp/metadata/saml20-idp-hosted.php
```

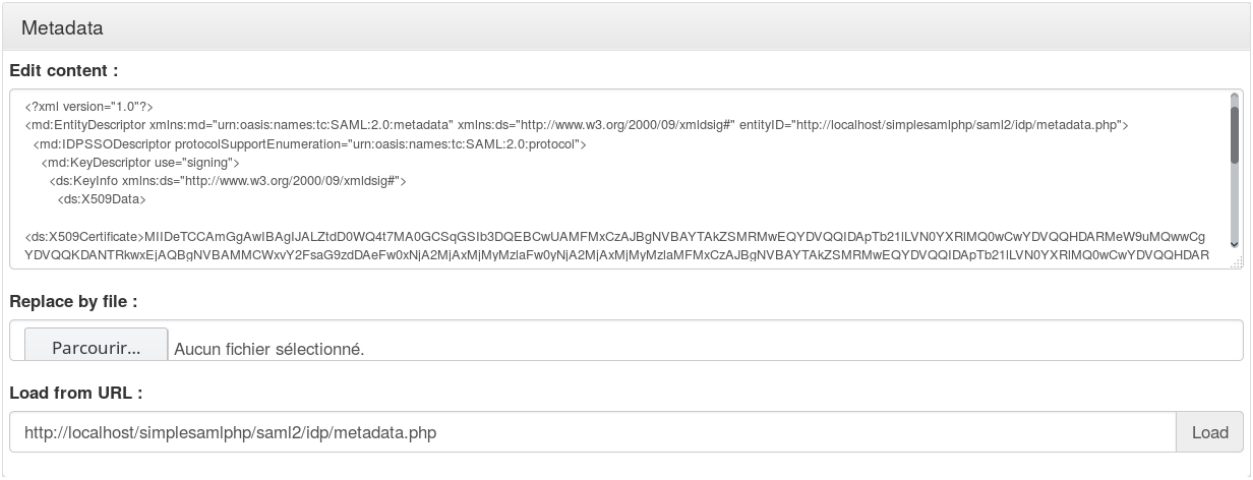
```
<?php
$metadata['__DYNAMIC:1__'] = array(
    /*
     * The hostname for this IdP. This makes it possible to run multiple
     * IdPs from the same configuration. '__DEFAULT__' means that this one
     * should be used by default.
     */
    'host' => '__DEFAULT__',

    /*
     * The private key and certificate to use when signing responses.
     * These are stored in the cert-directory.
     */
    'privatekey' => 'saml.pem',
    'certificate' => 'saml.crt',

    /*
     * The authentication source which should be used to authenticate the
     * user. This must match one of the entries in config/authsources.php.
     */
    'auth' => 'admin',
    // Sign SLO messages
    'sign.logout' => true,
);
?>
```

Attention: You need to configure your own certificates and authentication scheme

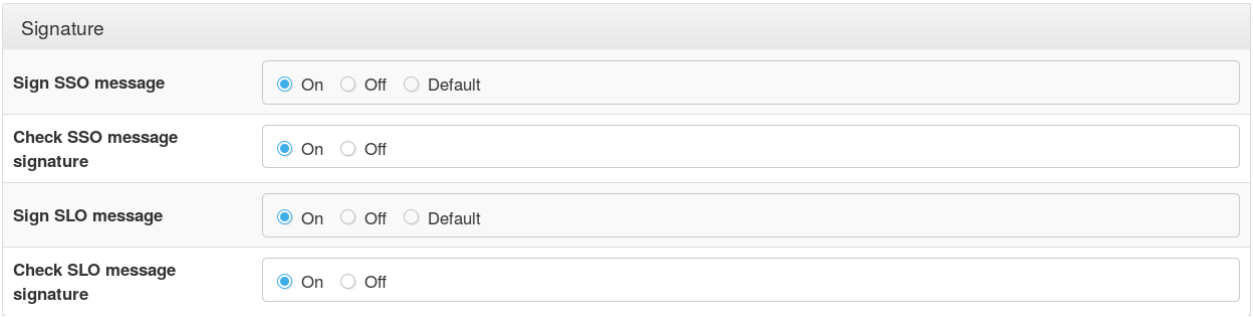
Now in LL::NG Manager, create a new IDP and import metadata with URL (by default: http://localhost/simplesamlphp/saml2/idp/metadata.php):



List attributes you want to collect:

Tip: You can keep `Mandatory` to `Off` to not fail if attribute is not sent by IDP

And activate all signatures:



To finish, you need to declare LL::NG SP in simpleSAMLphp. Use the metadata converter (by default: http://localhost/simplesamlphp/admin/metadata-converter.php) to convert LL::NG metadata (by default: http://auth.example.com/saml/metadata) into internal PHP representation. Copy the `saml20-sp-remote` content:

```
vi /etc/simplesamlphp/metadata/saml20-sp-remote.php
```

```
<?php
$metadata['http://auth.example.com/saml/metadata'] = array (
  'entityid' => 'http://auth.example.com/saml/metadata',
...
);
?>
```

Tip: Don't forget PHP start and end tag to have a valid PHP file.

All is ready, you can now test the authentication from LL::NG portal.

15.37 Spring Security (ACEGI)

15.37.1 Presentation

Spring Security is the new ACEGI name. This is a well known security framework for J2EE applications.

Spring Security provides a default `pre-authentication` mechanism that can be used to connect your J2EE application to LL::NG.

15.37.2 Configuration

You can find all suitable information here: http://static.springsource.org/spring-security/site/docs/3.0.x/reference/preauth.html

To summarize, to get the user connected through the `Auth-User` HTTP Header, use this Sping Security configuration:

```
<bean id="LemonLDAPNGFilter" class=
"org.springframework.security.web.authentication.preauth.header.
↪RequestHeaderPreAuthenticatedProcessingFilter">
    <security:custom-filter position="PRE_AUTH_FILTER" />
    <property name="principalRequestHeader" value="Auth-User"/>
    <property name="authenticationManager" ref="authenticationManager" />
</bean>

<bean id="preauthAuthProvider" class="org.springframework.security.web.authentication.
↪preauth.PreAuthenticatedAuthenticationProvider">
  <security:custom-authentication-provider />
    <property name="preAuthenticatedUserDetailsService">
    <bean id="userDetailsServiceWrapper" class="org.springframework.security.userdetails.
↪UserDetailsByNameServiceWrapper">
      <property name="userDetailsService" ref="userDetailsService"/>
    </bean>
  </property>
</bean>

<security:authentication-manager alias="authenticationManager" />
```

15.38 PHP (Symfony)

15.38.1 Presentation

Symfony is the well-known PHP framework. It is intended to ease the development of PHP applications.

Symfony provides many methods conventions to authenticate users (basic, ldap,...) and to load external user sources (ldap, database). The method presented here relies on the "remote_user" method. (in security firewall)

15.38.2 Configuration

Follow these step to protect your application using the "REMOTE_USER" HTTP header.

1. Adapt the app/config/security.yml configuration file as below:

```
security:

    encoders:
        AppBundle\Security\User\HeaderUser: plaintext

    providers:
        header:
            id: AppBundle\Security\User\HeaderUserProvider

    firewalls:
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false

        main:
            pattern: ^/
            remote_user:
                user: HTTP_REMOTE_USER
            provider: header
```

- encoders : define a password hashing scheme (useless in our case, but the parameter is mandatory)
- providers : define the user providers (even virtual)
- remote_user : define the authentication method to "assume the user is already authenticated and get an http variable to know his username"
- user : define the HTTP header containing the username
- provider : references the previously defined provider owning the user data (in our scenario, a virtual)

2. Define a "header user" class

Create the file src/AppBundle/Security/User/HeaderUser.php :

```php
<?php

// src/Security/User/HeaderUser.php
namespace AppBundle\Security\User;

use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\User\EquatableInterface;

class HeaderUser implements UserInterface, EquatableInterface
{
    private $username;
    private $password;
    private $salt;
    private $roles;

    public function __construct($username, $password, $salt, array $roles)
    {
        $this->username = $username;
        $this->password = $password;
        $this->salt = $salt;
        $this->roles = $roles;
    }

    public function getRoles()
    {
        return $this->roles;
    }

    public function getPassword()
    {
        return $this->password;
    }

    public function getSalt()
    {
        return $this->salt;
    }
    public function getUsername()
    {
        return $this->username;
    }

    public function eraseCredentials()
    {
    }

    public function isEqualTo(UserInterface $user)
    {
        if (!$user instanceof HeaderUser) {
            return false;
        }

        if ($this->username !== $user->getUsername()) {
```

(continues on next page)

(continued from previous page)

```
            return false;
        }

        //if ($this->password !== $user->getPassword()) {
        //    return false;
        //}

        return true;
    }
}
?>
```

3. Define a "header user provider" class relying on the previous class

Create the file src/AppBundle/Security/User/HeaderUserProvider.php :

```
<?php

// src/Security/User/HeaderUserProvider.php
namespace AppBundle\Security\User;

use AppBundle\Security\User\HeaderUser;
use Symfony\Component\Security\Core\User\UserProviderInterface;
use Symfony\Component\Security\Core\User\UserInterface;
use Symfony\Component\Security\Core\Exception\UsernameNotFoundException;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;

class HeaderUserProvider implements UserProviderInterface
{
    public function loadUserByUsername($username)
    {

        if ($username) {

            $password = "dummy";
            $salt = "";
            $roles = array('ROLE_USER');

            return new HeaderUser($username, $password, $salt, $roles);
        }

        throw new UsernameNotFoundException(
            sprintf('Username "%s" does not exist.', $username)
        );
    }

    public function refreshUser(UserInterface $user)
    {
        if (!$user instanceof HeaderUser) {
            throw new UnsupportedUserException(
                sprintf('Instances of "%s" are not supported.', get_class($user))
            );
        }
```

(continues on next page)

(continued from previous page)

```
        return $this->loadUserByUsername($user->getUsername());
    }

    public function supportsClass($class)
    {
        return HeaderUser::class === $class;
    }
}

?>
```

15.38.3 References

- http://symfony.com/doc/current/security/pre_authenticated.html#remote-user-based-authentication
- https://symfony.com/doc/current/security/custom_provider.html

15.39 Sympa

15.39.1 Presentation

Sympa is a mailing list manager.

To configure SSO with Sympa, use **Magic authentication**: a special SSO URL is protected by LL::NG, Sympa will display a button for users who wants to use this feature.

Tip: Since version 1.9 of LLNG, old Auto-Login feature has been removed since it works only with Sympa-5 which has been deprecated

---

15.39.2 Configuration

Sympa configuration

Edit the file "auth.conf", for example:

```
vi /etc/sympa/auth.conf
```

And fill it:

```
generic_sso
        service_name                  Centralized auth service
        service_id                         lemonldapng
        email_http_header            HTTP_MAIL
        netid_http_header             HTTP_AUTH_USER
        internal_email_by_netid    1
        logout_url                         http://sympa.example.com/wws/logout
```

Tip: You can also disable internal Sympa authentication to keep only LemonLDAP::NG by removing user_table paragraph

Note that if you use FastCGI, you must restart Apache to enable changes.

You can also use <portal>?logout=1 as logout_url to remove LemonLDAP::NG session when "disconnect" is chosen.

Sympa virtual host

Configure Sympa virtual host like other *protected virtual host* but protect only magic authentication URL.

Tip: The location URL end is based on the `service_id` defined in Sympa apache configuration.

- For Apache:

```
<VirtualHost *:80>
       ServerName sympa.example.com

       <Location /wws/sso_login/lemonldapng>
       PerlHeaderParserHandler Lemonldap::NG::Handler
       </Location>

       ...

</VirtualHost>
```

- For Nginx:

```
server {
  listen 80;
  server_name sympa.example.com;
  root /path/to/application;
  # Internal authentication request
```

(continues on next page)

(continued from previous page)

```
  location = /lmauth {
    internal;
    include /etc/nginx/fastcgi_params;
    fastcgi_pass unix:/var/run/llng-fastcgi-server/llng-fastcgi.sock;
    # Drop post datas
    fastcgi_pass_request_body  off;
    fastcgi_param CONTENT_LENGTH "";
    # Keep original hostname
    fastcgi_param HOST $http_host;
    # Keep original request (LLNG server will received /llauth)
    fastcgi_param X_ORIGINAL_URI  $original_uri;
  }

  # Client requests
  location /wws/sso_login/lemonldapng {
    auth_request /lmauth;
    set $original_uri $uri$is_args$args;
    auth_request_set $lmremote_user $upstream_http_lm_remote_user;
    auth_request_set $lmlocation $upstream_http_location;
    error_page 401 $lmlocation;
    try_files $uri $uri/ =404;

    ...

    include /etc/lemonldap-ng/nginx-lua-headers.conf;
  }
  location / {
    try_files $uri $uri/ =404;
  }
}
```

Sympa virtual host in Manager

Go to the Manager and *create a new virtual host* for Sympa.

Configure the *access rules* and define the following *headers*:

- Auth-User
- Mail

15.40 Apache Tomcat

Attention: The Tomcat Valve is only available for tomcat 5.5 or greater.

15.40.1 Presentation

Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies.

As J2EE servlet container, Tomcat provides standard security feature, like authentication: the application deployed in Tomcat can delegate its authentication to Tomcat.

By default, Tomcat provides a file called `users.xml` to manage authentication:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
</tomcat-users>
```

LL::NG provides a valve that will check an HTTP header to set the authenticated user on the J2EE container.

15.40.2 Compilation

The sources are available at https://github.com/LemonLDAPNG/lemonldap-valve-tomcat

Required :

- ant
- jre > 1.4
- tomcat >= 5.5

Configure your tomcat home in `build.properties` files.

Attention: Be careful for Windows user, path must contains "/". Example:

```
c:/my hardisk/tomcat/
```

Next run ant command:

```
ant
```

`ValveLemonLDAPNG.jar` is created under `/dist` directory.

15.40.3 Installation

Copy `ValveLemonLDAPNG.jar` in `<TOMCAT_HOME>/server/lib`:

```
cp ValveLemonLDAPNG.jar server/lib/
```

Tip: If needed, you can *recompile the valve from the sources*.

15.40.4 Configuration

Add on your `server.xml` file a new valve entry like this (in host section):

```
<Valve className="org.lemonLDAPNG.SSOValve" userKey="AUTH-USER" roleKey="AUTH-ROLE"␣
↪roleSeparator="," allows="127.0.0.1"/>
```

Configure attributes:

- **userKey**: key in the HTTP header containing user login.
- **roleKey**: key in the HTTP header containing roles. If LL::NG send some roles split by some commas, configure **roleSeparator**.
- **roleSeparator** (optional): role values separator.
- **allows** (optional): Define allowed remote IP (use "," separator for multiple IP). Just set the LL::NG Handler IP on this attribute in order to add more security. If this attribute is missed all hosts are allowed.
- **passThrough** (optional): Allow anonymous access or not. When it takes "false", HTTP headers have to be sent by LL::NG to make authentication. So, if the user is not recognized or HTTP headers not present, a 403 error is sent.

Tip: For debugging, this valve can print some helpful information in debug level. See how configure logging in Tomcat .

15.41 Wekan

15.41.1 Presentation

Wekan is an open-source Kanban, similar to trello.

See the official Wekan website for a complete presentation.

It feature an oauth2 login feature that work with LemonLDAP::NG

15.41.2 Configuring Wekan

Wekan is mostly configured with environement variables, you need to set theses :

- **OAUTH2_ENABLED**: `TRUE`
- **OAUTH2_CLIENT_ID**: `ClientID`
- **OAUTH2_SECRET**: `Secret`
- **OAUTH2_SERVER_URL**: `https://auth.example.com/`
- **OAUTH2_AUTH_ENDPOINT**: `oauth2/authorize`
- **OAUTH2_USERINFO_ENDPOINT**: `oauth2/userinfo`
- **OAUTH2_TOKEN_ENDPOINT**: `oauth2/token`
- **OAUTH2_ID_MAP**: `sub`
- **OAUTH2_USERNAME_MAP**: `sub`
- **OAUTH2_FULLNAME_MAP**: `name`
- **OAUTH2_EMAIL_MAP**: `email`

> **Danger:** Be careful to the / in server_url and endpoints, the complete URL need to be valid, ie auth.example.com/ for url & oauth2/xxx for endpoints, OR, auth.example.com & /oauth2/xxx for endpoints.

Configuring LemonLDAP

We now have to configure LemonLDAP::NG to recognize Wekan as a valid OAuth2 relaying party and send it the information it needs to recognize a user.

Add a *new OpenID Connect relaying party* with the following parameters:

- **Client ID**: the same you set in Wekan configuration (same as OAUTH2_CLIENT_ID)
- **Client Secret**: the same you set in Wekan configuration (same as OAUTH2_SECRET)
- **Add the following exported attributes**
  - `name`: session attribute containing the user's full name
  - `email`: session attribute containing the user's email or _singleMail

_singleMail Macro

Danger: OIDC login fails when an user as a multi-valued email attribute, this need to be fixed on wekan's side, we can bypass that by telling lemonldap to only send one email

Create a new macro, name it (_singleMail is an example), the macro should contain `(split(/; /,$mail))[1]`

15.42 Wiki.js

15.42.1 Presentation

Wiki.js is an open-source wiki.

See the official Wiki.js website for a complete presentation.

It feature an OpenID Connect login that work with LemonLDAP::NG.

15.42.2 Configuring Wiki.js

Connect to your wiki.js instance with an Admin user, in the admin panel, go to "Authentication".

Click on "Add Strategy" and Choose "Generic OpenID Connect / OAuth2".

Choose a Display Name.

Define a Client ID and a Client Secret.

- Authorization Endpoint URL : https://auth.example.com/oauth2/authorize
- Token Endpoint URL : https://auth.example.com/oauth2/token
- User info Endpoint URL : https://auth.example.com/oauth2/userinfo
- Issuer : https://auth.example.com
- Logout URL : https://auth.example.com/oauth2/logout

Make a note of the "Callback URL" and the bottom of the screen and save the configuration.

Configuring LemonLDAP

We now have to configure LemonLDAP::NG to recognize wiki.js as a valid OIDC relying party.

Add a *new OpenID Connect relying party* with the following parameters (Options -> Basic) :

- **Client ID**: the same you set in Wiki.js configuration.
- **Client Secret**: the same you set in Wiki.js configuration.
- **Allowed redirection addresses for login**: The "Callback URL" defined in wiki.js.

Portal with internal CA

Danger: OIDC login fails when your LemonLDAP portal doesn't use a publicaly recognized certificate (Internal Corporate CA), this is because nodejs use it's own keystore. You'll need to pass "NODE_EXTRA_CA_CERTS=" to your wiki installation. If done in docker you will have to create a new image from the official one, add your CA certificates into it, and use the env variable to use it in your wiki.js container.

15.43 Wordpress

15.43.1 Presentation

Wordpress is a famous tool to create websites.

A lot of authentication plugins are available. We propose here to use CAS protocol and WP Cassify plugin.

15.43.2 CAS

Plugin installation

Go in Wordpress admin and install WP Cassify plugin.

Plugin configuration

The full documentation is available on https://wpcassify.wordpress.com/

General settings

Configure CAS server and CAS version:

- CAS Server base url : https://auth.example.com/cas/
- CAS Version protocol: 2

Other options are correct by default.

User Roles Settings

You can assign WP Roles depending on values sent by CAS.

The rules syntax is quite special, you can use it or you can just define macros on LL::NG side and send them through CAS to keep simple rules on WP side.

For example create a macro `role_wordpress_admin` which contains 1 if the user is admin on WP, and send it in CAS attributes.

Then create this rule on WP side:

```
administrator|(CAS{role_wordpress_admin} -EQ "1")
```

15.44 X-Wiki

15.44.1 Presentation

XWiki is a free wiki software platform written in Java with a design emphasis on extensibility. XWiki is an enterprise wiki. It includes WYSIWYG editing, OpenDocument based document import/export, semantic annotations and tagging, and advanced permissions management.

15.44.2 Configuration

The integration with LL::NG is the following:

- LemonLDAP::NG is configured as a reverse-proxy for xwiki
- Xwiki is configured to accept HTTP Headers

Xwiki virtual host

Apache

You will configure Xwiki virtual host like other *protected virtual host*.

This is an example, with https and speaking to xwiki via AJP.

```
<VirtualHost *:80>
    ServerName wiki.acme.fr
    Redirect / https://wiki.acme.fr/
</VirtualHost>
```

(continues on next page)

(continued from previous page)

```
<VirtualHost *:443>
    ServerName wiki.acme.fr

    SSLEngine On
    SSLCertificateFile /etc/pki/tls/certs/wildcard.acme.fr.crt
    SSLCertificateKeyFile /etc/pki/tls/certs/wildcard.acme.fr.key
    SSLCertificateChainFile /etc/pki/tls/certs/CLASS_2_ACME_CA.crt
    SSLOptions +StdEnvVars
    SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
    SSLCipherSuite          ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
↪POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-
↪GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-
↪SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-
↪RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-
↪SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
↪SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-
↪CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-
↪SHA:AES256-SHA:DES-CBC3-SHA:!DSS
    SSLHonorCipherOrder     on
    SSLCompression          off

    PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2

    RewriteEngine on
    RewriteRule ^/$ /xwiki/ [R]

    ProxyPreserveHost On
    ProxyRequests On

    ProxyPass / ajp://192.168.11.130:8009/
    ProxyPassReverse / ajp://192.168.11.130:8009/

    ErrorLog /var/log/httpd/wiki_error.log
    CustomLog /var/log/httpd/wiki_access.log combined
</VirtualHost>
```

Xwiki virtual host in Manager

Go to the Manager and *create a new virtual host* for Xwiki.

Configure the *access rules*.

Configure the *headers*:

- remote_user: $uid
- remote_groups: encode_base64($groups,'')

Xwiki Configuration

```
xwiki.authentication.authclass=org.xwiki.contrib.authentication.XWikiTrustedAuthenticator
xwiki.authentication.trusted.adapterHint=headers
xwiki.authentication.trusted.auth_field=remote_user
xwiki.authentication.trusted.group_field=remote_groups
xwiki.authentication.trusted.logout_url=https://auth.acme.fr/#logout
```

15.45 Zimbra

15.45.1 Presentation

Zimbra is open source server software for email and collaboration - email, group calendar, contacts, instant messaging, file storage and web document management. The Zimbra email and calendar server is available for Linux, Mac OS X and virtualization platforms. Zimbra syncs to smartphones (iPhone, BlackBerry) and desktop clients like Outlook and Thunderbird. Zimbra also features archiving and discovery for compliance. Zimbra can be deployed on-premises or as a hosted email solution.

Zimbra use a specific preauthentication protocol to provide SSO on its application. This protocol is implemented in an LL::NG specific Handler.

Tip: Zimbra can also be connected to LL::NG via *SAML protocol* (see Zimbra blog).

15.45.2 Configuration

The integration with LL::NG is the following:

- A special URL is declared in application menu (like http://zimbra.example.com/zimbrasso)
- A Zimbra Handler is called
- Handler build the preauth request and redirect user on Zimbra preauth URL
- Then Zimbra do the SSO by setting a cookie in user's browser

Zimbra preauth key

You need to get a preauth key from Zimbra server.

See how to do this on Zimbra wiki.

Zimbra application in menu

Choose for example http://zimbra.example.com/zimbrasso as SSO URL and *set it in application menu*.

Zimbra virtual host

You just have to set "Type: ZimbraPreAuth" in virtualhost options and reload configuration in this handler.

Zimbra Handler parameters

Zimbra parameters are the following:

- **Preauthentication key**: the one you grab from zmprov command
- **Account session key**: session field used as Zimbra user account (by default: uid)
- **Account type**: for Zimbra this can be name, id or foreignKey (by default: id)
- **Preauthentication URL**: Zimbra preauthentication URL, either with full URL (ex: http://zimbra.lan/service/preauth), either only with path (ex: /service/preauth) (by default: /service/preauth)
- **Local SSO URL pattern**: regular expression to match the SSO URL (by default: ^/zimbrasso$)

Attention: Due to Handler API change in 1.9, you need to set these attributes in `lemonldap-ng.ini` and not in Manager, for example:

```
[handler]
zimbraPreAuthKey = XXXX
zimbraAccountKey = uid
zimbraBy =id
zimbraUrl = /service/preauth
zimbraSsoUrl = ^/zimbrasso$
```

Multi-domain issues

Some organizations have multiple zimbra domains:

1. foo@domain1.com
2. bar@domain2.com

However, the zimbra preauth key is:

- generated for one zimbra domain only
- declared globally for every LemonLDAP::NG virtual hosts.

Thus, if domain1 has been registered on LemonLDAP::NG, user bar won't be able to connect to zimbra because preauth key is different. If you accept to have the same preauth key for all zimbra domains, you can set the same preauth key using this procedure:

We are going to use the first key (the domain1 one) for every domain. On Zimbra machine, generate the keys:

```
zmprov generateDomainPreAuthKey domain1.com
preAuthKey: 4e2816f16c44fab20ecdee39fb850c3b0bb54d03f1d8e073aaea376a4f407f0c

zmprov generateDomainPreAuthKey domain2.com
preAuthKey: 6b7ead4bd425836e8cf0079cd6c1a05acc127acd07c8ee4b61023e19250e929c
```

Then, connect to your zimbra LDAP server with your favourite tool (Apache Directory Studio can do the job). Take care to connect with the super admin and password account.

- Expand the branch "dc=com", then click the "dc=domain1" branch
- Get the value of zimbraPreAuthKey
- Expand the branch "dc=com", then click the "dc=domain2" branch
- Replace the value of zimbraPreAuthKey you have previously copied
- Wait for all Zimbra servers to update, or restart the zcs server

That's it, all zimbra servers will be able to decipher the hmac because they share the same key!

15.46 How to integrate

To integrate a Web application in LL::NG, you have the following possibilities:

- Protect the application with the Handler, and push user identity through HTTP headers. This is how main Access Manager products, like CA SiteMinder, are working. This also how Apache authentication modules are working, so if your application is compatible with Apache authentication (often called "external authentifcation"), then you can use the Handler.
- Specific Handler: some applications can require a specific Handler, to manage preauthentication process for example.
- CAS: your application is a CAS client, you can configure LL::NG as a *CAS server*.
- SAML: your application is a SAML Service Provider, you can configure LL::NG as a *SAML Identity Provider*.
- OpenID Connect: your application is a OpenID Connect Relying Party, you can configure LL::NG as a *OpenID Connect Provider*.

If none of above methods is available, you can try:

- *HTTP Auth-Basic*: replay Auth Basic authentication
- *Form replay*: replay form based authentication

15.47 Application list

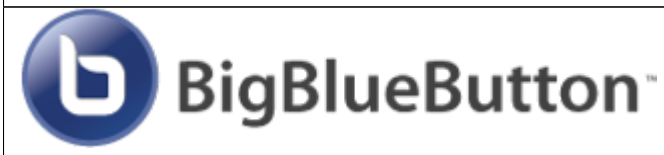
Application	Configuratio
	*ADFS*
	*Alfresco*
	*Amazon Web*
	*AWX (Ansible*
	*BigBlueButto*
	*Bugzilla*
	*Cornerstone*

Table 1 – continued from previous page

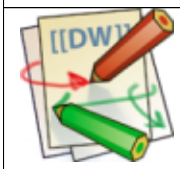
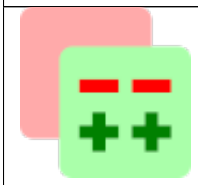
Application	Configuratio
	*Discourse*
	*Django*
	*Dokuwiki*
	*Drupal*
	*FusionDirect*
	*Gerrit*
	*Gitlab*
	*GLPI*

Table 1 – continued from previous page

Application	Configuratio
	*Google Apps*
	*Grafana*
	*GRR*
	*Apache Guac*
	*HumHub*
	*i-Parapheur*
	*Jitsi Meet*
	*Liferay*

Table 1 – continued from previous page

Application	Configuratio
	*LimeSurvey*
	*Mattermost*
	*Mediawiki*
	*Mobilizon*
	*NextCloud*
	*OBM*
	*Office 365*

Table 1 – continued from previous page

Application	Configuratio
	*Publik*
	*phpLDAPAdm*
	*Roundcube*
	*SalesForce*
	SAP
	*simpleSAMLp*
	*Spring*

Table 1 – continued from previous page

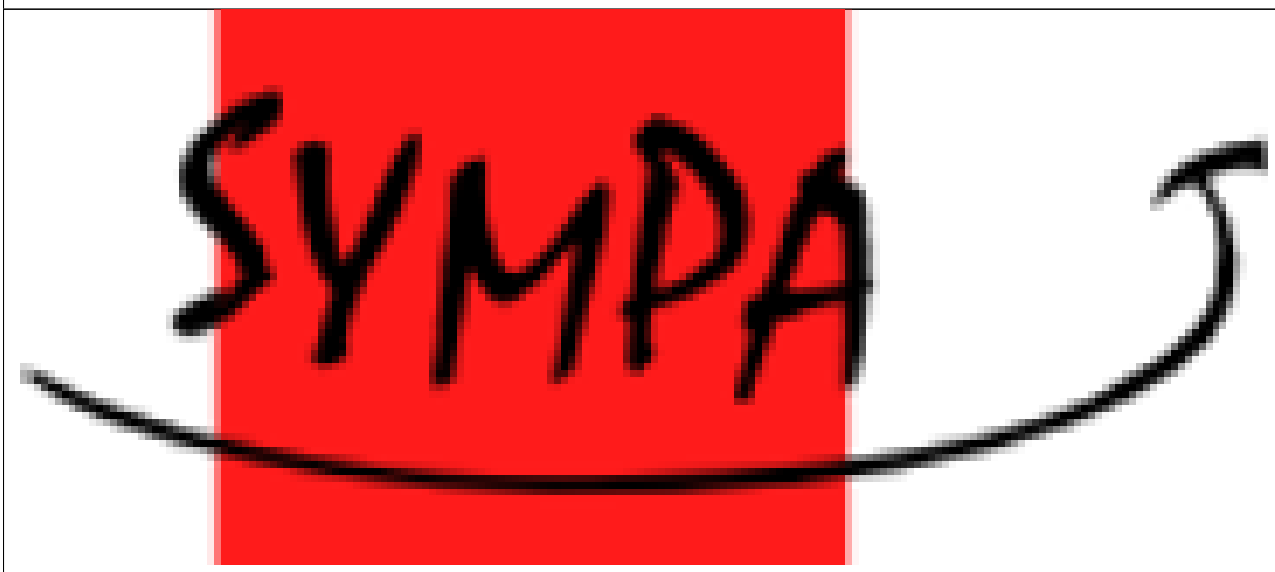
Application	Configuratio
	*Symfony*
	*Sympa*
	*Tomcat*
	*Wekan*
	Wiki.js

Table 1 – continued from previous page

Application	Configuratio
	*Wordpress*
	*XWiki*
	*Zimbra*

CHAPTER

SIXTEEN

ADVANCED FEATURES

16.1 SMTP server setup

Go in `General Parameters` > `Advanced Parameters` > `SMTP`:

- **Session key containing mail address**: choose which session field contains mail address
- **SMTP Server**: IP or hostname of the SMTP server
- **SMTP Port**: Port of the SMTP server
- **SMTP User**: SMTP user if authentication is required
- **SMTP Password**: SMTP password if authentication is required
- **SSL/TLS protocol** and **SSL/TLS options**: Here you can enable SMTPS or startTLS. A list of possible options can be found in the IO::Socket::SSL documentation.

Tip:

- If no SMTP server is configured, the mail will be sent via the local sendmail program. Else, Net::SMTP module is required to use the SMTP server
- The SMTP server value can hold the port, for example: `mail.example.com:25`

Warning:

- Older versions of the Email::Sender library have limitations when it comes to SMTPS or STARTTLS support. Versions lower than 1.300027 will not be able to check the remote server certificate or use custom IO::Socket::SSL options.

- **Mail headers**:
  - **Mail sender**: address seen in the "From" field (default: noreply@[DOMAIN])
  - **Reply address**: address seen in the "Reply-To" field
  - **charset**: Charset used for the body of the mail (default: utf-8)

16.1.1 Testing your email setup

New in version 2.0.10.

You can test your email setup in the `General Parameters` > `Advanced Parameters` > `SMTP` page by using the `Send test email` button in the manager.

Tip: You need to save your SMTP configuration before you can test it

New in version 2.0.10.

You can also test your email setup using the `test-email` command in the CLI

```
lemonldap-ng-cli test-email dwho@badwolf.org
```

16.2 Store user password in session

16.2.1 Presentation

Password is not a common attribute. Indeed, in most of the cases, it is not stored in clear text in the backend (LDAP or database).

So, to keep user password in session, you cannot just export the password variable in session. To bypass this, LL::NG can remember what password was given by user on authentication phase.

Attention:

- As this may be a security hole, password store in session is not activated by default
- This mechanism can only work with authentication backends using a login/password form (*LDAP*, *DBI*, ...)

16.2.2 Configuration

Go in Manager, `General Parameters` » ''Sessions '' » `Store user password in session data` and set to `On`.

16.2.3 Usage

User password is now available in `$_password` variable. For example, to send it in an header:

```
Auth-Password => $_password
```

Tip: For security reasons, the password is not shown in sessions explorer.

16.3 RBAC model

16.3.1 Presentation

RBAC stands for Role Based Access Control. It means that you manage authorizations to access applications by checking the role(s) of the user, and provide this role to the application.

As the definition of access rules is free in LemonLDAP::NG, you can implement an RBAC model if you need.

16.3.2 Configuration

Roles as simple values of a user attribute

Imagine you've set your directory schema to store roles as values of an attribute of the user, for example "description". This is simple because you can send the role to the application by creating a HTTP header (for example Auth-Role) with the concatenated values (';' is the concatenation string):

```
Auth-Roles => $description
```

If the user has these values inside its entry:

```
description: user
description: admin
```

Then you got this value inside the Auth-Roles header:

```
user; admin
```

Roles as entries in the directory

Now imagine the following DIT:

- dc=example,dc=com
  - ou=users
    - uid=coudot
  - ou=roles
    - ou=aaa
      - cn=admin
      - cn=user
    - ou=bbb
      - cn=admin
      - cn=user

Roles are entries, below branches representing applications. We can use the standard LDAP objectClass `organizationalRole` to maintain roles, for example:

```
dn: cn=admin,ou=aaa,ou=roles,dc=example,dc=com
objectClass: organizationalRole
objectClass: top
cn: admin
ou: aaa
roleOccupant: uid=coudot,ou=users,dc=example,dc=com
```

A user is attached to a role if its DN is in `roleOccupant` attribute. We add the attribute `ou` to allow LL::NG to know which application is concerned by this role.

So imagine the user coudot is "user" on application "BBB" and "admin" on application "AAA".

Gather roles in session

Use the *LDAP group* configuration to store roles as groups in the user session:

- Base: ou=roles,dc=example,dc=com
- Object class: organizationalRole
- Target attribute: roleOccupant
- Searched attributes: cn ou

Restrict access to application

We configure LL::NG to authorize people on an application only if they have a role on it. For this, we use the `$hGroups` variable.

- For application AAA:

```
default => groupMatch($hGroups, 'ou', 'aaa')
```

- For application BBB:

```
default => groupMatch($hGroups, 'ou', 'bbb')
```

Send role to application

It is done by creating the correct HTTP header:

- For application AAA:

```
Auth-Roles => ((grep{/aaa/} split(';',$groups))[0] =~ /([a-zA-Z]+?)/)[0]
```

- For application BBB:

```
Auth-Roles => ((grep{/bbb/} split(';',$groups))[0] =~ /([a-zA-Z]+?)/)[0]
```

16.4 Custom functions

Custom functions allow one to extend LL::NG, they can be used in *Headers*, *Rules* or *form replay data*. Two actions are needed:

- declare them in LLNG configuration
- load the relevant code

16.4.1 Implementation

Your perl custom function must be declared on appropriate server when separating :

portal type : declare custom function here when using it in rules, macros, menu

reverse-proxy type : declare custom function here when using it in headers

16.4.2 Write custom functions library

Create your Perl module with custom functions. You can name your module as you want, for example `SSOExtensions.pm`:

```
vi /path/to/SSOExtensions.pm
```

```
package SSOExtensions;

sub function1 {
  my (@args) = @_;

  # Your nice code here
  return $result;
}

sub function2 {
  return $_[0];
}

1;
```

16.4.3 Import custom functions in LemonLDAP::NG

Load relevant code in handler server

New method

Just declare files or Perl module that must be loaded:

```
[all]
require = /path/to/functions.pl, /path/to/SSOExtensions.pm
# OR
require = SSOExtensions::function1, SSOExtensions::function2
```

(continues on next page)

(continued from previous page)

```
; Prevent Portal to crash if Perl module is not found
;requireDontDie = 1
```

Old method

Danger: This method is available but unusable by Portal under Apache. So if your rule may be used by the menu, use the new method.

Apache

Your module has to be loaded by Apache (for example after Handler load):

```
# Perl environment
PerlRequire Lemonldap::NG::Handler
PerlRequire /path/to/SSOExtensions.pm
PerlOptions +GlobalRequest
```

FastCGI server (Nginx)

You've just to incicate to *LLNG FastCGI server* the file to read using either `-f` option or `CUSTOM_FUNCTIONS_FILE` environment variable. Using packages, you just have to modify your `/etc/default/llng-fastcgi-server` (or `/etc/default/lemonldap-ng-fastcgi-server`) file:

```
# Number of process (default: 7)
#NPROC = 7

# Unix socket to listen to
SOCKET=/var/run/llng-fastcgi-server/llng-fastcgi.sock

# Pid file
PID=/var/run/llng-fastcgi-server/llng-fastcgi-server.pid

# User and GROUP
USER=www-data
GROUP=www-data

# Custom functions file
CUSTOM_FUNCTIONS_FILE=/path/to/SSOExtensions.pm
```

Declare custom functions

Go in Manager, `General Parameters` » `Advanced Parameters` » `Custom functions` and set:

```
SSOExtensions::function1 SSOExtensions::function2
```

> **Attention:** If your function is not compliant with *Safe jail*, you will need to disable the jail.

16.4.4 Use it

You can now use your function in a macro, an header or an access rule, for example:

```
SSOExtensions::function1( $uid, $ENV{REMOTE_ADDR} )
```

16.5 Extended functions

16.5.1 Presentation

When *writing rules and headers*, you can use Perl expressions that will be evaluated in a jail, to prevent bad code execution.

This is also true for:

- *Menu modules activation rules*
- *Form replay data*
- Macros
- Issuer databases use rules
- etc.

Inside this jail, you can access to:

- all session values and CGI environment variables (through *$ENV{<HTTP_NAME>}*)
- Core Perl subroutines (split, pop, map, etc.)
- *Custom functions*
- The encode_base64 subroutine
- Information about current request
- Extended functions:
  - *basic*
  - *checkDate*
  - *checkLogonHours*
  - *date*
  - *dateToTime* (NEW in version 2.0.12)
  - *encrypt*

- *groupMatch*
- *has2f* (NEW in version 2.0.10)
- *inGroup* (NEW in version 2.0.8)
- *isInNet6*
- *iso2unicode*
- *listMatch* (NEW in version 2.0.7)
- *token*
- *unicode2iso*
- *varIsInUri* (NEW in version 2.0.7)

Tip: To know more about the jail, check Safe module documentation.

16.5.2 Extended Functions List

date

Returns the date, in format YYYYMMDDHHMMSS, local time by default, GMT by calling `date(1)`

```
For example: date(1) lt '19551018080000'
```

dateToTime

New in version 2.0.12.

Converts a string date into epoch time.

The date format is the LDAP date syntax, for example for the 1st March 2009 (GMT):

```
20090301000000Z
```

The date may end with a differential timezone that is interpreted to adjust the epoch time, for example for the 1st March 2009 (+0100):

```
20090301000000+0100
```

Simple usage example:

```
dateToTime($ssoStartDate) lt dateToTime(date(1))
```

checkLogonHours

This function will check the day and the hour of current request, and compare it to allowed days and hours. It returns 1 if this match, 0 else. By default, the allowed days and hours is an hexadecimal value, representing each hour of the week. A day has 24 hours, and a week 7 days, so the value contains 168 bits, converted into 42 hexadecimal characters. Sunday is the first day.

For example, for a full access, excepted week-end:

```
000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF000000
```

Tip: You can use the binary value from the logonHours attribute of Active Directory, or create a custom attribute in your LDAP schema.

Functions parameters:

- **logon_hours**: string representing allowed logon hours (GMT)
- **syntax** (optional): `hexadecimal` (default) or `octetstring`
- **time_correction** (optional): hours to add or to subtract
- **default_access** (optional): what result to return if **logon_hours** is empty

Simple usage example:

```
checkLogonHours($ssoLogonHours)
```

If you use the binary value (Active Directory), use this:

```
checkLogonHours($ssoLogonHours, 'octetstring')
```

You can also configure jetlag (if all of your users use the same timezone):

```
checkLogonHours($ssoLogonHours, '', '+2')
```

If you manage different timezones, you have to take the jetlag into account in ssoLogonHours values, or use the $_timezone parameter. This parameter is set by the portal and use javascript to get the connected user timezone. It should works on every browser:

```
checkLogonHours($ssoLogonHours, '', $_timezone)
```

You can modify the default behavior for people without value in ssoLogonHours. Indeed, by default, users without logon hours values are rejected. You can allow these users instead of reject them:

```
checkLogonHours($ssoLogonHours, '', '', '1')
```

checkDate

This function will check the date of current request, and compare it to a start date and an end date. It returns 1 if this match, 0 else.

The date format is the LDAP date syntax, for example for the 1st of March 2009 (GMT)

```
20090301000000Z
```

NEW Since version 2.0.12, the date may end with a differential timezone, for example for the 1st of March 2009 (+0100):

```
20090301000000+0100
```

Functions parameters:

- **start**: Start date (GMT unless, NEW since version 2.0.12, a differential timezone is included)
- **end**: End date (GMT unless, NEW since version 2.0.12, a differential timezone is included)
- **default_access** (optional): what result to return if **start** and **end** are empty

Simple usage example:

```
checkDate($ssoStartDate, $ssoEndDate)
```

basic

> **Attention:** This function is not compliant with *Safe jail*, you will need to disable the jail to use it.

This function builds the `Authorization` HTTP header used in *HTTP Basic authentication scheme*. It will force conversion from UTF-8 to ISO-8859-1 of user and password data.

Functions parameters:

- **user**
- **password**

Simple usage example:

```
basic($uid,$_password)
```

unicode2iso

> **Attention:** This function is not compliant with *Safe jail*, you will need to disable the jail to use it.

This function convert a string from UTF-8 to ISO-8859-1.

Functions parameters:

- **string**

Simple usage example:

```
unicode2iso($name)
```

iso2unicode

Attention: This function is not compliant with *Safe jail*, you will need to disable the jail to use it.

This function convert a string from ISO-8859-1 to UTF-8.

Functions parameters:

- **string**

Simple usage example:

```
iso2unicode($name)
```

groupMatch

this function allows one to parse the `$hGroups` variable to check if a value is present inside a group attribute.

Function parameter:

- **groups**: `$hGroups` variable
- **attribute**: Name of group attribute
- **value**: Value to check

Simple usage example:

```
groupMatch($hGroups, 'description', 'Service 1')
```

has2f

New in version 2.0.10.

This function tests if the current user has registered a second factor. The following types are supported:

- *TOTP*
- *U2F*
- *UBK*

Example:

```
has2f()
has2f('UBK')
has2f('UBK') or has2f('TOTP')
```

Warning: Do **NOT** use this test to check if the user has **used** their second factor for logging in! This test only checks if the user has registered a second factor. Regardless of their **current** authentication level. It can be used to simplify second factor activation rules.

Note: Before version 2.0.10, you need to use the following syntax

```
$_2fDevices =~ /"type":\s*"TOTP"/s
```

listMatch

New in version 2.0.7.

This function lets you test if a particular value can be found with a multi-valued session attribute.

Function parameter:

- **list**: Variable containing several values (plain string with separator, array or hash)
- **value**: Value to search in the list
- **ignorecase**: Ignore case, by default the search is case-sensitive

Simple usage example:

```
# Case sensitive match
listMatch($roles, 'role-app1')

# Case insensitive match
listMatch($roles, 'RoLe-aPp1', 1)
```

The function returns 1 if the value was found, and 0 if it was not found.

inGroup

New in version 2.0.8.

This function lets you test if the user is in a given group. It is case-insensitive.

Usage example:

```
inGroup('admins')

inGroup('test users')
```

The function returns 1 if the user belongs to the given group, and 0 if they don't.

encrypt

Tip: Since version 2.0, this function is now compliant with *Safe jail*.

This function uses the secret key of LLNG configuration to crypt a data. This can be used for anonymizing identifier given to the protected application.

```
encrypt($_whatToTrace)
```

token

This function generates token used for *handling server webservice calls*.

```
token($_session_id,'webapp1.example.com','webapp2.example.com')
```

isInNet6

Function to check if an IPv6 address is in a subnet. Example *check if IP address is local*:

```
isInNet6($ipAddr, 'fe80::/10')
```

varIsInUri

New in version 2.0.7.

Function to check if a variable is in requested URI

Example *check if $uid is in /check-auth/ URI*:

```
varIsInUri($ENV{REQUEST_URI}, '/check-auth/', $uid)

https://test1.example.com/check-auth/dwho     -> true
https://test1.example.com/check-auth/dwho/api -> true
https://test1.example.com/check-auth/dwh      -> false
```

* You can set "restricted" flag to match exact URI:

```
varIsInUri($ENV{REQUEST_URI}, '/check-auth/', "$uid/", 1)

https://test1.example.com/check-auth/rtyler/     -> true
https://test1.example.com/check-auth/rtyler/api  -> false
https://test1.example.com/check-auth/rtyler      -> false
```

16.6 Register a new account

16.6.1 Presentation

This feature is a page that allows a user to create an account. The steps are the following:

1. User click on the button "Create a new account"
2. He enters first name, last name and email
3. He gets a mail with a confirmation link
4. After clicking, his entry is added
5. He gets a mail with his login and his password

16.6.2 Configuration

You can enable the "Create your account" button in *portal customization parameters*.

Then, go in `Portal` > `Advanced parameters` > `Register new account`:

- **Module**: Choose the backend to use to create the new account.
- **Page URL**: URL of register page
- **Validity time of a register request**: duration in seconds of a new account request. The request will be deleted after this time if user do not click on the link.
- **Subject for confirmation mail**: Subject of the mail containing the confirmation link
- **Subject for done mail**: Subject of the mail giving login and password

16.7 Logout forward

16.7.1 Presentation

Even if LL:NG can catch logout URL through *virtual host rules*, you can have the need to forward a logout to other applications, to close their local sessions.

LL::NG has a logout forward mechanism, that will add a step in logout process, to send logout requests (indeed, GET requests on application logout URL) inside hidden iframes.

Tip: The logout request will be sent even if the user did not use the application.

16.7.2 Configuration

Go in Manager, `General parameters` » `Advanced parameters` » `Logout forward` and click on `Add a key`, then fill:

- **Key**: application name
- **Value**: application logout URL

Attention: The request on logout URL will be sent after user is disconnected, so you should unprotect this URL if it is protected by an LL::NG Handler.

16.8 FastCGI support

Attention: Since 2.0, all LLNG components run under FastCGI

16.9 LemonLDAP::NG FastCGI server

Since 1.9, Lemonldap::NG provides a FastCGI server usable to protect applications with Nginx (See *Manage virtual hosts* page to configure virtual hosts).

This FastCGI server can be used for all LLNG components. It compiles enabled components on-the-fly.

16.9.1 Start

Using packages

You just have to install lemonldap-ng-fastcgi-server package, it will be started automatically.

Using "make install"

To enable the FastCGI server at startup, copy the script `llng-fastcgi-server` installed in INITDIR (default `/usr/local/lemonldap-ng/etc/init.d/`) in `/etc/init.d` and enable it (links to `/etc/rc<x>.d`).

16.9.2 Configuration

FastCGI server has few parameters. They can be set by environment variables (read by startup script) or by command line options. A default configuration file can be found in `/usr/local/lemonlda-ng/etc/default/llng-fastcgi-server` (or `/etc/default/lemonldap-ng-fastcgi-server` in Debian package).

The FastCGI server reads also `LLTYPE` parameter in FastCGI requests (see portal-nginx.conf or manager-nginx.conf) to choose which module is called:

- `cgi` for the portal (or any CGI: it works like PHP-FPM for Perl !)
- `manager` for the manager
- `status` to see statistics (if enabled)

if `LLTYPE` is set to another value or not set, FastCGI server works as handler.

16.10 Advanced PSGI usage

LLNG is build on Plack, so it can be used with any compatible server:

- Starman
- Twiggy
- Twiggy::Prefork
- Starman
- uWSGI using uWSGI PSGI plugin
- **Alternative**: *Node.js handler* can be used as FastCGI server, only for application protection

uWSGI or *Node.js FastCGI server* may provide the highest performance.

16.10.1 FastCGI server replacement

A `llng-server.psgi` is provided in example directory. It is designed to replace exactly FastCGI server. You can use it :

- with a FCGI Plack server, but you just have to change llng-fastcgi-server engine *(in /etc/default/lemonldap-ng-fastcgi-server)* to have the same result. Available engines:
  - FCGI **(default)**. It can use the following managers:
    - FCGI::ProcManager (default)
    - FCGI::ProcManager::Constrained
    - FCGI::ProcManager::Dynamic
  - AnyEvent::FCGI
  - FCGI::EV
  - FCGI::Engine
  - FCGI::Engine::ProcManager
  - FCGI::Async
- with uWSGI **(see below)**

Attention: Starman, Twiggy,... are HTTP servers, not FastCGI ones !

You can also replace only a part of it to create a specialized FastCGI server (portal,...). Look at `llng-server.psgi` example and take the part you want to use.

There are also some other psgi files in examples directory.

LLNG FastCGI Server

`llng-fastcgi-server` can be launched with the following options:

Command-line options		Environment variable	Explanation
Short	Long		
-p	–pid	PID	Process PID
-u	–user	USER	Unix uid
-g	–group	GROUP	Unix gid
-n	–proc	NPROC	Number of process to launch *(FCGI::ProcManager)*
-s	–socket	SOCKET	Socket to listen to
-l	–listen	LISTEN	Listening address. Examples: `host:port`, `:port`, `/socket/path`
-f	–customFunctionsFile	CUSTOM_FUNCTIONS_FILE	File to load for custom functions
-e	–engine	ENGINE	Plack::Handler engine, default to FCGI *(see below)*
	–plackOptions		Other options to path to Plack. Can bu multi-valued. Values must look like `--key=value`

See `llng-fastcgi-server(1)` manpage.

Some examples

FCGI with FCGI::ProcManager::Constrained

```
llng-fastcgi-server -u nobody -g nobody -s /run/llng.sock -n 10 -e FCGI \
                    --plackOptions=--manager=FCGI::ProcManager::Constrained
```

FCGI::Engine::ProcManager

```
llng-fastcgi-server -u nobody -g nobody -s /run/llng.sock -n 10 \
                    -e FCGI::Engine::ProcManager
```

Using uWSGI

You must install uWSGI PSGI plugin. Then for example, launch llng-server.psgi *(simple example)*:

```
/usr/bin/uwsgi --plugins psgi --socket :5000 --uid www-data --gid www-data --psgi /usr/
↪share/lemonldap-ng/llng-server/llng-server.psgi
```

You will find in LLNG Nginx configuration files some comments that explain how to configure Nginx to use uWSGI instead of LLNG FastCGI server.

Using Debian lemonldap-ng-uwsgi-app package

lemonldap-ng-uwsgi-app installs a uWSGI application: `/etc/uwsgi/apps-available/llng-server.yaml`. To enable it, link it in `apps-enabled` and restart your uWSGI daemon:

```
apt-get install uwsgi uwsgi-plugin-psgi
cd /etc/uwsgi/apps-enabled
ln -s ../apps-available/llng-server.yaml
service uwsgi restart
```

Then adapt your Nginx configuration to use this uWSGI app.

Configuration

To serve large requests with uWSGI, you could have to modify in uWSGI and/or Nginx init files several options. Example:

```
workers = 4
buffer-size = 65535
limit-post = 0
```

```
client_max_body_size 300M;
proxy_send_timeout 600;
proxy_read_timeout 600;
proxy_connect_timeout 600;
uwsgi_read_timeout 120;
uwsgi_send_timeout 120;
```

Note: Nginx natively includes support for upstream servers speaking the uwsgi protocol since version 0.8.40. To improve performances, you can switch from a TCP socket to an Unix Domain Socket by editing `llng-server.yaml`:

```
uwsgi:
        plugins: psgi
        socket: /tmp/uwsgi.sock
```

and adapting Nignx configuration files:

```
# OR TO USE uWSGI
include /etc/nginx/uwsgi_params;
uwsgi_pass unix:///tmp/uwsgi.sock;
uwsgi_param LLTYPE psgi;
uwsgi_param SCRIPT_FILENAME $document_root$sc;
uwsgi_param SCRIPT_NAME $sc;
# Uncomment this if you use Auth SSL:
#uwsgi_param  SSL_CLIENT_S_DN_CN $ssl_client_s_dn_cn;
```

16.10.2 Protect a PSGI application

LLNG provides `Plack::Middleware::Auth::LemonldapNG` that can be used to protect any PSGI application: it acts exactly like a LLNG handler. Simple example:

```
use Plack::Builder;

my $app   = sub { ... };
builder {
    enable "Auth::LemonldapNG";
    $app;
};
```

More advanced example:

```
use Plack::Builder;

my $app   = sub { ... };

# Optionally ($proposedResponse is the PSGI response of Lemonldap::NG handler)
sub on_reject {
    my($self,$env,$proposedResponse) = @_;
    # ...
}

builder {
    enable "Auth::LemonldapNG",
      llparams => {
        # ...
      },
      on_reject => \&on_reject;
    $app;
};
```

16.11 Ignore some manager tests

Each time you save a configuration, Manager launch a lot of tests:

- unit tests for each key: they are declared in Lemonldap::NG::Manager::Attributes *(source Lemonldap::NG::Manager::Build::Attributes)*
- more advanced tests declared in Lemonldap::NG::Manager::Conf::Tests

In some case *(conf overridden in INI file,...)*, you may have to ignore some of them. You just have to list them *(space separated)* in a special key in `lemonldap-ng.ini`, section `[Manager]`:

- `skippedUnitTests` for unit tests
- `skippedGlobalTests` for global tests

Example:

```
[Manager]
skippedUnitTests   = grantSessionRules portalSkinRules
skippedGlobalTests = testApacheSession
```

16.12 Rules examples

This page contains a few useful Perl expressions you can use in your *Handler rules*, SAML/OIDC/CAS security rules, 2FA Activation rules, etc.

16.12.1 Using session attributes

Session attributes are visible in the Manager's Session browser, any attribute you see there can be used in a rule!

- Restricting access to a single user:

```
$uid eq "dwho"
$uidNumber == 1000
$cn eq "Doctor Who"
$email eq "dwho@badwolf.org"
etc.
```

Tip: In Perl, `eq` means *Equal* and must be used on strings. `==` should be used only on numbers

Danger: In Perl, `@` character means an array and `%` a hash! If you want to write a macro with these characters, you have to escape them like this:

```
$my_email = "$uid\@my-domain.com"
$percent = "$rate\%more"
```

- Restricting access to specific groups

```
$groups =~ /\b(?:admins|su)\b/ # admins OR su
$groups =~ /\badmin_[1-3a]\b/ # admin_1 OR admin_2 OR admin_3 OR admin_a

defined $hGroups->{'administrators'}

# 2.0.8 and higher only
inGroup('administrators')
```

- Combining multiple expressions

```
inGroup('timelords') and not $uid eq 'missy'
```

- Using Perl's regular expressions

```
$cn =~ /^Doctor.*/i
$email !~ /@spam.com$/
```

- Filtering on Authentication Level

```
$authenticationLevel >= 3
```

- Filtering on Authentication method

```
$_auth ne 'Demo'
```

- Checking if the user has a an **available** second factor.

```
# Since 2.0.10
has2f()
has2f('TOTP')
has2f('TOTP') or has2f('U2F')

# Before 2.0.10
$_2fDevices =~ /"type":\s*"TOTP"/s
```

Tip: In Perl, `ne` means *Not Equal* and must be used on strings. `\b` means *word Boundary*. (?:) means *non capturing* parenthesis.

16.12.2 Using environment variables

- Comparing the IP address

```
$env->{REMOTE_ADDR} =~ /^10\./
```

- Comparing requested URI

```
$env->{REQUEST_URI} =~ /test/
```

16.13 Parameter list

Tip: Click on a column header to sort table. The attribute key name can be used directly in `lemonldap-ng.ini` or in Perl scripts to override configuration parameters (see *configuration location*).

16.13.1 Main parameters

Key name	Documentation
ADPwdExpireWarning	AD password expire warning
ADPwdMaxAge	AD password max age
AuthLDAPFilter	LDAP filter for auth search
LDAPFilter	Default LDAP filter
SMTPAuthPass	Password to use to send mails
SMTPAuthUser	Login to use to send mails
SMTPPort	Fix SMTP port
SMTPServer	SMTP Server
SMTPTLS	TLS protocol to use with SMTP
SMTPTLSOpts	TLS/SSL options for SMTP
SSLAuthnLevel	SSL authentication level
SSLVar	
SSLVarIf	
activeTimer	Enable timers on portal pages
adaptativeAuthenticationLevelRules	Adaptative authentication level rules
apacheAuthnLevel	Apache authentication level
applicationList	Applications list
authChoiceAuthBasic	Auth module used by AuthBasic handler
authChoiceFindUser	Auth module used by FindUser plugin
authChoiceModules	Hash list of Choice strings
authChoiceParam	Applications list
authentication	Authentication module
autoSigninRules	List of auto signin rules
available2F	Available second factor modules
available2FSelfRegistration	Available self-registration modules for second factor
avoidAssignment	Avoid assignment in expressions
browsersDontStorePassword	Avoid browsers to store users password
bruteForceProtection	Enable brute force attack protection
bruteForceProtectionIncrementalTempo	Enable incremental lock time for brute force attack protection
bruteForceProtectionLockTimes	Incremental lock time values for brute force attack protection
bruteForceProtectionMaxAge	Max age between current and first failed login
bruteForceProtectionMaxFailed	Max allowed failed login
bruteForceProtectionMaxLockTime	Max lock time
bruteForceProtectionTempo	Lock time
captcha_login_enabled	Captcha on login page
captcha_mail_enabled	Captcha on password reset page
captcha_register_enabled	Captcha on account creation page
captcha_size	Captcha size
casAccessControlPolicy	CAS access control policy

Table 1 – continued from previous page

Key name	Documentation
casAppMetaDataOptions	Root of CAS app options
casAttr	Pivot attribute for CAS
casAttributes	CAS exported attributes
casAuthnLevel	CAS authentication level
casSrvMetaDataOptions	Root of CAS server options
casStorage	Apache::Session module to store CAS user data
casStorageOptions	Apache::Session module parameters
cda	Enable Cross Domain Authentication
certificateResetByMailCeaAttribute	
certificateResetByMailCertificateAttribute	
certificateResetByMailStep1Body	Custom Certificate reset mail body
certificateResetByMailStep1Subject	Mail subject for certificate reset email
certificateResetByMailStep2Body	Custom confirm Certificate reset mail body
certificateResetByMailStep2Subject	Mail subject for reset confirmation
certificateResetByMailURL	URL of certificate reset page
certificateResetByMailValidityDelay	
cfgAuthor	Name of the author of the current configuration
cfgAuthorIP	Uploader IP address of the current configuration
cfgDate	Timestamp of the current configuration
cfgLog	Configuration update log
cfgNum	Enable Cross Domain Authentication
cfgVersion	Version of LLNG which build configuration
checkState	Enable CheckState plugin
checkStateSecret	Secret token for CheckState plugin
checkTime	Timeout to check new configuration in local cache
checkUser	Enable check user
checkUserDisplayComputedSession	Display empty headers rule
checkUserDisplayEmptyHeaders	Display empty headers rule
checkUserDisplayEmptyValues	Display session empty values rule
checkUserDisplayNormalizedHeaders	Display normalized headers rule
checkUserDisplayPersistentInfo	Display persistent session info rule
checkUserHiddenAttributes	Attributes to hide in CheckUser plugin
checkUserHiddenHeaders	Header values to hide if not empty
checkUserIdRule	checkUser identities rule
checkUserSearchAttributes	Attributes used for retrieving sessions in user DataBase
checkUserUnrestrictedUsersRule	checkUser unrestricted users rule
checkXSS	Check XSS
combModules	Combination module description
combination	Combination rule
compactConf	Compact configuration
configStorage	Configuration storage
confirmFormMethod	HTTP method for confirm page form
contextSwitchingAllowed2fModifications	Allowed SFA modifications
contextSwitchingIdRule	Context switching identities rule
contextSwitchingPrefix	Prefix to store real session Id
contextSwitchingRule	Context switching activation rule
contextSwitchingStopWithLogout	Stop context switching by logout
contextSwitchingUnrestrictedUsersRule	Context switching unrestricted users rule
cookieExpiration	Cookie expiration

Table 1 – continued from previous page

Key name	Documentation
cookieName	Name of the main cookie
corsAllow_Credentials	Allow credentials for Cross-Origin Resource Sharing
corsAllow_Headers	Allowed headers for Cross-Origin Resource Sharing
corsAllow_Methods	Allowed methods for Cross-Origin Resource Sharing
corsAllow_Origin	Allowed origine for Cross-Origin Resource Sharing
corsEnabled	Enable Cross-Origin Resource Sharing
corsExpose_Headers	Exposed headers for Cross-Origin Resource Sharing
corsMax_Age	MAx-age for Cross-Origin Resource Sharing
cspConnect	Authorized Ajax destination for Content-Security-Policy
cspDefault	Default value for Content-Security-Policy
cspFont	Font source for Content-Security-Policy
cspFormAction	Form action destination for Content-Security-Policy
cspFrameAncestors	Frame-Ancestors for Content-Security-Policy
cspImg	Image source for Content-Security-Policy
cspScript	Javascript source for Content-Security-Policy
cspStyle	Style source for Content-Security-Policy
customAddParams	Custom additional parameters
customAuth	Custom auth module
customFunctions	List of custom functions
customPassword	Custom password module
customPlugins	Custom plugins
customPluginsParams	Custom plugins parameters
customRegister	Custom register module
customResetCertByMail	Custom certificateResetByMail module
customToTrace	Session parameter used to fill REMOTE_CUSTOM
customUserDB	Custom user DB module
dbiAuthChain	
dbiAuthLoginCol	
dbiAuthPassword	
dbiAuthPasswordCol	
dbiAuthPasswordHash	
dbiAuthTable	
dbiAuthUser	
dbiAuthnLevel	DBI authentication level
dbiDynamicHashEnabled	
dbiDynamicHashNewPasswordScheme	
dbiDynamicHashValidSaltedSchemes	
dbiDynamicHashValidSchemes	
dbiExportedVars	DBI exported variables
dbiPasswordMailCol	
dbiUserChain	
dbiUserPassword	
dbiUserTable	
dbiUserUser	
decryptValueFunctions	Custom function used for decrypting values
decryptValueRule	Decrypt value activation rule
demoExportedVars	Demo exported variables
disablePersistentStorage	Enabled persistent storage
displaySessionId	Display _session_id with sessions explorer

Table 1 – continued from previous page

Key name	Documentation
domain	DNS domain
exportedAttr	List of attributes to export by SOAP or REST servers
exportedVars	Main exported variables
ext2FSendCommand	Send command of External second factor
ext2FValidateCommand	Validation command of External second factor
ext2fActivation	External second factor activation
ext2fAuthnLevel	Authentication level for users authentified by External second facto
ext2fCodeActivation	OTP generated by Portal
ext2fLabel	Portal label for External second factor
ext2fLogo	Custom logo for External 2F
facebookAppId	
facebookAppSecret	
facebookAuthnLevel	Facebook authentication level
facebookExportedVars	Facebook exported variables
facebookUserField	
failedLoginNumber	Number of failures stored in login history
findUser	Enable find user
findUserControl	Regular expression to validate parameters
findUserExcludingAttributes	Attributes used for excluding accounts
findUserSearchingAttributes	Attributes used for searching accounts
findUserWildcard	Character used as wildcard
forceGlobalStorageIssuerOTT	Force Issuer tokens to be stored into Global Storage
forceGlobalStorageUpgradeOTT	Force Upgrade tokens be stored into Global Storage
formTimeout	Token timeout for forms
githubAuthnLevel	GitHub authentication level
githubClientID	
githubClientSecret	
githubScope	
githubUserField	
globalLogoutCustomParam	Custom session parameter to display
globalLogoutRule	Global logout activation rule
globalLogoutTimer	Global logout auto accept time
globalStorage	Session backend module
globalStorageOptions	Session backend module options
gpgAuthnLevel	GPG authentication level
gpgDb	GPG keys database
grantSessionRules	Rules to grant sessions
groups	Groups
groupsBeforeMacros	Compute groups before macros
handlerInternalCache	Handler internal cache timeout
handlerServiceTokenTTL	Handler ServiceToken timeout
hiddenAttributes	Name of attributes to hide in logs
hideOldPassword	Hide old password in portal
httpOnly	Enable httpOnly flag in cookie
https	Use HTTPS for redirection from portal
impersonationHiddenAttributes	Attributes to skip
impersonationIdRule	Impersonation identities rule
impersonationMergeSSOgroups	Merge spoofed and real SSO groups
impersonationPrefix	Prefix to rename real session attributes

Table 1 – continued from previous page

Key name	Documentation
impersonationRule	Impersonation activation rule
impersonationSkipEmptyValues	Skip session empty values
impersonationUnrestrictedUsersRule	Impersonation unrestricted users rule
infoFormMethod	HTTP method for info page form
issuerDBCASActivation	CAS server activation
issuerDBCASPath	CAS server request path
issuerDBCASRule	CAS server rule
issuerDBGetActivation	Get issuer activation
issuerDBGetParameters	List of virtualHosts with their get parameters
issuerDBGetPath	Get issuer request path
issuerDBGetRule	Get issuer rule
issuerDBOpenIDActivation	OpenID server activation
issuerDBOpenIDConnectActivation	OpenID Connect server activation
issuerDBOpenIDConnectPath	OpenID Connect server request path
issuerDBOpenIDConnectRule	OpenID Connect server rule
issuerDBOpenIDPath	OpenID server request path
issuerDBOpenIDRule	OpenID server rule
issuerDBSAMLActivation	SAML IDP activation
issuerDBSAMLPath	SAML IDP request path
issuerDBSAMLRule	SAML IDP rule
issuersTimeout	Token timeout for issuers
jsRedirect	Use javascript for redirections
key	Secret key
krbAllowedDomains	Allowed domains
krbAuthnLevel	Null authentication level
krbByJs	Launch Kerberos authentication by Ajax
krbKeytab	Kerberos keytab
krbRemoveDomain	Remove domain in Kerberos username
ldapAllowResetExpiredPassword	Allow a user to reset his expired password
ldapAuthnLevel	LDAP authentication level
ldapBase	LDAP search base
ldapCAFile	Location of the certificate file for LDAP connections
ldapCAPath	Location of the CA directory for LDAP connections
ldapChangePasswordAsUser	
ldapExportedVars	LDAP exported variables
ldapGetUserBeforePasswordChange	
ldapGroupAttributeName	LDAP attribute name for member in groups
ldapGroupAttributeNameGroup	LDAP attribute name in group entry referenced as member in grou
ldapGroupAttributeNameSearch	LDAP attributes to search in groups
ldapGroupAttributeNameUser	LDAP attribute name in user entry referenced as member in groups
ldapGroupBase	
ldapGroupDecodeSearchedValue	Decode value before searching it in LDAP groups
ldapGroupObjectClass	LDAP object class of groups
ldapGroupRecursive	LDAP recursive search in groups
ldapIOTimeout	LDAP operation timeout
ldapITDS	Support for IBM Tivoli Directory Server
ldapPasswordResetAttribute	LDAP password reset attribute
ldapPasswordResetAttributeValue	LDAP password reset value
ldapPort	LDAP port

Table 1 – continued from previous page

Key name	Documentation
ldapPpolicyControl	
ldapPwdEnc	LDAP password encoding
ldapRaw	
ldapSearchDeref	“deref” param of Net::LDAP::search()
ldapServer	LDAP server (host or URI)
ldapSetPassword	
ldapTimeout	LDAP connection timeout
ldapUsePasswordResetAttribute	LDAP store reset flag in an attribute
ldapVerify	Whether to validate LDAP certificates
ldapVersion	LDAP protocol version
linkedInAuthnLevel	LinkedIn authentication level
linkedInClientID	
linkedInClientSecret	
linkedInFields	
linkedInScope	
linkedInUserField	
localSessionStorage	Local sessions cache module
localSessionStorageOptions	Sessions cache module options
localStorage	Local cache
localStorageOptions	Local cache parameters
log4perlConfFile	Log4Perl logger configuration file
logLevel	Log level, must be set in .ini
logger	technical logger
loginHistoryEnabled	Enable login history
logoutServices	Send logout through GET request to these services
lwpOpts	Options given to LWP::UserAgent
lwpSslOpts	SSL options given to LWP::UserAgent
macros	Macros
mail2fActivation	Mail second factor activation
mail2fAuthnLevel	Authentication level for users authenticated by Mail second factor
mail2fBody	Mail body for second factor authentication
mail2fCodeRegex	Regular expression to create a mail OTP code
mail2fLabel	Portal label for Mail second factor
mail2fLogo	Custom logo for Mail 2F
mail2fSessionKey	Session parameter where mail is stored
mail2fSubject	Mail subject for second factor authentication
mail2fTimeout	Second factor code timeout
mailBody	Custom password reset mail body
mailCharset	Mail charset
mailConfirmBody	Custom confirm password reset mail body
mailConfirmSubject	Mail subject for reset confirmation
mailFrom	Sender email
mailLDAPFilter	LDAP filter for mail search
mailOnPasswordChange	Send a mail when password is changed
mailReplyTo	Reply-To address
mailSessionKey	Session parameter where mail is stored
mailSubject	Mail subject for new password email
mailTimeout	Mail password reset session timeout
mailUrl	URL of password reset page

Table 1 – continued from previous page

Key name	Documentation
maintenance	Maintenance mode for all virtual hosts
managerDn	LDAP manager DN
managerPassword	LDAP manager Password
max2FDevices	Maximum registered 2F devices
max2FDevicesNameLength	Maximum 2F devices name length
multiValuesSeparator	Separator for multiple values
mySessionAuthorizedRWKeys	Alterable session keys by user itself
nginxCustomHandlers	Custom Nginx handler (deprecated)
noAjaxHook	Avoid replacing 302 by 401 for Ajax responses
notification	Notification activation
notificationDefaultCond	Notification default condition
notificationServer	Notification server activation
notificationServerDELETE	Notification server activation
notificationServerGET	Notification server activation
notificationServerPOST	Notification server activation
notificationServerSentAttributes	Prameters to send with notification server GET method
notificationStorage	Notification backend
notificationStorageOptions	Notification backend options
notificationWildcard	Notification string to match all users
notificationXSLTfile	Custom XSLT document for notifications
notificationsExplorer	Notifications explorer activation
notificationsMaxRetrieve	Max number of displayed notifications
notifyDeleted	Show deleted sessions in portal
notifyOther	Show other sessions in portal
nullAuthnLevel	Null authentication level
oidcAuthnLevel	OpenID Connect authentication level
oidcOPMetaDataOptions	
oidcRPCallbackGetParam	OpenID Connect Callback GET URLparameter
oidcRPMetaDataOptions	
oidcRPStateTimeout	OpenID Connect Timeout of state sessions
oidcServiceAccessTokenExpiration	OpenID Connect global access token TTL
oidcServiceAllowAuthorizationCodeFlow	OpenID Connect allow authorization code flow
oidcServiceAllowDynamicRegistration	OpenID Connect allow dynamic client registration
oidcServiceAllowHybridFlow	OpenID Connect allow hybrid flow
oidcServiceAllowImplicitFlow	OpenID Connect allow implicit flow
oidcServiceAuthorizationCodeExpiration	OpenID Connect global code TTL
oidcServiceDynamicRegistrationExportedVars	OpenID Connect exported variables for dynamic registration
oidcServiceDynamicRegistrationExtraClaims	OpenID Connect extra claims for dynamic registration
oidcServiceIDTokenExpiration	OpenID Connect global ID token TTL
oidcServiceKeyIdSig	OpenID Connect Signature Key ID
oidcServiceMetaDataAuthnContext	OpenID Connect Authentication Context Class Ref
oidcServiceMetaDataAuthorizeURI	OpenID Connect authorizaton endpoint
oidcServiceMetaDataBackChannelURI	OpenID Connect Front-Channel logout endpoint
oidcServiceMetaDataCheckSessionURI	OpenID Connect check session iframe
oidcServiceMetaDataEndSessionURI	OpenID Connect end session endpoint
oidcServiceMetaDataFrontChannelURI	OpenID Connect Front-Channel logout endpoint
oidcServiceMetaDataIntrospectionURI	OpenID Connect introspection endpoint
oidcServiceMetaDataIssuer	OpenID Connect issuer
oidcServiceMetaDataJWKSURI	OpenID Connect JWKS endpoint

Table 1 – continued from previous page

Key name	Documentation
oidcServiceMetaDataRegistrationURI	OpenID Connect registration endpoint
oidcServiceMetaDataTokenURI	OpenID Connect token endpoint
oidcServiceMetaDataUserInfoURI	OpenID Connect user info endpoint
oidcServiceOfflineSessionExpiration	OpenID Connect global offline session TTL
oidcServicePrivateKeySig	
oidcServicePublicKeySig	
oidcStorage	Apache::Session module to store OIDC user data
oidcStorageOptions	Apache::Session module parameters
oldNotifFormat	Use old XML format for notifications
openIdAttr	
openIdAuthnLevel	OpenID authentication level
openIdExportedVars	OpenID exported variables
openIdIDPList	
openIdIssuerSecret	
openIdSPList	
openIdSecret	
openIdSreg_country	
openIdSreg_dob	
openIdSreg_email	OpenID SREG email session parameter
openIdSreg_fullname	OpenID SREG fullname session parameter
openIdSreg_gender	
openIdSreg_language	
openIdSreg_nickname	OpenID SREG nickname session parameter
openIdSreg_postcode	
openIdSreg_timezone	OpenID SREG timezone session parameter
pamAuthnLevel	PAM authentication level
pamService	PAM service
passwordDB	Password module
passwordPolicyActivation	Enable password policy
passwordPolicyMinDigit	Password policy: minimal digit characters
passwordPolicyMinLower	Password policy: minimal lower characters
passwordPolicyMinSize	Password policy: minimal size
passwordPolicyMinSpeChar	Password policy: minimal special characters
passwordPolicyMinUpper	Password policy: minimal upper characters
passwordPolicySpecialChar	Password policy: allowed special characters
passwordResetAllowedRetries	Maximum number of retries to reset password
pdataDomain	pdata cookie DNS domain
persistentSessionAttributes	Persistent session attributes to hide
persistentStorage	Storage module for persistent sessions
persistentStorageOptions	Options for persistent sessions storage module
port	Force port in redirection
portal	Portal URL
portalAntiFrame	Avoid portal to be displayed inside frames
portalCheckLogins	Display login history checkbox in portal
portalCustomCss	Path to custom CSS file
portalDisplayAppslist	Display applications tab in portal
portalDisplayCertificateResetByMail	Display certificate reset by mail button in portal
portalDisplayChangePassword	Display password tab in portal
portalDisplayGeneratePassword	Display password generate box in reset password form

Table 1 – continued from previous page

Key name	Documentation
portalDisplayLoginHistory	Display login history tab in portal
portalDisplayLogout	Display logout tab in portal
portalDisplayOidcConsents	Display OIDC consent tab in portal
portalDisplayPasswordPolicy	Display policy in password form
portalDisplayRefreshMyRights	Display link to refresh the user session
portalDisplayRegister	Display register button in portal
portalDisplayResetPassword	Display reset password button in portal
portalErrorOnExpiredSession	Show error if session is expired
portalErrorOnMailNotFound	Show error if mail is not found in password reset process
portalForceAuthn	Enable force to authenticate when displaying portal
portalForceAuthnInterval	Maximum interval in seconds since last authentication to force reau
portalMainLogo	Portal main logo path
portalOpenLinkInNewWindow	Open applications in new windows
portalPingInterval	Interval in ms between portal Ajax pings
portalRequireOldPassword	Rule to require old password to change the password
portalSkin	Name of portal skin
portalSkinBackground	Background image of portal skin
portalSkinRules	Rules to choose portal skin
portalStatus	Enable portal status
portalUserAttr	Session parameter to display connected user in portal
protection	Manager protection method
proxyAuthService	
proxyAuthnLevel	Proxy authentication level
proxySessionService	
proxyUseSoap	Use SOAP instead of REST
radius2fActivation	Radius second factor activation
radius2fAuthnLevel	Authentication level for users authenticated by Radius second facto
radius2fLabel	Portal label for Radius 2F
radius2fLogo	Custom logo for Radius 2F
radius2fSecret	
radius2fServer	
radius2fTimeout	Radius 2f verification timeout
radius2fUsernameSessionKey	Session key used as Radius login
radiusAuthnLevel	Radius authentication level
radiusSecret	
radiusServer	
randomPasswordRegexp	Regular expression to create a random password
redirectFormMethod	HTTP method for redirect page form
refreshSessions	Refresh sessions plugin
registerConfirmSubject	Mail subject for register confirmation
registerDB	Register module
registerDoneSubject	Mail subject when register is done
registerTimeout	Register session timeout
registerUrl	URL of register page
reloadTimeout	Configuration reload timeout
reloadUrls	URL to call on reload
remoteCookieName	
remoteGlobalStorage	Remote session backend
remoteGlobalStorageOptions	Apache::Session module parameters

Table 1 – continued from previous page

Key name	Documentation
remotePortal	
requireToken	Enable token for forms
rest2fActivation	REST second factor activation
rest2fAuthnLevel	Authentication level for users authentified by REST second factor
rest2fInitArgs	Args for REST 2F init
rest2fInitUrl	REST 2F init URL
rest2fLabel	Portal label for REST second factor
rest2fLogo	Custom logo for REST 2F
rest2fVerifyArgs	Args for REST 2F init
rest2fVerifyUrl	REST 2F init URL
restAuthServer	Enable REST authentication server
restAuthUrl	
restAuthnLevel	REST authentication level
restClockTolerance	How tolerant the REST session server will be to clock dift
restConfigServer	Enable REST config server
restExportSecretKeys	Allow to export secret keys in REST session server
restFindUserDBUrl	
restPasswordServer	Enable REST password reset server
restPwdConfirmUrl	
restPwdModifyUrl	
restSessionServer	Enable REST session server
restUserDBUrl	
sameSite	Cookie SameSite value
samlAttributeAuthorityDescriptorAttributeServiceSOAP	SAML Attribute Authority SOAP
samlAuthnContextMapKerberos	SAML authn context kerberos level
samlAuthnContextMapPassword	SAML authn context password level
samlAuthnContextMapPasswordProtectedTransport	SAML authn context password protected transport level
samlAuthnContextMapTLSClient	SAML authn context TLS client level
samlCommonDomainCookieActivation	SAML CDC activation
samlCommonDomainCookieDomain	
samlCommonDomainCookieReader	
samlCommonDomainCookieWriter	
samlDiscoveryProtocolActivation	SAML Discovery Protocol activation
samlDiscoveryProtocolIsPassive	SAML Discovery Protocol Is Passive
samlDiscoveryProtocolPolicy	SAML Discovery Protocol Policy
samlDiscoveryProtocolURL	SAML Discovery Protocol EndPoint URL
samlEntityID	SAML service entityID
samlIDPMetaDataOptions	
samlIDPSSODescriptorArtifactResolutionServiceArtifact	SAML IDP artifact resolution service
samlIDPSSODescriptorSingleLogoutServiceHTTPPost	SAML IDP SLO HTTP POST
samlIDPSSODescriptorSingleLogoutServiceHTTPRedirect	SAML IDP SLO HTTP Redirect
samlIDPSSODescriptorSingleLogoutServiceSOAP	SAML IDP SLO SOAP
samlIDPSSODescriptorSingleSignOnServiceHTTPArtifact	SAML IDP SSO HTTP Artifact
samlIDPSSODescriptorSingleSignOnServiceHTTPPost	SAML IDP SSO HTTP POST
samlIDPSSODescriptorSingleSignOnServiceHTTPRedirect	SAML IDP SSO HTTP Redirect
samlIDPSSODescriptorWantAuthnRequestsSigned	SAML IDP want authn request signed
samlMetadataForceUTF8	SAML force metadata UTF8 conversion
samlNameIDFormatMapEmail	SAML session parameter for NameID email
samlNameIDFormatMapKerberos	SAML session parameter for NameID kerberos

Table 1 – continued from previous page

Key name	Documentation
samlNameIDFormatMapWindows	SAML session parameter for NameID windows
samlNameIDFormatMapX509	SAML session parameter for NameID x509
samlOrganizationDisplayName	SAML service organization display name
samlOrganizationName	SAML service organization name
samlOrganizationURL	SAML service organization URL
samlOverrideIDPEntityID	Override SAML EntityID when acting as an IDP
samlRelayStateTimeout	SAML timeout of relay state
samlSPMetaDataOptions	
samlSPSSODescriptorArtifactResolutionServiceArtifact	SAML SP artifact resolution service
samlSPSSODescriptorAssertionConsumerServiceHTTPArtifact	SAML SP ACS HTTP artifact
samlSPSSODescriptorAssertionConsumerServiceHTTPPost	SAML SP ACS HTTP POST
samlSPSSODescriptorAuthnRequestsSigned	SAML SP AuthnRequestsSigned
samlSPSSODescriptorSingleLogoutServiceHTTPPost	SAML SP SLO HTTP POST
samlSPSSODescriptorSingleLogoutServiceHTTPRedirect	SAML SP SLO HTTP Redirect
samlSPSSODescriptorSingleLogoutServiceSOAP	SAML SP SLO SOAP
samlSPSSODescriptorWantAssertionsSigned	SAML SP WantAssertionsSigned
samlServicePrivateKeyEnc	SAML encryption private key
samlServicePrivateKeyEncPwd	
samlServicePrivateKeySig	SAML signature private key
samlServicePrivateKeySigPwd	SAML signature private key password
samlServicePublicKeyEnc	SAML encryption public key
samlServicePublicKeySig	SAML signature public key
samlServiceSignatureMethod	
samlServiceUseCertificateInResponse	Use certificate instead of public key in SAML responses
samlStorage	Apache::Session module to store SAML user data
samlStorageOptions	Apache::Session module parameters
samlUseQueryStringSpecific	SAML use specific method for query_string
secureTokenAllowOnError	Secure Token allow requests in error
secureTokenAttribute	Secure Token attribute
secureTokenExpiration	Secure Token expiration
secureTokenHeader	Secure Token header
secureTokenMemcachedServers	Secure Token Memcached servers
secureTokenUrls	
securedCookie	Cookie securisation method
sentryDsn	Sentry logger DSN
sessionDataToRemember	Data to remember in login history
sfEngine	Second factor engine
sfExtra	Extra second factors
sfManagerRule	Rule to display second factor Manager link
sfOnlyUpgrade	Only trigger second factor on session upgrade
sfRemovedMsgRule	Display a message if at leat one expired SF has been removed
sfRemovedNotifMsg	Notification message
sfRemovedNotifRef	Notification reference
sfRemovedNotifTitle	Notification title
sfRemovedUseNotif	Use Notifications plugin to display message
sfRequired	Second factor required
showLanguages	Display langs icons
singleIP	Allow only one session per IP
singleSession	Allow only one session per user

Table 1 – continued from previous page

Key name	Documentation
singleUserByIP	Allow only one user per IP
skipRenewConfirmation	Avoid asking confirmation when an Issuer asks to renew auth
skipUpgradeConfirmation	Avoid asking confirmation during a session upgrade
slaveAuthnLevel	Slave authentication level
slaveDisplayLogo	Display Slave authentication logo
slaveExportedVars	Slave exported variables
slaveHeaderContent	
slaveHeaderName	
slaveMasterIP	
slaveUserHeader	
soapConfigServer	Enable SOAP config server
soapProxyUrn	SOAP URN for Proxy
soapSessionServer	Enable SOAP session server
sslByAjax	Use Ajax request for SSL
sslHost	URL for SSL Ajax request
staticPrefix	Prefix of static files for HTML templates
status	Status daemon activation
stayConnected	Enable StayConnected plugin
stayConnectedCookieName	Name of the stayConnected plugin cookie
stayConnectedTimeout	StayConnected persistent connexion session timeout
storePassword	Store password in session
successLoginNumber	Number of success stored in login history
syslogFacility	Syslog logger technical facility
timeout	Session timeout on server side
timeoutActivity	Session activity timeout on server side
timeoutActivityInterval	Update session timeout interval on server side
tokenUseGlobalStorage	Enable global token storage
totp2fActivation	TOTP activation
totp2fAuthnLevel	Authentication level for users authentified by password+TOTP
totp2fDigits	Number of digits for TOTP code
totp2fDisplayExistingSecret	Display existing TOTP secret in registration form
totp2fInterval	TOTP interval
totp2fIssuer	TOTP Issuer
totp2fLabel	Portal label for TOTP 2F
totp2fLogo	Custom logo for TOTP 2F
totp2fRange	TOTP range (number of interval to test)
totp2fSelfRegistration	TOTP self registration activation
totp2fTTL	TOTP device time to live
totp2fUserCanChangeKey	Authorize users to change existing TOTP secret
totp2fUserCanRemoveKey	Authorize users to remove existing TOTP secret
trustedDomains	Trusted domains
twitterAppName	
twitterAuthnLevel	Twitter authentication level
twitterKey	
twitterSecret	
twitterUserField	
u2fActivation	U2F activation
u2fAuthnLevel	Authentication level for users authentified by password+U2F
u2fLabel	Portal label for U2F

Table 1 – continued from previous page

Key name	Documentation
u2fLogo	Custom logo for U2F
u2fSelfRegistration	U2F self registration activation
u2fTTL	U2F device time to live
u2fUserCanRemoveKey	Authorize users to remove existing U2F key
upgradeSession	Upgrade session activation
useRedirectOnError	Use 302 redirect code for error (500)
useRedirectOnForbidden	Use 302 redirect code for forbidden (403)
useSafeJail	Activate Safe jail
userControl	Regular expression to validate login
userDB	User module
userLogger	User actions logger
userPivot	
userSyslogFacility	Syslog logger user-actions facility
utotp2fActivation	UTOTP activation (mixed U2F/TOTP module)
utotp2fAuthnLevel	Authentication level for users authentified by password+(U2F or T
utotp2fLabel	Portal label for U2F+TOTP
utotp2fLogo	Custom logo for U2F+TOTP
vhostOptions	
viewerAllowBrowser	Allow configuration browser
viewerAllowDiff	Allow configuration diff
viewerHiddenKeys	Hidden Conf keys
webIDAuthnLevel	WebID authentication level
webIDExportedVars	WebID exported variables
webIDWhitelist	
whatToTrace	Session parameter used to fill REMOTE_USER
wsdlServer	Enable /portal.wsdl server
yubikey2fActivation	Yubikey second factor activation
yubikey2fAuthnLevel	Authentication level for users authentified by Yubikey second facto
yubikey2fClientID	Yubico client ID
yubikey2fFromSessionAttribute	Provision yubikey from the given session variable
yubikey2fLabel	Portal label for Yubikey second factor
yubikey2fLogo	Custom logo for Yubikey 2F
yubikey2fNonce	Yubico nonce
yubikey2fPublicIDSize	Yubikey public ID size
yubikey2fSecretKey	Yubico secret key
yubikey2fSelfRegistration	Yubikey self registration activation
yubikey2fTTL	Yubikey device time to live
yubikey2fUrl	Yubico server
yubikey2fUserCanRemoveKey	Authorize users to remove existing Yubikey
zimbraAccountKey	Zimbra account session key
zimbraBy	Zimbra account type
zimbraPreAuthKey	Zimbra preauthentication key
zimbraSsoUrl	Zimbra local SSO URL pattern
zimbraUrl	Zimbra preauthentication URL

[1]: complex nodes

16.13.2 Configuration backend parameters

Full name	Key name	Configuration backend
Configuration load timeout	confTimeout	all backends (default: 10)
DBI connection string	dbiChain	*CDBI / RDBI*
DBI user	dbiUser	
DBI password	dbiPassword	
DBI table name	dbiTable	
Directory	dirName	*File* / *YAML*
LDAP server	ldapServer	*LDAP*
LDAP port	ldapPort	
LDAP base	ldapConfBase	
LDAP bind dn	ldapBindDN	
LDAP bind password	ldapBindPassword	
LDAP ObjectClass	ldapObjectClass	
LDAP ID attribute	ldapAttributeId	
LDAP content attribute	ldapAttributeContent	
Certificate authorities file	caFile	
Certificate authorities directory	caPath	
MongoDB database	dbName	*MongoDB*
MongoDB collection	collectionName	
Pretty print	prettyPrint	*File*
REST base URL	baseUrl	*REST*
REST realm	realm	
REST user	user	
REST password	password	
SOAP server location (URL)	proxy	*SOAP*
LWP::UserAgent parameters	proxyOptions	
SOAP user	User	
SOAP password	Password	

CHAPTER

SEVENTEEN

MINI HOWTOS

17.1 Command Line Interface (lemonldap-ng-cli) examples

This page shows some examples of LL::NG Command Line Interface. See *how to use the command*.

Attention: On Debian, the command is located in `/usr/share/lemonldap-ng/bin` and on CentOS in `/usr/libexec/lemonldap-ng/bin`. Adapt the path for the system you are using.

17.1.1 Save/restore configuration

This part requires LLNG 2.0.5 at least.

Save:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli save >config.json
```

Restore:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli restore config.json
# Or
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli restore - <config.json
```

Rollback (restore previous configuration, *since 2.0.8*):

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli rollback
```

17.1.2 Configure HTTPS

When setting HTTPS, you first need to modify Apache/Nginx configuration, then you must configure LL::NG to change portal URL, Handler redirections, cookie settings, ...

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        portal https://auth.example.com \
        mailUrl https://auth.example.com/resetpwd \
        registerUrl https://auth.example.com/register \
        https 1 \
        securedCookie 1
```

17.1.3 Configure sessions backend

For production, it is recommended to use *Browseable session backend*. Once tables are created with columns corresponding to index, the following commands can be executed to set all the session backends.

In this example we have:

- Backend: PostGreSQL
- DB user: lemonldaplogin
- DB password: lemonldappw
- Database: lemonldapdb
- Host: pg.example.com
- SSO sessions:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    delKey \
        globalStorageOptions Directory \
        globalStorageOptions LockDirectory

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        globalStorage Apache::Session::Browseable::Postgres

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        globalStorageOptions DataSource 'DBI:Pg:database=lemonldapdb;host=pg.example.com
↪' \
        globalStorageOptions UserName 'lemonldaplogin' \
        globalStorageOptions Password 'lemonldappw' \
        globalStorageOptions Commit 1 \
        globalStorageOptions Index 'ipAddr _whatToTrace user' \
        globalStorageOptions TableName 'sessions'
```

- Persistent sessions:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    delKey \
        persistentStorageOptions Directory \
        persistentStorageOptions LockDirectory

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        persistentStorage Apache::Session::Browseable::Postgres

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        persistentStorageOptions DataSource 'DBI:Pg:database=lemonldapdb;host=pg.example.
↪com' \
        persistentStorageOptions UserName 'lemonldaplogin' \
        persistentStorageOptions Password 'lemonldappw' \
        persistentStorageOptions Commit 1 \
```

(continues on next page)

(continued from previous page)

```
        persistentStorageOptions Index '_session_uid' \
        persistentStorageOptions TableName 'psessions'
```

- CAS sessions

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        casStorage Apache::Session::Browseable::Postgres

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        casStorageOptions DataSource 'DBI:Pg:database=lemonldapdb;host=pg.example.com' \
        casStorageOptions UserName 'lemonldaplogin' \
        casStorageOptions Password 'lemonldappw' \
        casStorageOptions Commit 1 \
        casStorageOptions Index '_cas_id' \
        casStorageOptions TableName 'cassessions'
```

- SAML sessions

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        samlStorage Apache::Session::Browseable::Postgres

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
       samlStorageOptions DataSource 'DBI:Pg:database=lemonldapdb;host=pg.example.com' \
       samlStorageOptions UserName 'lemonldaplogin' \
       samlStorageOptions Password 'lemonldappw' \
       samlStorageOptions Commit 1 \
       samlStorageOptions Index '_saml_id ProxyID _nameID _assert_id _art_id _session_id
↪' \
       samlStorageOptions TableName 'samlsessions'
```

- OpenID Connect sessions

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
       oidcStorage Apache::Session::Browseable::Postgres

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
       oidcStorageOptions DataSource 'DBI:Pg:database=lemonldapdb;host=pg.example.com' \
       oidcStorageOptions UserName 'lemonldaplogin' \
       oidcStorageOptions Password 'lemonldappw' \
       oidcStorageOptions Commit 1 \
       oidcStorageOptions TableName 'oidcsessions'
```

17.1.4 Configure virtual host

A virtual host must be defined in Apache/Nginx and access rules and exported headers must be configured in LL::NG.

In this example we have:

- host: test.example.com
- Access rules:
  - default => accept
  - Logout: ^/logout.php => logout_sso
- Headers:
  - Auth-User: $uid
  - Auth-Mail: $mail

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        'locationRules/test.example.com' 'default' 'accept' \
        'locationRules/test.example.com' '(?#Logout)^/logout\.php' 'logout_sso' \
        'exportedHeaders/test.example.com' 'Auth-User' '$uid' \
        'exportedHeaders/test.example.com' 'Auth-Mail' '$mail'
```

17.1.5 Configure form replay

To add form replay on a host, you need to set the catched URI and the variables to post.

In this example we have:

- Host: test.example.com
- Catched URI: /login.php
- jQuery URL: default
- **Variables:**
  - login: $uid
  - password: $_password

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 -sep , \
    addKey \
        post,test.example.com,'/login.php' jqueryUrl default

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 -sep , \
    addPostVars \
        post,test.example.com,'/login.php' login '$uid' \
        post,test.example.com,'/login.php' password '$_password'
```

17.1.6 Configure LDAP authentication backend

In this example we use:

- LDAP server: ldap://ldap.example.com
- LDAP Bind DN : cn=lemonldapng,ou=dsa,dc=example,dc=com
- LDAP Bind PW: changeit
- LDAP search base: ou=users,dc=example,dc=com
- LDAP attributes:
  - uid => uid
  - cn => cn
  - mail => mail
  - sn => sn
  - givenName => givenName
  - mobile => mobile
- LDAP group base: ou=groups,dc=example,dc=com
- Use recursive search for groups

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        authentication LDAP \
        userDB LDAP \
        passwordDB LDAP \
        ldapServer 'ldap://ldap.example.com' \
        managerDn 'cn=lemonldapng,ou=dsa,dc=example,dc=com' \
        managerPassword 'changeit' \
        ldapBase 'ou=users,dc=example,dc=com'

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        ldapExportedVars uid uid \
        ldapExportedVars cn cn \
        ldapExportedVars sn sn \
        ldapExportedVars mobile mobile \
        ldapExportedVars mail mail \
        ldapExportedVars givenName givenName

/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        ldapGroupBase 'ou=groups,dc=example,dc=com' \
        ldapGroupObjectClass groupOfNames \
        ldapGroupAttributeName member \
        ldapGroupAttributeNameGroup dn \
        ldapGroupAttributeNameSearch cn \
        ldapGroupAttributeNameUser dn \
        ldapGroupRecursive 1
```

17.1.7 Configure CAS Identity Provider

You just have to enable the CAS server feature, and you can set the access control policy (see *CAS service options*):

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        issuerDBCASActivation 1 \
        casAccessControlPolicy error
```

17.1.8 Register a CAS application

This is only required if your access control policy is not `none`.

In this example we have:

- App configuration key: testapp
- App service URL: https://testapp.example.com/
- App exported attribute: mail and cn

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        casAppMetaDataExportedVars/testapp mail mail \
        casAppMetaDataExportedVars/testapp cn cn
        casAppMetaDataOptions/testapp casAppMetaDataOptionsService 'https://testapp.
↪example.com/'
```

17.1.9 Configure SAML Identity Provider

You can then generate a private key and a self-signed certificate with these commands;

```
openssl req -new -newkey rsa:4096 -keyout saml.key -nodes  -out saml.pem -x509 -days 3650
```

Fix the certificate key format (you can skip this step if you are running >= 2.0.6)

```
sed -e "s/END PRIVATE/END RSA PRIVATE/" \
    -e "s/BEGIN PRIVATE/BEGIN RSA PRIVATE/" \
    -i saml.key
```

Import them in configuration and activate the SAML issuer

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        samlServicePrivateKeySig "`cat saml.key`" \
        samlServicePublicKeySig "`cat saml.pem`" \
        issuerDBSAMLActivation 1
```

You can also define organization name and URL for SAML metadata:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        samlOrganizationName 'ACME' \
```

(continues on next page)

(continued from previous page)

```
        samlOrganizationDisplayName 'ACME Corporation' \
        samlOrganizationURL 'http://www.acme.com'
```

17.1.10 Register an SAML Service Provider

In this example we have:

- SP configuration key: testsp
- SP metadata file: metadata-testsp.xml
- SP exported attribute: EmailAdress (filled with mail session key)

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        samlSPMetaDataXML/testsp samlSPMetaDataXML "`cat metadata-testsp.xml`" \
        samlSPMetaDataExportedAttributes/testsp mail '1;EmailAddress'
```

17.1.11 Configure OpenID Connect Identity Provider

Activate the OpenID Connect Issuer and set issuer name (equal to portal URL):

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        issuerDBOpenIDConnectActivation 1
```

Generate keys:

```
openssl genrsa -out oidc.key 4096
openssl rsa -pubout -in oidc.key -out oidc_pub.key
```

Import them:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        oidcServicePrivateKeySig "`cat oidc.key`" \
        oidcServicePublicKeySig "`cat oidc_pub.key`" \
        oidcServiceKeyIdSig "randomstring"
```

If needed you can allow implicit and hybrid flows:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        oidcServiceAllowImplicitFlow 1 \
        oidcServiceAllowHybridFlow 1
```

17.1.12 Register an OpenID Connect Relying Party

In this example we have:

- RP configuration key: testrp
- Client ID : testclientid
- Client secret : testclientsecret
- Allowed redirection URL:
  - For login: https://testrp.example.com/?callback=1
  - For logout: https://testrp.example.com/
- Exported attributes:
  - email => mail
  - familiy_name => sn
  - name => cn
- Exported attributes:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        oidcRPMetaDataExportedVars/testrp email mail \
        oidcRPMetaDataExportedVars/testrp family_name sn \
        oidcRPMetaDataExportedVars/testrp name cn
```

- Credentials:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        oidcRPMetaDataOptions/testrp oidcRPMetaDataOptionsClientID testclientid \
        oidcRPMetaDataOptions/testrp oidcRPMetaDataOptionsClientSecret testclientsecret
```

- Redirection:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        oidcRPMetaDataOptions/testrp oidcRPMetaDataOptionsRedirectUris 'https://testrp.
↪example.com/?callback=1' \
        oidcRPMetaDataOptions/testrp oidcRPMetaDataOptionsPostLogoutRedirectUris 'https:/
↪/testrp.example.com/'
```

- Signature and token expiration:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        oidcRPMetaDataOptions/testrp  oidcRPMetaDataOptionsIDTokenSignAlg RS512 \
        oidcRPMetaDataOptions/testrp  oidcRPMetaDataOptionsIDTokenExpiration 3600 \
        oidcRPMetaDataOptions/testrp oidcRPMetaDataOptionsAccessTokenExpiration 3600
```

17.1.13 Categories and applications in menu

Create the category "applications":

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        applicationList/applications type category \
        applicationList/applications catname Applications
```

Create the application "sample" inside category "applications":

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    addKey \
        applicationList/applications/sample type application \
        applicationList/applications/sample/options description "A sample application" \
        applicationList/applications/sample/options display "auto" \
        applicationList/applications/sample/options logo "tux.png" \
        applicationList/applications/sample/options name "Sample application" \
        applicationList/applications/sample/options uri "https://sample.example.com/"
```

17.1.14 Encryption key

To update the master encryption key:

```
/usr/share/lemonldap-ng/bin/lemonldap-ng-cli -yes 1 \
    set \
        key 'xxxxxxxxxxxxxx'
```

17.1.15 Sessions Management

New in version 2.0.9.

Get the content of a session

```
lemonldap-ng-sessions get 9684dd2a6489bf2be2fbdd799a8028e3
```

Get the content of a persistent session

```
lemonldap-ng-sessions get --persistent dwho
```

Search all sessions by username

```
lemonldap-ng-sessions search --where uid=dwho
```

Modify session

```
lemonldap-ng-sessions setKey 9684dd2a6489bf2be2fbdd799a8028e3 \
   authenticationLevel 1
```

New in version 2.0.10: Delete all sessions by username

```
lemonldap-ng-sessions delete --where uid=dwho
```

17.1.16 Second Factors management

New in version 2.0.9.

List second factors of a user

```
lemonldap-ng-sessions secondfactors get dwho
```

Deregister Yubikey of a user

```
lemonldap-ng-sessions secondfactors delType dwho UBK
```

17.1.17 OIDC Consents management

New in version 2.0.9.

List consents of a user

```
lemonldap-ng-sessions consents get dwho
```

Revoke consents on OIDC provider 'test' for a user:

```
lemonldap-ng-sessions consents delete dwho test
```

17.2 Manager protection

When installing LL::NG, the Manager can only be accessed with the demo account `dwho`. This How To explains how change this default behavior to protect Manager with other rules.

17.2.1 Apache based protection

Tip: Apache based protection allow one to be independent from WebSSO, so Manager will always be reachable even if WebSSO configuration is corrupted.

The configuration can be changed in `etc/manager-apache2.conf`, for example to restrict the IP allowed to access the Manager:

```
<Directory /usr/local/lemonldap-ng/htdocs/manager/>
    Order deny,allow
    Deny from all
    Allow from 127.0.0.0/8 192.168.100.0/32
    Options +ExecCGI
</Directory>
```

But you will rather prefer to use an Apache authentication module, like for example LDAP authentication module:

```
<Directory /usr/local/lemonldap-ng/htdocs/manager/>
    AuthzLDAPAuthoritative On
    AuthName "LL::NG Manager"
```

(continues on next page)

(continued from previous page)

```
    AuthType Basic
    AuthBasicProvider ldap
    AuthLDAPBindDN "ou=websso,ou=applications,dc=example,dc=com"
    AuthLDAPBindPassword "secret"
    AuthLDAPURL ldap://localhost:389/ou=users,dc=example,dc=com???
↪(objectClass=inetOrgPerson) TLS
    Require ldap-user coudot xguimard tchemineau
    Options +ExecCGI
</Directory>
```

Attention: You need to disable default Manager protection in lemonldap-ng.ini to rely only on Apache:

```
[manager]
;protection = manager
```

17.2.2 LL::NG based protection

Danger: Before enabling Manager protection by LL::NG, you must have configured how users authenticate on Portal, and test that you can log in without difficulties. Else, you will lock access to Manager and will never access it anymore.

By default, you will have a manager virtual host define in configuration. If not Go on Manager, and declare Manager as a new *virtual host*, for example `manager.example.com`. You can then set the access rule. No headers are needed.

The default rule is:

```
$uid eq "dwho"
```

You have to change it to match your admin user (or use other conditions like group membership, or any other rule based on a session variable).

Save the configuration and exit the Manager.

Tip: The next time you will access Manager, it will be through LL::NG.

Enable protection on Manager, by editing `lemonldap-ng.ini`:

```
[manager]
protection = manager
```

You can also adapt Apache access control:

```
<Directory /usr/local/lemonldap-ng/htdocs/manager/>
    Order deny,allow
    Allow from all
    Options +ExecCGI
</Directory>
```

Restart Apache and try to log on Manager. You should be redirected to LL::NG Portal.

You can then add the Manager as *an application in the menu*.

Tip: If for an obscure reason, the WebSSO is not working and you want to access the Manager, remove the protection in `lemonldap-ng.ini`. Add an Apache access control to avoid other access.

17.3 Configure LemonLDAP::NG to use MySQL as main database

LL::NG use 2 internal databases to store its configuration and sessions.

17.3.1 Use MySQL for Lemonldap::NG configuration

Steps:

- *Prepare the database and the LL::NG configuration file*
- *Convert existing configuration*
- Restart all your Apache servers

17.3.2 Use MySQL for Lemonldap::NG sessions

Steps:

- Choose one of the following:
  - *Using Apache::Session::Browseable::MySQL* (recommended for best performances)
  - *Using Apache::Session::MySQL* (if you choose this option, then read *how to increase MySQL performances*)

17.4 Configure LemonLDAP::NG to use LDAP as main database

LL::NG use 2 internal databases to store its configuration and sessions.

17.4.1 Use LDAP for configuration

Steps:

- *Prepare the LDAP server and the LL::NG configuration file*
- *Convert existing configuration*
- Restart all your Apache servers

17.4.2 Use LDAP for sessions

Steps:

- Follow *LDAP session backend* doc

17.5 Configure LemonLDAP::NG to use REST proxy mechanism

LL::NG use 2 internal databases to store its configuration and sessions. It can be configured to use REST instead of direct access to those databases (for remote servers).

Tip: This mechanism can be used to secure access for remote servers that cross an unsecured network to access to LL::NG databases.

17.5.1 Use REST for Lemonldap::NG configuration

Steps:

- *Choose and configure your main configuration storage system*
- Follow *REST configuration backend* page
- Restart all your remote Apache servers

17.5.2 Use REST for Lemonldap::NG sessions

Steps:

- *Choose and configure your main sessions storage system*
- Follow *REST sessions backend* page

17.6 Configure LemonLDAP::NG to use SOAP proxy mechanism

LL::NG use 2 internal databases to store its configuration and sessions. It can be configured to use SOAP instead of direct access to those databases (for remote servers). .. tip:

```
This mechanism can be used to
secure access for remote servers that cross an unsecured network to
access to LL::NG databases.
```

Since version 2.0, same services are available by REST.

17.6.1 Use SOAP for Lemonldap::NG configuration

Steps:

- *Choose and configure your main configuration storage system*
- Follow *SOAP configuration backend* page
- Restart all your remote Apache servers

17.6.2 Use SOAP for Lemonldap::NG sessions

Steps:

- *Choose and configure your main sessions storage system*
- Follow *SOAP sessions backend* page

17.7 Using LemonLDAP::NG with Active-Directory

17.7.1 Authentication with login/password

To use Active Directory as LDAP backend, you must change few things in the manager :

- Use "Active Directory" as authentication, userDB and passwordDBbackends,
- Export sAMAccountName in a variable declared in *exported variables*
- Change the user attribute to store in Apache logs *("General Parameters » Logs » REMOTE_USER")*: use the variable declared above

17.7.2 Authentication with Kerberos

- Choose "Apache" as authentication module *("General Parameters » Authentication modules » Authentication module")*
- *Configure the Apache server* that host the portal to use the Apache Kerberos authentication module

17.8 Kerberos

17.8.1 Presentation

This documentation will explain how to use Active Directory as Kerberos server, and provide transparent authentication for one or multiple AD domains.

You can use Kerberos in LL::NG with the following authentication modules:

- *Kerberos* (recommended): use Perl GSSAPI module, compatible with Apache and Nginx
- *Apache*: use mod_auth_kerb or mod_auth_gssapi in Apache

17.8.2 Prerequisites

Example values

We will use the following values in our examples

- **EXAMPLE.COM**: First AD domain
- **ACME.COM**: Second AD domain
- **auth.example.com**: DNS of the LL::NG portal
- **KERB_AUTH**: AD account to generate the keytab for LL::NG server

Server time

It is mandatory that LL::NG servers and AD servers have the same time. It is recommended to use NTP to do this.

DNS

In our experience, we have observed the following limitations when using Kerberos for web applications in an Active Directory environment

- `auth.example.com` must be registered in the DNS server as a `A` record. `CNAME` usually do not work
- The reverse DNS (`PTR`) for `auth.example.com`'s IP address MUST point back to `auth.example.com`

Tip: If you have a SSO cluster, you must setup a Virtual IP in cluster and register this IP in DNS.

Tip: If you cannot configure the PTR record to point to the portal's hostname, it may help to run the following command. Assuming that `proxy.example.com` is the PTR record of the portal's IP address

```
setspn -s HTTP/proxy.example.com keytab-account
```

SSL

SSL is not mandatory, but it is strongly recommended. Your portal URL should be https://auth.example.com.

Web browser configuration

Firefox

Type `about:config` in a tab and search for `trusted`. Then edit the property `network.negotiate-auth.trusted-uris` and set value `example.com`.

Internet Explorer

Add `https://auth.example.com` as trusted site.

Check into security parameters that Kerberos authentication is allowed.

17.8.3 Single AD domain

Client Kerberos configuration

On LL::NG server, edit `/etc/krb5.conf`:

```
[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_kdc = false
 dns_lookup_realm = no
 ticket_lifetime = 24h
 forwardable = yes
 renewable = true

[realms]
 EXAMPLE.COM = {
  kdc = ad.example.com
  admin_server = ad.example.com
 }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

You can check that Kerberos is working by trying to get a ticket for a user of the domain (for example coudot):

```
kinit coudot@EXAMPLE.COM
```

You should be prompted to enter password. Then list the tickets:

```
klist -e
```

You should see a krbtgt ticket:

```
Valid starting     Expires            Service principal
06/04/15 15:43:24  06/05/15 01:43:29  krbtgt/EXAMPLE.COM@EXAMPLE.COM
        renew until 06/05/15 15:43:24, Etype (skey, tkt): aes256-cts-hmac-sha1-96,␣
↪aes256-cts-hmac-sha1-96
```

You can then close the Kerberos session:

```
kdestroy
```

Obtain keytab file

You have to run this command on Active Directory:

```
ktpass -princ HTTP/auth.example.com@EXAMPLE.COM -mapuser KERB_AUTH@EXAMPLE.COM -crypto␣
↪All -ptype KRB5_NT_PRINCIPAL -mapOp set -pass <PASSWORD> -out c:\auth.keytab
```

Attention: The values passed in -crypto and -ptype depend on the Active Directory version and the windows version of the workstations. You can for example use RC4-HMAC-NT as crypto protocol if DES is not supported by workstations (this the case by default for Window 8 for example).

The file `auth.keytab` should then be copied (with a secure media) to the Linux server (for example in `/etc/lemonldap-ng`).

Change rights on keytab file:

```
chown apache /etc/lemonldap-ng/auth.keytab
chmod 600 /etc/lemonldap-ng/auth.keytab
```

You can check the validity of the keytab file by trying to request a service ticket, and compare the result with the keytab content.

Open a Kerberos session (like done in the previous step):

```
kinit coudot@example.com
```

Request a service ticket:

```
kvno HTTP/auth.example.com@EXAMPLE.COM
```

The result of the command should be:

```
HTTP/auth.example.com@EXAMPLE.COM: kvno = 3
```

Read the service ticket:

```
klist -e
```

You should see this kind of ticket:

```
06/04/15 16:28:49  06/05/15 02:28:11  HTTP/auth.example.com@EXAMPLE.COM
        renew until 06/05/15 16:28:07, Etype (skey, tkt): arcfour-hmac, arcfour-hmac
```

You can close the Kerberos session:

```
kdestroy
```

Now you can compare the above result with the same request done through the keytab file:

```
klist -e -k -t /etc/lemonldap-ng/auth.keytab
```

The result of the command should be:

```
Keytab name: FILE:/etc/lemonldap-ng/auth.keytab
KVNO Timestamp         Principal
---- ----------------- --------------------------------------------------------
   3 01/01/70 01:00:00 HTTP/auth.example.com@EXAMPLE.COM (arcfour-hmac)
```

The important things to check are:

- KVNO must be the same
- Principal names must be the same
- Encryption types must be the same

17.8.4 Multiple AD domains

Client Kerberos configuration

The two domains must be defined in `/etc/krb5.conf`:

```
[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_kdc = false
 dns_lookup_realm = no
 ticket_lifetime = 24h
 forwardable = yes
 renewable = true

[realms]
 EXAMPLE.COM = {
  kdc = ad.example.com
  admin_server = ad.example.com
  default_domain = EXAMPLE.COM
 }
 ACME.COM = {
  kdc = ad.acme.com
  admin_server = ad.acme.com
  }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .acme.com = ACME.COM
 acme.com = ACME.COM
```

You should then be able to open a Kerberos session on each domain:

```
kinit coudot@EXAMPLE.COM
klist -e
kdestroy
```

```
kinit coudot@ACME.COM
klist -e
kdestroy
```

Obtain keytab file

You need to obtain a keytab for each node on each domain. This means the ktpass commands should be run on both AD.

Then you will have 2 keytab files for each node, for example:

- node1-example.keytab
- node1-acme.keytab

You need to concatenate the keytab files, thanks to `ktutil` command:

```
ktutil
ktutil: read_kt node1-example.keytab
ktutil: read_kt node1-acme.keytab
ktutil: write_kt /etc/lemonldap-ng/auth.keytab
ktutil: quit
```

You can then remove the original keytab files and protect the final keytab file:

```
chown apache /etc/lemonldap-ng/auth.keytab
chmod 600 /etc/lemonldap-ng/auth.keytab
```

17.8.5 Other resources

You can check these documentations to get more information:

- http://modauthkerb.sourceforge.net/configure.html
- http://www.grolmsnet.de/kerbtut/

17.9 LL::NG as federation protocol proxy

LL::NG can use federation protocols (SAML, CAS, OpenID) independently to:

- authenticate users
- provide identities to other systems

So you can configure it to authenticate users using a federation protocol and simultaneously to provide identities using other(s) federation protocols.

Schemes tested:

- SAML / OpenID-Connect:
  - SAML-SP **<=>** LLNG as *SAML*/*OpenID-Connect* proxy **<=>** OIDC Provider
  - OIDC-RP **<=>** LLNG as *OpenID-Connect*/*SAML* proxy **<=>** SAML Identity Provider
- SAML / CAS
  - SAML-SP **<=>** LLNG as *SAML*/*CAS* proxy **<=>** CAS Server
  - CAS Application **<=>** LLNG as *CAS*/*SAML* proxy **<=>** SAML Identity Provider

Note that OpenID-Connect consortium hasn't already defined single-logout initiated by OpenID-Connect Provider. LLNG will implement it when this standard will be published.

Attention: Federation proxy installation can be complex. Don't hesitate to contact us on lemonldap-ng-users@ow2.org

See the following chapters:

- *Authentication protocols*
- *Identity provider*

17.10 Convert HTTP header into environment variable

17.10.1 Apache

Using LL::NG in reverse proxy mode, you will not have the REMOTE_USER environment variable set. Indeed, this variable is set by the Handler on the physical server hosting the Handler, and not on other servers where the Handler is not installed.

Apache SetEnvIf module will let you transform the Auth-User HTTP header in REMOTE_USER environment variable:

```
SetEnvIfNoCase Auth-User "(.*)" REMOTE_USER=$1
```

This can be used to protect applications relying on REMOTE_USER environment variable in reverse proxy mode. In this case you will have two Apache configuration files:

- Apache configuration file on LL::NG reverse proxy (hosting LL::NG Handler):

```
<VirtualHost *:80>
        ServerName application.example.com

        PerlHeaderParserHandler Lemonldap::NG::Handler::ApacheMP2

        ProxyPreserveHost on
        ProxyPass / http://APPLICATION_IP/
        ProxyPassReverse / http://APPLICATION_IP/

</VirtualHost>
```

- Apache configuration file on application server (hosting the application):

```
<VirtualHost *:80>
        ServerName application.example.com

        SetEnvIfNoCase Auth-User "(.*)" REMOTE_USER=$1

        DocumentRoot /var/www/application

</VirtualHost>
```

Tip: Sometimes, PHP applications also check the PHP_AUTH_USER and PHP_AUHT_PW environment variables. You can set them the same way:

```
SetEnvIfNoCase Auth-User "(.*)" PHP_AUTH_USER=$1
SetEnvIfNoCase Auth-Password "(.*)" PHP_AUTH_PW=$1
```

Of course, you need to *store password in session* to fill PHP_AUTH_PW.

17.10.2 Nginx

Nginx doesn't launch directly PHP pages (or other languages): it dials with FastCGI servers (like php-fpm). As you can see in examples, it's easy to map a LLNG header to a fastcgi param. Example:

```
auth_request_set $authuser $upstream_http_auth_user;
fastcgi_param HTTP_MYVAR $authuser;
```

17.11 Connect to Renater Federation

17.11.1 Presentation

Renater provides an SAML federation for higher education in France.

It is based on SAMLv2 but add some specific items like a WAYF service and a metadata bundle to list all SP and IDP from the federation.

Since LL::NG 2.0, you can register into Renater federation.

17.11.2 Register as Service Provider

LL::NG configuration

Configure LL::NG as SAML Service Provider with this *documentation*. You don't need to declare any IDP for the moment.

Configure *SAML Discovery Protocol* to redirect users on WAYF Service. The endpoint URL is https://discovery.renater.fr/renater/WAYF.

Metadata import

You now need to import IDP metadata in LL::NG configuration. Use the `importMetadata` script that should be installed in /usr/share/lemonldap-ng/bin. You need to select the correct metadata bundle proposed by Renater: https://services.renater.fr/federation/technique/metadata.

For Renater, you need to customize some settings of the script, copy it and edit configuration:

```
cp /usr/share/lemonldap-ng/bin/importMetadata /usr/share/lemonldap-ng/bin/
↪importMetadataRenater
vi /usr/share/lemonldap-ng/bin/importMetadataRenater
```

Set attributes (use the SAML Name, not FriendlyName) that are provided by IDPs, for example:

```
my $exportedAttributes = {
    'cn'                          => '0;urn:oid:2.5.4.3',
    'eduPersonPrincipalName'      => '1;urn:oid:1.3.6.1.4.1.5923.1.1.1.6',
    'givenName'                   => '0;urn:oid:2.5.4.42',
    'sn'                          => '0;urn:oid:2.5.4.4',
    'eduPersonAffiliation'        => '0;urn:oid:1.3.6.1.4.1.5923.1.1.1.1',
    'eduPersonPrimaryAffiliation' => '0;urn:oid:1.3.6.1.4.1.5923.1.1.1.5',
    'mail'                        => '0;urn:oid:0.9.2342.19200300.100.1.3',
    'supannListeRouge'            => '0;urn:oid:1.3.6.1.4.1.7135.1.2.1.1',
    'supannEtuCursusAnnee'        => '0;rn:oid:1.3.6.1.4.1.5923.1.1.1.10',
};
```

Adapt IDP options, for example:

```
my $idpOptions = {
    'samlIDPMetaDataOptionsAdaptSessionUtime'        => 0,
    'samlIDPMetaDataOptionsAllowLoginFromIDP'        => 0,
    'samlIDPMetaDataOptionsAllowProxiedAuthn'        => 0,
    'samlIDPMetaDataOptionsCheckAudience'            => 1,
    'samlIDPMetaDataOptionsCheckSLOMessageSignature' => 1,
    'samlIDPMetaDataOptionsCheckSSOMessageSignature' => 1,
    'samlIDPMetaDataOptionsCheckTime'                => 1,
    'samlIDPMetaDataOptionsEncryptionMode'           => 'none',
    'samlIDPMetaDataOptionsForceAuthn'               => 0,
    'samlIDPMetaDataOptionsForceUTF8'                => 1,
    'samlIDPMetaDataOptionsIsPassive'                => 0,
    'samlIDPMetaDataOptionsNameIDFormat'             => 'transient',
    'samlIDPMetaDataOptionsRelayStateURL'            => 0,
    'samlIDPMetaDataOptionsSignSLOMessage'           => -1,
    'samlIDPMetaDataOptionsSignSSOMessage'           => -1,
```

(continues on next page)

(continued from previous page)

```
    'samlIDPMetaDataOptionsStoreSAMLToken'           => 0,
    'samlIDPMetaDataOptionsUserAttribute' => 'urn:oid:1.3.6.1.4.1.5923.1.1.1.6',
};
```

Then run the script:

```
/usr/share/lemonldap-ng/bin/importMetadataRenater -m https://metadata.federation.renater.
↪fr/renater/main/main-idps-renater-metadata.xml -r -i "idp-renater-" -s "sp-renater-"
```

Attention: You need to add this in cron to refresh metadata into LL::NG configuration.

Add your SP into the federation

Go to https://federation.renater.fr/registry and register your SP.

Attention: Be sure to check all attributes as mandatory to be able to get them in SAML assertions.

17.11.3 Register as Identity Provider

LL::NG configuration

Configure LL::NG as SAML Identity Provider with this *documentation*. You don't need to declare any SP for the moment.

Attention: If your LL::NG server will act as SP and IDP inside Renater federation, you need to set the advanced parameter "Override Entity ID for IDP". Indeed, Renater do not allow to register a SP and an IDP with the same entityID.

Metadata import

You now need to import SP metadata in LL::NG configuration. Use the `importMetadata` script that should be installed in /usr/share/lemonldap-ng/bin. You need to select the correct metadata bundle proposed by Renater: https://services.renater.fr/federation/technique/metadata.

For Renater, you may need to customize some settings of the script, copy it and edit configuration:

```
cp /usr/share/lemonldap-ng/bin/importMetadata /usr/share/lemonldap-ng/bin/
↪importMetadataRenater
vi /usr/share/lemonldap-ng/bin/importMetadataRenater
```

Adapt IDP options, for example:

```
my $spOptions = {
    'samlSPMetaDataOptionsCheckSLOMessageSignature'    => 1,
    'samlSPMetaDataOptionsCheckSSOMessageSignature'    => 1,
    'samlSPMetaDataOptionsEnableIDPInitiatedURL'       => 0,
```

(continues on next page)

(continued from previous page)

```
    'samlSPMetaDataOptionsEncryptionMode'             => 'none',
    'samlSPMetaDataOptionsForceUTF8'                  => 1,
    'samlSPMetaDataOptionsNameIDFormat'               => '',
    'samlSPMetaDataOptionsNotOnOrAfterTimeout'        => 72000,
    'samlSPMetaDataOptionsOneTimeUse'                 => 0,
    'samlSPMetaDataOptionsSessionNotOnOrAfterTimeout' => 72000,
    'samlSPMetaDataOptionsSignSLOMessage'             => 1,
    'samlSPMetaDataOptionsSignSSOMessage'             => 1
};
```

Then run the script:

```
/usr/share/lemonldap-ng/bin/importMetadataRenater -m https://metadata.federation.renater.
↪fr/renater/main/main-sps-renater-metadata.xml -r -i "idp-renater" -s "sp-renater"
```

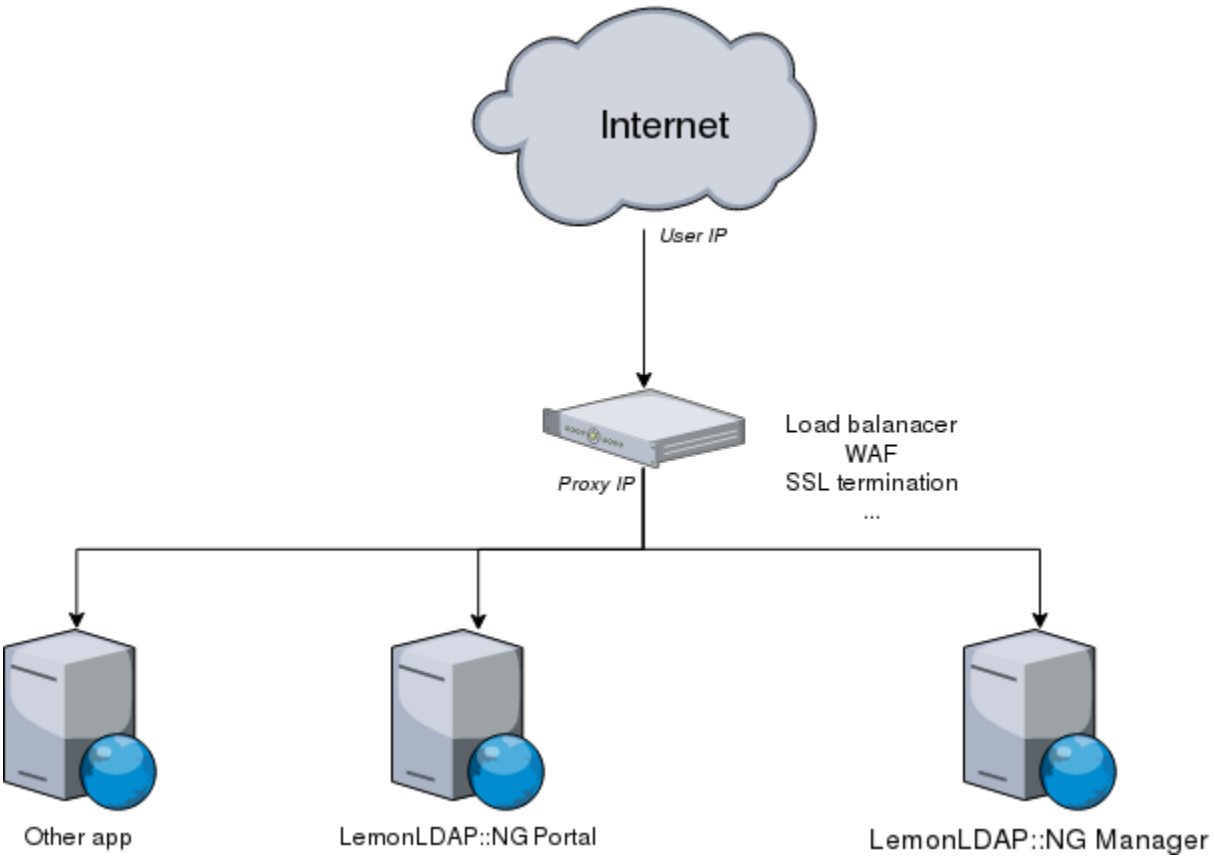
Attention: You need to add this in cron to refresh metadata into LL::NG configuration.

Add your IDP into the federation

Go to https://federation.renater.fr/registry and register your IDP.

17.12 Running LemonLDAP::NG behind a reverse proxy

Your network infrastructure might require that LemonLDAP::NG components (Portal, Manager, Handler) run behind a reverse proxy.



17.12.1 Transmitting the correct IP address to the portal

In this case, LemonLDAP::NG components will store the ip address of the connection between the reverse proxy and the webserver in the session, and in logs. This prevents features such as session restrictions and rules based on `ipAddr` from working as expected.

A Content Delivery Network (CDN) would also have the same issue.

In order to make LemonLDAP::NG behave correctly behind a proxy, you need to forward the original IP address all the way to LemonLDAP::NG.

In order to do this you have several options.

HTTP Header

This generic method is the most likely to work in your particular environment.

First, configure your reverse proxy (or CDN) to send the origin IP address in a HTTP header. Most reverse proxies do this by default, generally in a header named `X-Forwarded-For` or `X-Real-IP`.

Once the header is transmitted to LemonLDAP::NG's web server, you may uncomment the relevant parts of the configuration file.

- For Nginx:

```
set_real_ip_from  127.0.0.1;
real_ip_header    X-Forwarded-For;
```

Tip: Make sure Nginx was compiled with the http_real_ip module

- For Apache:

```
RemoteIPHeader X-Forwarded-For
RemoteIPInternalProxy 127.0.0.1
```

Tip: Make sure the mod_remoteip module is enabled in your Apache installation

Danger: Both modules need you to specify the address of your reverse proxy. Using the `http_real_ip` or `mod_remoteip` module might let an attacker impersonate any IP address they want by setting the `X-Forwarded-For` header themselves. Please read the relevant module documentation carefully.

PROXY Protocol

Alternatively, if your proxy supports the PROXY protocol (Nginx, HAProxy, Amazon ELB), you may use it to carry over the information almost transparently.

Refer to your reverse proxy's documentation to find out how to enable the PROXY protocol on the reverse proxy side.

Then, on the LemonLDAP::NG side, in the NGINX configuration of your Portal/Manager/Handler:

```
 listen 80   proxy_protocol;
# or
# listen 443 ssl proxy_protocol;

set_real_ip_from  127.0.0.1;
real_ip_header    proxy_protocol;
```

17.12.2 Fixing handler redirections

If your handler server runs behind a reverse proxy, it may have trouble figuring out the right URL to redirect you to after logging in.

In this case, you can force a particular port and scheme in the Virtual Host's Options.

But is instead you want this scheme to be auto-detected by LemonLDAP (in order to have a same VHost domain available over multiple schemes), you can also use the following declarations in the handler's virtual host to force LemonLDAP to use the correct port and scheme

Nginx

```
fastcgi_param SERVER_PORT 443
fastcgi_param HTTPS On
```

Apache

```
PerlSetEnv SERVER_PORT 443
PerlSetEnv HTTPS On
```

17.13 Use an outgoing proxy

For some protocols, LL::NG has to directly contact the external server. This is the case for example with CAS authentication (validation of service ticket) or OpenID Connect authentication (access to token endpoint and userinfo endpoint).

If the LL::NG server needs a proxy for outgoing connections, then you need to configure some environment variables.

17.13.1 Apache

In Apache configuration, set:

```
FcgidInitialEnv http_proxy http://X.X.X.X:X
FcgidInitialEnv https_proxy http://X.X.X.X:X
# on Centos7, you need LWP::Protocol::connect
# FcgidInitialEnv https_proxy connect://X.X.X.X:X
```

17.13.2 Nginx/FastCGI

add in `/etc/default/lemonldap-ng-fastcgi-server` :

```
http_proxy=http://X.X.X.X:X
https_proxy=http://X.X.X.X:X
# on Centos7, you need LWP::Protocol::connect
# https_proxy=connect://X.X.X.X:X
```

17.14 Test OpenID Connect with command line tools

We present here how to test the OpenID Connect protocol (authorization code flow) with commande line tools, like *curl*.

We use in this example a public OIDC provider based on LL::NG: https://oidctest.wsweet.org

17.14.1 Authentication

The first step is to obtain a valid SSO session on the portal. Several solutions:

- Use a web browser and log into the portal, then get the value of the SSO cookie
- Use portal REST API, and adapt the *requireToken* configuration to get cookie value in JSON response (see *REST services*)

Example of REST service usage, with credentials *dwho/dwho*:

```
curl -X POST -d user=dwho -d password=dwho -H 'Accept: application/json' 'https://
↪oidctest.wsweet.org/oauth2/'
```

The session id is displayed in JSON response:

```
{
 "error" : "0",
 "id" : "0640f95827111f00ba7ad5863ba819fe46cfbcecdb18ce525836369fb4c8350b",
 "result" : 1
}
```

17.14.2 Authorization code

In the first step of authorization code flow, we request a temporary code, ont the *authorize* end point.

Parameters needed:

- SSO session id (will be passed in *lemonldap* cookie, adapt the name if needed)
- Client ID: given by your OIDC provider, we use here *private*
- Scope: depends on which information you need, we will use here *openid profile email*
- Redirect URI: shoud match the value registered in your OIDC provider, we will use here *http://localhost*

The OIDC provide will return the code in the location header, so we just output this reponse header:

```
curl -s -D - -o /dev/null -b␣
↪lemonldap=0640f95827111f00ba7ad5863ba819fe46cfbcecdb18ce525836369fb4c8350b 'https://
↪oidctest.wsweet.org/oauth2/authorize?response_type=code&client_id=private&
↪scope=openid+profile+email&redirect_uri=http://localhost' | grep '^location'
```

The value of the location header is:

```
location: http://localhost?
↪code=294b0facd91a0fa92762edc48d18369e99c330ba2b8fb05ab2c45999fcef6e17&session_
↪state=BpB8KRMBEDUs%2B7lAjsz4DRk3E0RJImxgUbMsCFFAUa8%3D.
↪N3dVOFg3a2RpNXVJK3ltSldrYXZjUjhtU0tvd29sWkpuWWJJbll5ZGs5NzhZMnh5bmQwd0IxRmJVWUxJSTlkWDBnSWZ2SWFVZmU0U
```

So we get the code value: *94b0facd91a0fa92762edc48d18369e99c330ba2b8fb05ab2c45999fcef6e17*

This code has a short lifetime, we will use it to get access token and ID token in the next step

17.14.3 Tokens

In this step, we exchange the authorization code against tokens:

- Access token
- ID token
- Refresh token (optional)

Parameters needed:

- Authorization code: see previous step
- Grant type: we use here *authorization_code*
- Redirect URI: same value as the one used in the previous step
- Client ID and Client Secret: given by your OIDC provider, we use here *private/tardis*

```
curl -X POST -d grant_type=authorization_code -d 'redirect_uri=http://localhost' -d␣
↪code=94b0facd91a0fa92762edc48d18369e99c330ba2b8fb05ab2c45999fcef6e17 -u 'private:tardis
↪' 'https://oidctest.wsweet.org/oauth2/token' | json_pp
```

The JSON response looks like this:

```
{
 "access_token" : "a88b8dde538719e55c3cb8fbd14d06ed77853c685a62abf6ecb88d86228a9c64",
 "expires_in" : 3600,
 "id_token" : "eyJhbGciOiJSUzI1NiIsImtpZCI6Im9pZGN0ZXN0IiwidHlwIjoiSldUIn0.
↪eyJhdXRoX3RpbWUiOjE2MTQxNjAwMDYsImlhdCI6MTYxNDE2MzIxOCwiaXNzIjoiaHR0cHM6Ly9vaWRjdGVzdC53c3dlZXQub3JnL
↪N3TNufjKLzKM3qiIitA7JHUei4L572XjF6AcVl7UAFB6efdGUCiAL7amlUl0FgjZfzW9bzvulBVDidoYSicIaysIdI4KkjmjpVN0Z
↪tmA3txeR18nzfhdckq-S-6Lx7wrWpPNyrzGx-FImbOaUPN2yeVhKPXhdyHJbzI0RqJETxnBkyW-
↪CLEzAJyq3rCUVX-D8kHADvg6a42QQyPdxvBuGrdBfyDDDb_Py13H1qhn40NnuFknR1wSahsY6U97uUooyk-0_
↪U4J3XJAHySjCtivtSeP0fM_
↪5eblMuh6WdVjrfnUF0xnCTbCa2gYRlTS38BkqcsWY26PXoRAOo31a1cmB5sMSZyPtRF9UZcmGiNBIymMMdFgVAJONb6uliiTS5j9-
↪nkmHOqVC-XJ6tuiU3ZSBQ8nCRyNW2LaCzpJ5c3ytP9yYQtyT8HmhN0VnXob3K1uJEA_Xcu4sADjtrm-
↪LbrGiwaVMkfu-C6YIrbuC9riOW6TneV2gAzAjXPOW_
↪UZeXrCrx66GHIJPsJIq29UfbTN5Pxo9SH2yKw6PSfxevkZhBIhEXCOMaIUHrlWz2jDBBzPIWeiSRbK_
↪MRtejQmdRUs8nqdq-McVwnFiUMDt1KZXxqScTtMDF_Lo9oK2RaCijEJ7MSPEscr_YOyp3KIq2FLVg",
 "refresh_token" : "19434440ed4da2803e8ba9d91cb2eabd5b8bd12af2609429bda03ed487e6ef57",
 "token_type" : "Bearer"
}
```

The access token will be used for the last step, to get information about the user.

The ID Token is a JWT (JSON Web Token) and can be parsed easily, as this is the concatenation of 3 JSON strings encoded in base 64: *base64(header).base64(payload).base64(signature)*.

Decoding the payload gives:

```
{
 "acr" : "loa-2",
 "at_hash" : "HTk09ScJ4NlAnky9HaE_eQ",
 "aud" : [
   "private"
 ],
 "auth_time" : 1614160006,
 "azp" : "private",
```

(continues on next page)

(continued from previous page)

```
 "exp" : 1614166818,
 "iat" : 1614163218,
 "iss" : "https://oidctest.wsweet.org/",
 "sub" : "dwho"
}
```

17.14.4 User info

This step is optional and allows to fetch user information linked to scopes requested in the first step.

Parameters needed:

- Access token, used as bearer authorization

```
curl -H 'Authorization: Bearer␣
↪a88b8dde538719e55c3cb8fbd14d06ed77853c685a62abf6ecb88d86228a9c64' 'https://oidctest.
↪wsweet.org/oauth2/userinfo' | json_pp
```

JSON response:

```
{
 "email" : "dwho@badwolf.org",
 "name" : "Doctor Who",
 "preferred_username" : "dwho",
 "sub" : "dwho"
}
```

CHAPTER

EIGHTEEN

EXPLOITATION

18.1 Performances

LemonLDAP::NG is designed for high performance, both in throughput and response time. Indeed, it can use Apache2 threads capabilities **but** since Apache version 2.4, mpm_worker seems to break mod_perl. So to increase performances, prefer using Nginx.

18.1.1 Built-in

Cache system

LLNG uses different cache systems to avoid querying to many the databases:

	Lifetime in memory		Lifetime in Local-Cache (file)		DB
	Parameter	Default	Parameter	Default	
Configuration	`checkTime`	1 second		Until "reload" order	✓
Session	`handlerInternalCache`	15 seconds	`default_expires_in`[1]	10 minutes	✓

Note: Configuration and sessions are first looked up in-memory, then in the cache file, and then in their backing store. This means that after a configuration reload (using Manager), you have to wait for `checkTime` before you can see your changes, or wait for configuration cache expiration in `checkTime` is disabled.

[1] Manager >> General parameters >> Sessions >> Sessions storage >> Cache module options

18.1.2 Global performance

By default, Linux does not use DNS cache and LemonLDAP::NG portal request DNS for each connexions on LDAP or DB. Under heavy loads, that can generated hundred of DNS queries and many errors on LDAP connexions (timed out) from IO::Socket.

To bypass this, you can:

- Use IP in configuration to avoid DNS resolution
- Install a DNS cache like nscd, dnsmasq or unbound

Cron optimization (or systemd timers)

LLNG installs its cron files without knowing how many servers are installed. You should optimize this to launch:

- purgeCentralCache: only 1 time every 10 minutes for the whole system (or more)
- purgeLocalCache: ~ 1 time per hour on each server

18.1.3 Handler performance

For Nginx, you can use another auth server instead of llng-fastcgi-server. See: *Advanced PSGI usage*.

To increase handler performance, you can disable "Sessions activity timeout" to prevent it from writing to the session database.

Handlers check rights and calculate headers for each HTTP hit. So to improve performances, avoid too complex rules by using macros, groups or local macros.

Local macros

Macros and groups are stored in session database. Local macros is a special feature of handler that permit one to have macros useable localy only. Those macros are calculated only at the first usage and stored in the local session cache (only for this server) and only if the user access to the related applications. This avoid to have to many datas stored.

```
# rule
admin -> $admin ||= ($uid eq 'foo' or $uid eq 'bar')
# header
Display-Name -> $displayName ||= $givenName." ".$surName
```

Tip: Note that this feature is interesting only for the Lemonldap::NG systems protecting a high number of applications

18.1.4 Portal performances

General performances

The portal is the biggest component of Lemonldap::NG. Since version 2.0, portal runs under FastCGI and has been rewritten using plugins, so performance is increased in comparison to earlier versions. You just have to disable unused plugins:

- disable unused issuer modules

- disable notifications if not used
- ...

By default it uses local storage to store its tokens. If you have more than 1 portal and if your load-balancer doesn't keep state, you have to disable this to use the global session storage *(General parameters » portal Parameters » Advanced Parameters » Forms)*. Note that this will decrease performances.

Tip: In production environment for network performance, prefer using minified versions of javascript and css libs: use `make install PROD=yes`. This is done by default in RPM/DEB packages.

Apache::Session performances

Lemonldap::NG handlers use a local cache to store sessions (for 10 minutes). So Apache::Session module is not a problem for handlers. But it can be a bottleneck for the portal:

1. When you use the multiple sessions restriction parameters, sessions are parsed for each authentication unless you use an Apache::Session::Browseable module.
2. Since MySQL does not have always transaction feature, Apache::Session::MySQL has been designed to use MySQL locks. Since MySQL performances are very bad using this, if you want to store sessions in a MySQL database, prefer one of the following

---

Tip: Since 1.9.6, LLNG portal and handler check if session is valid at each access, so purgeCentralCache cron no longer needs to be launched every 10 minutes: one or two times per day is enough.

Replace MySQL by Apache::Session::Flex

In "Apache::Session module" field, set "*Apache::Session::Flex <https://metacpan.org/module/Apache::Session::Flex>*" and use the following parameters:

```
Store      -> MySQL
Lock       -> Null
Generate   -> MD5
Serialize  -> Storable
DataSource -> dbi:mysql:sessions;host=...
UserName   -> ...
Password   -> ...
```

Tip: Since version 1.90 of Apache::Session, you can use Apache::Session::MySQL::NoLock instead

---

Use Apache::Session::Browseable

Apache::Session::Browseable is a wrapper for other Apache::Session modules that add the capability to manage indexes. Prefer versions 1.2.5 for better performances in DB cleaning. To use it (with PostgreSQL for example), choose "Apachedoc:*Session::Browseable<session::browseable>*::Postgres" as "Apache::Session module" and use the following parameters:

```
DataSource -> dbi:Pg:database=sessions;host=...
UserName   -> user
Password   -> password
Index      -> ipAddr uid
```

Note that Apache::Session::Browseable::MySQL doesn't use MySQL locks.

Look at *Browseable session backend* to known which index to choose.

Attention: Some Apache::Session module are not fully usable by Lemonldap::NG such as Apache::Session::Memcached since these modules do not offer capability to browse sessions. They does not allow one to use sessions explorer neither manage one-off sessions.

Performance test

Tip: A Apache::Session::Browseable::Redis has been created, it is the fastest (except for session explorer, defeated by Apache::Session::Browseable:: DBI / LDAP)

This test isn't an "only-backend" test but embedded some LLNG methods, so real differences between engines are mitigate here.

Backend		Portal and handlers			Session explorer and one-off sessions		
Name	Configuration	Insert 1000	Search 1	Purge 500	Parse all	Search by substring	Search by UID
Apache::Session::**Browseable::LDAP**		159.66	0.0120	49.22	0.1110	0.0076	0.0050
Apache::Session::**MySQL**	No lock	87.20	**0.0039**	23.14	0.0281	0.0252	0.0235
Apache::Session::**Browseable::MySQL**		91.79	**0.0039**	**0.139**[2]	0.0272	**0.0036**	**0.0026**
Apache::Session::**Browseable::MySQLJSON**		86.06	0.0145	** 0.151**[3]	**0.0104**	0.0137	0.0038
Apache::Session::**Postgres**		18.31	0.0095	13.40	0.0323	0.0277	0.0264
Apache::Session::**Postgres**	Unlogged table	9.16	0.0095	7.91	0.0318	0.0270	0.0254
Apache::Session::**Browseable::Postgres**	Unlogged table with indexes	9.24	0.0094	**0.103**?	0.0301	**0.0036**	**0.0028**
Apache::Session::**Browseable::PgJSON**	Unlogged table, json field	9.25	0.0091	**0.108**?	0.0247	**0.0035**	**0.0029**
Apache::Session::**Browseable::PgJSON**	Unlogged table, jsonb field	9.25	0.0091	**0.105**?	**0.0126**	**0.0034**	**0.0029**
Apache::Session::**Browseable::PgHstore**	Unlogged table, hstore field	9.62	0.0111	**0.105**?	**0.0125**	**0.0033**	**0.0029**
Apache::Session::**Redis**		**2.13**	**0.0033**	1.158	0.0623	0.0570	0.0550
Apache::Session::**Browseable::Redis**		**2.36**	**0.0033**	1.154	0.0643	0.1048	**0.0024**

The source of this test is available in sources: e2e-tests/sbperf.pl

Analysis:

- LDAP servers are "write-once-read-many", so write performances are very bad. Don't use this on heavy load if "Session activity timeout" is enabled *(if set, handler "write" sessions)*
- MySQL/MariaDB is better to read than to write. Prefer PostgreSQL if you use "Session activity timeout"
- Logged tables decrease a lot insert performances with PostgreSQL, so use unlogged tables for sessions except for persistent sessions
- Redis is the best for main usage
- Browseable::Postgres/PgHstore/PgJSON are the best SQL solutions on average

LDAP performances

LDAP server can slow you down when you use LDAP groups retrieval. You can avoid this by setting "memberOf" fields in your LDAP scheme:

```
dn: uid=foo,dmdName=people,dc=example,dc=com
...
memberOf: cn=admin,dmdName=groups,dc=example,dc=com
memberOf: cn=su,dmdName=groups,dc=example,dc=com
```

So instead of using LDAP groups retrieval, you just have to store "memberOf" field in your exported variables. With OpenLDAP, you can use the memberof overlay to do it automatically.

Attention: Don't forget to create an index on the field used to find users (uid by default)

Tip: To avoid storing the full group DNs in session data, you can use a macro to rewrite `memberOf`:

- In *Exported variables*, export the `memberOf` LDAP attribute as a `ldapGroups` session variable
  - key: `ldapGroups`
  - value: `memberOf`
- Next, add a `ldapGroups` macro that will overwrite the exported attribute
  - key: `ldapGroups`
  - value:

```
join("; ",($ldapGroups =~ /cn=(.*?),/g))
```

`ldapGroups` should now contain something like `admin; su` just like it would if you had used the regular, slower group resolution mechanism.

You can use *listMatch($ldapGroups, "some_group")* in your access rules.

---

[2] "purge" test is done with Apache::Session::Browseable-1.2.5 and LLG-2.0. Earlier results are not so good.
[3] "purge" test is done with Apache::Session::Browseable-1.2.6 and LLG-2.0.

NGINX performances

To increase launch by web browser, for example to load js, css, or fonts, Gzip compression can be activated.

Edit file /etc/nginx/mime.types Check those lines or add :

```
application/vnd.ms-fontobject    eot;
application/x-font-ttf           ttf;
application/font-woff            woff;
font/opentype                    ott;
```

Edit file /etc/nginx/nginx.conf

```
gzip on; # active la compression Gzip
gzip_disable "msie6";

gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_min_length 128;
gzip_types text/plain text/css application/json application/javascript application/x-
↪javascript text/xml application/xml application/rss+xml text/javascript application/
↪vnd.ms-fontobject application/x-font-ttf font/opentype image/jpeg image/png image/
↪svg+xml image/x-icon;
```

Restart NGINX and watch web-browser console.

18.1.5 Manager performances

Disable unused modules

In lemonldap-ng.ini, set only modules that you will use. By default, configuration, sessions explorer, notifications explorer and second factor are enabled. Example:

```
[manager]
enabledModules = conf, sessions
```

Enable compactConf parameter

By enabling compactConf option, all unused configuration parameters are removed. Could be usefull to shrink lemonldap-ng configuration file and save space.

Go in Manager, `General Parameters` » `Configuration reload` » ''Compact configuration file '' and set to `On`.

Use static HTML files

Once Manager is installed, browse enabled modules (configuration, sessions, notifications) and save the web pages respectively under `manager.html`, `sessions.html` and `notifications.html` in the `DocumentRoot` directory. Then replace this in Manager file of Apache configuration:

```
RewriteRule "^/$" "/psgi/manager-server.fcgi" [PT]
# DirectoryIndex manager.html
# RewriteCond "%{REQUEST_FILENAME}" "!\.html$"
RewriteCond "%{REQUEST_FILENAME}" "!^/(?:static|doc|lib).*"
RewriteRule "^/(.+)$" "/psgi/manager-server.fcgi/$1" [PT]
```

by:

```
# RewriteRule "^/$" "/psgi/manager-server.fcgi" [PT]
DirectoryIndex manager.html
RewriteCond "%{REQUEST_FILENAME}" "!\.html$"
RewriteCond "%{REQUEST_FILENAME}" "!^/(?:static|doc|lib).*"
RewriteRule "^/(.+)$" "/psgi/manager-server.fcgi/$1" [PT]
```

So manager HTML templates will be no more generated by Perl but directly given by the web server.

18.2 Security recommendation

18.2.1 Secure configuration access

Configuration can be stored in several formats (*SQL*, *File*, *LDAP*) but must be shared over the network if you use more than 1 server. If some of your servers are not in the same (secured) network than the database, it is recommended to use *SOAP access* for those servers.

Tip: You can use different type of access: *SQL*, *File* or *LDAP* for servers in secured network and *SOAP* for remote servers.

Next, you have to configure the SOAP access as described *here* since SOAP access is denied by default.

18.2.2 Protect the Manager

By default, the manager is restricted to the user 'dwho' (default backend is Demo). To protect the manager, you have to choose one or both of :

- protect the manager by Apache configuration
- protect the manager by LL::NG

Protect the Manager by the web server

You can use any of the mechanisms proposed by Apache: SSL, Auth-Basic, Kerberos,... Example

```
<VirtualHost *:443>
    ServerName manager.example.com
    # SSL parameters
    ...
    # DocumentRoot
    DocumentRoot /var/lib/lemonldap-ng/manager/
    <Location />
        AuthType Basic
        AuthName "Lemonldap::NG manager"
        AuthUserFile /usr/local/apache/passwd/passwords
        Require user rbowen
        Order allow,deny
        Deny from all
        Allow from 192.168.142.0/24
        Options +ExecCGI
    </Location>
</VirtualHost>
```

Protect the Manager by LL::NG

To protect the manager by LL::NG, you just have to set this in `lemonldap-ng.ini` configuration file (section [manager]):

```
[manager]
protection = manager
```

Attention: Before, you have to create the virtual host `manager.your.domain` in the manager and set a *rule*, else access to the manager will be denied.

18.2.3 Portal

LLNG portal now embeds the following features:

- CSRF protection *(Cross-Site Request Forgery)*: a token is build for each form. To disable it, set 'require Token for forms' to Off *(portal security parameters in the manager)*. Token timeout can be defined via manager (default to 120 seconds)
- Brute-force attack protection: after some failed logins, user must wait before re-try to log into Portal
- Content-Security-Policy header: portal builds dynamically this header. You can modify default values in the manager *(General parameters » Advanced parameters » Security » Content-Security-Policy)*
- Cross-Origin Resource Sharing headers: CORS is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. A web page may freely embed cross-origin images, stylesheets, scripts, iframes, and videos. Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy. You can modify default values in the manager *(General parameters » Advanced parameters » Security » Cross-Origin Resource Sharing)*

Attention:

- Brute-force attack protection is DISABLED by default
- Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify formAction value with wildcard likes *.

Split portal when using SOAP/REST

If you use *SOAP* or *REST* session backend, dedicate a portal especially for these internal requests.

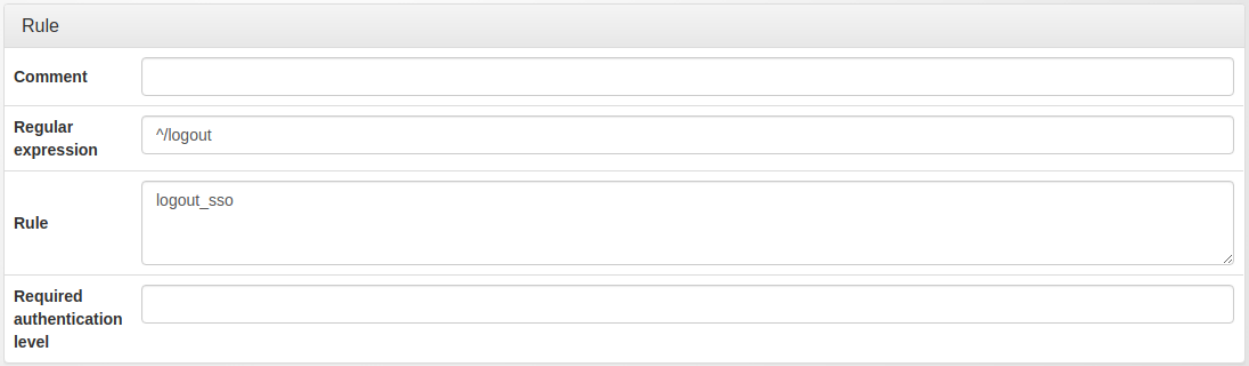
18.2.4 Write good rules

Order your rules

Rules are applied in alphabetical order (comment and regular expression). The first matching rule is applied.

Attention: The "default" rule is only applied if no other rule matchs

The Manager let you define comments in rules, to order them:



For example, if these rules are used without comments:

Regular expression	Rule	Comment
^/pub/admin/	$uid eq "root"	
^/pub/	accept	

Then the second rule will be applied first, so every authenticated user will access to `/pub/admin` directory.

Use comment to correct this:

Regular expression	Rule	Comment
^/pub/admin/	$uid eq "root"	1_admin
^/pub/	accept	2_pub

Tip:

- Reload the Manager to see the effective order

- Use rule comments to order your rules

Be careful with URL parameters

You can write *rules* matching any component of URL to protect including GET parameters, but be careful.

For example with this rule on the `access` parameter:

Regular expression	Rule	Comment
`^/index.php\?.*access=admin`	`$groups =~ /\badmin\b/`	
`default`	`accept`	

Then a user that try to access to one of the following will be granted !

- `/index.php?access=admin&access=other`
- `/index.php?Access=admin`

You can use the following rules instead:

Regular expression	Rule	Comment
`^/(?i)index.php\?.*access.*access`	`deny`	0_bad
`^/(?i)index.php\?.*access=admin`	`$groups =~ /\badmin\b/`	1_admin
`default`	`accept`	

Tip: **(?i)** means case no sensitive.

Danger: Remember that rules written on GET parameters must be tested.

Encoded characters

Some characters are encoded in URLs by the browser (such as space,...). To avoid problems, LL::NG decode them using https://metacpan.org/pod/Apache2::URI#unescape_url. So write your rules using normal characters.

IP in rules

Danger: If you are running LemonLDAP::NG behind a reverse proxy, make sure you check the *Reverse Proxy how-to* so that the rule applies to the real user IP and not the reverse proxy's IP. Make sure you only specify trusted proxy addresses so that an attacker cannot forge the `X-Forwarded-For` header

18.2.5 Secure reverse-proxies

LL::NG can protect any Apache hosted application including Apache reverse-proxy mechanism. Example:

```
PerlOptions +GlobalRequest
PerlRequire /var/lib/lemonldap-ng/handler/MyHandler.pm
<VirtualHost *:443>
    SSLEngine On
    ... other SSL parameters ...
    PerlInitHandler My::Handler
    ServerName appl1.example.com
    ProxyPass / http://hiddenappl1.example.com/
    ProxyPassReverse / http://hiddenappl1.example.com/
    ProxyPassReverseCookieDomain / http://hiddenappl1.example.com/
</VirtualHost>
```

See mod_proxy and mod_rewrite documentation for more about configuring Apache reverse-proxies.

Such configuration can have some security problems:

- if a user can access directly to the hidden application, it can bypass LL::NG protection
- if many hidden applications are on the same private network, if one is corrupted (by SQL injection, or another attack), the hacker will be able to access to other applications without using reverse-proxies so it can bypass LL::NG protection

It is recommended to secure the channel between reverse-proxies and application to be sure that only request coming from the LL::NG protected reverse-proxies are allowed. You can use one or a combination of:

- firewalls (but be careful if more than 1 server is behind the firewall)
- server based restriction (like Apache "allow/deny" mechanism)
- SSL client certificate for the reverse-proxy (see SSLProxy* parameters in mod_ssl documentation)

18.2.6 Configure security settings

Go in Manager, `General parameters` » `Advanced parameters` » `Security`:

- **Username control**: Regular expression used to check user login syntax.
- **Avoid browsers to store users password**: Enable this option to prevent browsers from prompting users to save passwords.
- **Force authentication**: set to 'On' to force authentication when user connects to portal, even if he has a valid session.
- **Force authentication interval**: time interval (in seconds) when an authentication renewal cannot be forced, used to prevent to loose the current authentication during the main process. If you experience slow network performances, you can increase this value.
- **Encryption key**: key used to crypt some data, should not be known by other applications
- **Trusted domains**: domains on which the user can be redirected after login on portal.
  - Example: `myapp.example.com .subdomain.example.com`
  - * allows redirections to any external domain (DANGEROUS)
- **Use Safe jail**: set to 'Off' to disable Safe jail. Safe module is used to eval expressions in headers, rules, etc. Disabling it can lead to security issues.
- **Avoid assignment in expressions**: Set to 'Off' to disable syntax checking. Equal sign can be replaced by x3D i.e. "dcx3Dorg"

- **Check XSS Attacks**: Set to 'Off' to disable XSS checks. XSS checks will still be done with warning in logs, but this will not prevent the process to continue.
- **Required token for forms**: To prevent CSRF attack, a token is build for each form. To disable it, set this parameter to 'Off' or set a special rule
- **Form timeout**: Form token timeout (default to 120 seconds)
- **Use global storage**: Local cache is used by default for one time tokens. To use global storage, set it to 'On'
- **CrowdSec Bouncer**: set to 'On' to enable *CrowdSec Bouncer plugin*
- **Brute-Force Attack protection**: set to 'On' to enable *Brute-force protection plugin*
- **LWP::UserAgent and SSL options**: insert here options to pass to LWP::UserAgent object (used by SAML or OpenID-Connect to query partners and AuthSSL or AuthBasic handler to request Portal URL). Example: `verify_hostname => 0`, `SSL_verify_mode => 0`
- **Content Security Policy**: Portal builds dynamically this header. You can modify default values. Browser implementations of formAction directive are inconsistent (e.g. Firefox doesn't block the redirects whereas Chrome does). Administrators may have to modify `formAction` value with wildcard likes *.
- **Cross-Origin Resource Sharing**: Portal builds those headers. You can modify default values. Administrators may have to modify `Access-Control-Allow-Origin` value with ' '.

Attention: If URLs are protected with AuthBasic handler, you have to disable CSRF token by setting a special rule based on callers IP address like this :

requireToken => $env->{REMOTE_ADDR} && $env->{REMOTE_ADDR} !~ /^127.0.[1-3].1$/

Danger: Enable global storage for one time tokens will downgrade Portal performance!!!

Must ONLY be use with outdated or low performance Load Balancer.

18.2.7 Fail2ban

To prevent brute force attack with fail2ban

Edit /etc/fail2ban/jail.conf

```
[lemonldap-ng]
enabled = true
port    = http,https
filter  = lemonldap
action   = iptables-multiport[name=lemonldap, port="http,https"]
logpath = /var/log/apache*/error*.log
maxretry = 3
```

and edit /etc/fail2ban/filter.d/lemonldap.conf

```
# Fail2Ban configuration file
#
# Author: Adrien Beudin
#
# $Revision: 2 $
```

(continues on next page)

(continued from previous page)

```
#

[Definition]

# Option:  failregex
# Notes.:  regex to match the password failure messages in the logfile. The
#          host must be matched by a group named "host". The tag "<HOST>" can
#          be used for standard IP/hostname matching and is only an alias for
#          (?f{4,6}:)?(?P<host>[\w\-.^_]+)
# Values:  TEXT
#
failregex = Lemonldap\:\:NG \: .* was not found in LDAP directory \(<HOST>\)
            Lemonldap\:\:NG \: Bad password for .* \(<HOST>\)

# Option:  ignoreregex
# Notes.:  regex to ignore. If this regex matches, the line is ignored.
# Values:  TEXT
#
ignoreregex =
```

Restart fail2ban

18.2.8 Sessions identifier

You can change the module used for sessions identifier generation. To do, add `generateModule` key in the configured session backend options.

We recommend to use : `Lemonldap::NG::Common::Apache::Session::Generate::SHA256`.

18.2.9 SAML

See *samlservice#security_parameters*

18.3 SELinux

To make LemonLDAP::NG work with SELinux, you may need to set up some options.

18.3.1 SELinux policy package

If you are using a RPM distribution and Apache as the web server, you need to install the `lemonldap-ng-selinux` package to configure SELinux context correctly

```
yum install lemonldap-ng-selinux
```

Note: On CentOS 8 and Fedora, this is done automatically

This package will not configure SELinux booleans, please read the next sections to see which booleans you need to enable manually

18.3.2 Disk cache (sessions an configuration)

You need to set the correct context on the cache directory

Deprecated since version 2.0.10: this is now done by the `lemonldap-ng-selinux` package

```
semanage fcontext --add -t httpd_cache_t -f a '/var/cache/lemonldap-ng(/.*)?'
restorecon -R /var/cache/lemonldap-ng/
```

18.3.3 LDAP

```
setsebool -P httpd_can_connect_ldap 1
```

18.3.4 Databases

```
setsebool -P httpd_can_network_connect_db 1
```

18.3.5 Memcache

```
setsebool -P httpd_can_network_memcache 1
```

18.3.6 Proxy HTTP

```
setsebool -P httpd_can_network_relay 1
```

18.4 Monitoring

18.4.1 MRTG monitoring

The status page can be read by MRTG using the script **lmng-mrtg** that can be found in manager example directory.

MRTG configuration example:

```
##################################################################
# Multi Router Traffic Grapher -- Sample Configuration File
##################################################################
# This file is for use with mrtg-2.5.4c

# Global configuration
WorkDir: /var/www/mrtg
WriteExpires: Yes

Title[^]: Traffic Analysis for

# 128K leased line
```

(continues on next page)

(continued from previous page)

```
# ----------------
#Title[leased]: a 128K leased line
#PageTop[leased]: <H1>Our 128K link to the outside world</H1>
#Target[leased]: 1:public@router.localnet
#MaxBytes[leased]: 16000
Target[test.example.com]: `/etc/mrtg/lmng-mrtg 172.16.1.2 https://test.example.com/
↪status OK OK`
Options[test.example.com]: nopercent, growright, nobanner, perminute
PageTop[test.example.com]: <h1>Requests OK from test.example.com</h1>
MaxBytes[test.example.com]: 1000000
YLegend[test.example.com]: hits/minute
ShortLegend[test.example.com]:   hits/mn
Legend0[test.example.com]: Hits:
Legend1[test.example.com]: Hits:
Legend2[test.example.com]: Hits per minute
Legend4[test.example.com]: Hits max per minute
Title[test.example.com]: Hits per minute
WithPeak[test.example.com]: wmy
```

Handler can be monitored by using MRTG. See *MRTG monitoring*.

Portal can also publish its status using REST. To enable it, go to the manager, general parameters, plugins. Then enable "publish portal status" option.

Then protect http://auth.yourdomain/portalStatus in webserver configuration.

This REST URL just publishes a hash containing number of sessions of each type.

18.5 Logs

18.5.1 Presentation

Main settings:

- **REMOTE_USER** : session attribute used for logging user access
- **REMOTE_CUSTOM** : can be used for logging an another user attribute or a macro (optional)
- **Hidden attributes** : session attributes never displayed or sent

LemonLDAP::NG provides 5 levels of error and has two kind of logs:

- technical logs
- user actions logs

Each category can be handle by a different logging framework. You can choose between:

- **Lemonldap::NG::Common::Logger::Std**: standard output (mapped in web server logs, see below)
- **Lemonldap::NG::Common::Logger::Syslog**: syslog logging
- **Lemonldap::NG::Common::Logger::Apache2**: use Apache2 logging, levels are stored in Apache2 logs and the log level is defined by `LogLevel` Apache parameter
- **Lemonldap::NG::Common::Logger::Log4perl**: use `Log4perl` framework to log *(inspired by Java Log4J)*
- **Lemonldap::NG::Common::Logger::Sentry (experimental)**: use Sentry to store logs

- **Lemonldap::NG::Common::Logger::Dispatch**: dispatch logs in other backends depending on log level

Attention: Except for Apache2 and Log4Perl, log level is defined by `logLevel` parameter set in `lemonldap-ng.ini` file. Logger configurations are defined in lemonldap-ng.ini. Example:

```
[all]
logger     = Lemonldap::NG::Common::Logger::Log4perl
userLogger = Lemonldap::NG::Common::Logger::Syslog
logLevel   = notice
```

You can also modify these values in each lemonldap-ng.ini section to have different values for portal, manager and handlers.

Therefore, LLNG provides a username that can be used by webservers in their access log. To configure the user identifier to write into access logs, go into Manager, `General Parameters` > `Logging` > `REMOTE_USER`.

18.5.2 User log samples

Note: The user name set in user log messages is configured with *whatToTrace* parameter, except for messages corresponding to failed authentification, whe the user name logged is the login used by the user.

Authentication:

```
[notice] Session granted for dwho by LDAP (81.20.13.21)
[notice] User dwho.com successfully authenticated at level 2
[notice] dwho connected
```

Failed authentication:

```
[warn] foo.bar was not found in LDAP directory (81.20.13.21)
[warn] Bad password for dwho (81.20.13.21)
```

Failed authentication with Combination module:

```
[warn] All schemes failed for user dwho (81.20.13.21)
```

Logout:

```
[notice] User dwho has been disconnected from LDAP (81.20.13.21)
```

Password change:

```
[notice] Password changed for dwho (81.20.13.21)
```

Access to a CAS application non registered in configuration (when CAS server is open):

```
[notice] User dwho is redirected to https://cas.service.url
```

Access to a CAS application whose configuration key is `app-example`:

```
[notice] User dwho is authorized to access to app-example
```

Access to an SAML SP whose configuration key is `sp-example`:

```
[notice] User dwho is authorized to access to sp-example
```

Access to an OIDC RP whose configuration key is `rp-example`:

```
[notice] User dwho is authorized to access to rp-example
```

Access to a Get application whose vhost configuration key is `host.example.com`:

```
[notice] User dwho is authorized to access to host.example.com
```

18.5.3 Default loggers

- Apache handlers use by default Apache2 logger. This logger can't be used for other LLNG components
- Except when launched by LLNG FastCGI server *(used by Nginx)*, Portal and Manager use Std logger by default
- All components launched by LLNG FastCGI server use Syslog by default

18.5.4 Log levels

Technical log levels

- **error** is used for problems that must be reported to administrator and needs an action. In this case, some feature may not work
- **warn** is used for problems that doesn't block LLNG features but should be solved
- **notice** is used for actions that must be kept in logs
- **info** display some technical information
- **debug** produce a lot a debugging logs

Log levels for user actions

- **error** is used to log bad user actions that looks malicious
- **warn** is used to log some errors like "bad password"
- **notice** is used for actions that must be kept in logs for accounting (connections, logout)
- **info** display some useful information like handler authorizations (at least 1 for each HTTP hit)
- **debug** isn't used

18.5.5 Logger configuration

Std logger

Nothing to configure except logLevel.

Apache2 logger

The log level can be set with Apache `LogLevel` parameter. It can be configured globally, or inside a virtual host.

See http://httpd.apache.org/docs/current/mod/core.html#loglevel for more information.

Syslog

You can choose facility in lemonldap-ng.ini file. Default values:

```
syslogFacility     = daemon
userSyslogFacility = auth
```

You can also override options. Default values:

```
syslogOptions      = cons,pid,ndelay
userSyslogOptions  = cons,pid,ndelay
```

Tip: You can find more information on Syslog options in Sys::Syslog Perl module.

Log4perl

You can indicate the Log4perl configuration file and the classes to use. Default values:

```
log4perlConfFile   = /etc/log4perl.conf
log4perlLogger     = LLNG
log4perlUserLogger = LLNG.user
```

Sentry

You just have to give your DSN:

```
sentryDsn = https://...
```

Attention: This experimental logger requires Sentry::Raven Perl module.

Dispatch

Use it to use more than one logger. Example:

```
logger               = Lemonldap::NG::Common::Logger::Dispatch
userLogger           = Lemonldap::NG::Common::Logger::Dispatch
logDispatchError     = Lemonldap::NG::Common::Logger::Sentry
logDispatchNotice    = Lemonldap::NG::Common::Logger::Syslog
userLogDispatchError = Lemonldap::NG::Common::Logger::Sentry
; Other parameters
syslogFacility    = daemon
sentryDsn         = https://...
```

Attention: At least `logDispatchError` (or `userLogDispatchError` for user logs) must be defined. All sub level will be dispatched on it, until another lever is declared. In the above example, Sentry collects `error` and `warn` levels and all user actions, while syslog stores technical `notice`, `info` and `debug` logs.

18.6 Error messages

Note: This page do not reference all error messages, but only the most common

18.6.1 Lemonldap::NG::Common

```
Warning: key is not defined, set it in the manager !
```

→ LemonLDAP::NG uses a key to crypt/decrypt some datas. You have to set its value in Manager. This message is displayed only when you upgrade from a version older than 1.0

```
Can't locate /usr/share/lemonldap-ng/configStorage.pl
```

→ When you upgrade from Debian Lenny with customized index.pl files, you must upgrade them.

18.6.2 Lemonldap::NG::Handler

```
Unable to clear local cache
```

→ Local cache cannot be cleared, check the localStorage and localStorageOptions or file permissions

```
Status module can not be loaded without localStorage parameter
```

→ You tried to activate Status module without localStorage. Configure local cache first.

```
No configuration found
```

→ The configuration cannot be loaded. Check configStorage and configStorageOptionsor file permissions.

```
User rejected because VirtualHost XXXX has no configuration
```

→ The specified virtual host is not configured in Manager.

```
mkdir /tmp/MyNamespace/2: Permission denied ...
```

→ The cache has been created by another user than Apache's user. Restart Apache to purge it.

Attention: This can append when you use lmConfigEditor or launch **cron files** with a different user than Apache process. That is why it is important to set APACHEUSER variable when you launch "make install"

```
Lemonldap::NG::Handler::SharedConf: No cookie found
```

→ User does not have Lemonldap::NG cookie, handler redirect it to the portal

```
The cookie $id isn't yet available: Object does not exist in the data store
```

→ User session has expired or handler does not have access to the same Apache::Session database than the portal

```
Firefox has detected that the server is redirecting the request for this address in a␣
↪way that will never complete
```

→ Your browser loops between portal and handler, it is probably a cookie problem. Verify that:

- the portal is in the declared domain
- CDA is set if the handler is not in the same domain
- portal is in a https virtualhost if securedCookie is set
- you've restart all Apache server after having change cookie name or domain

18.6.3 Lemonldap::NG::Manager

```
XXXX was not found in tree
```

→ The specified node is not the uploaded tree.

18.6.4 Lemonldap::NG::Portal

```
User XXXX was not granted to open session
```

→ Check grantSessionRule parameter.

```
XML menu configuration is deprecated. Please use lmMigrateConfFiles2ini to migrate your␣
↪menu configuration
```

→ You do not use the new configuration syntax for application list. XML file is no more accepted.

```
Apache is not configured to authenticate users !
```

→ You use the Apache authentication backend, but Apache is not or bad configured (no REMOTE_USER send to LemonLDAP::NG).

```
URL contains a non protected host
```

→ The host is not known by LemonLDAP::NG. Add it to trustedDomains (or set * in trustedDomains to accept all).

```
XSS attack detected
```

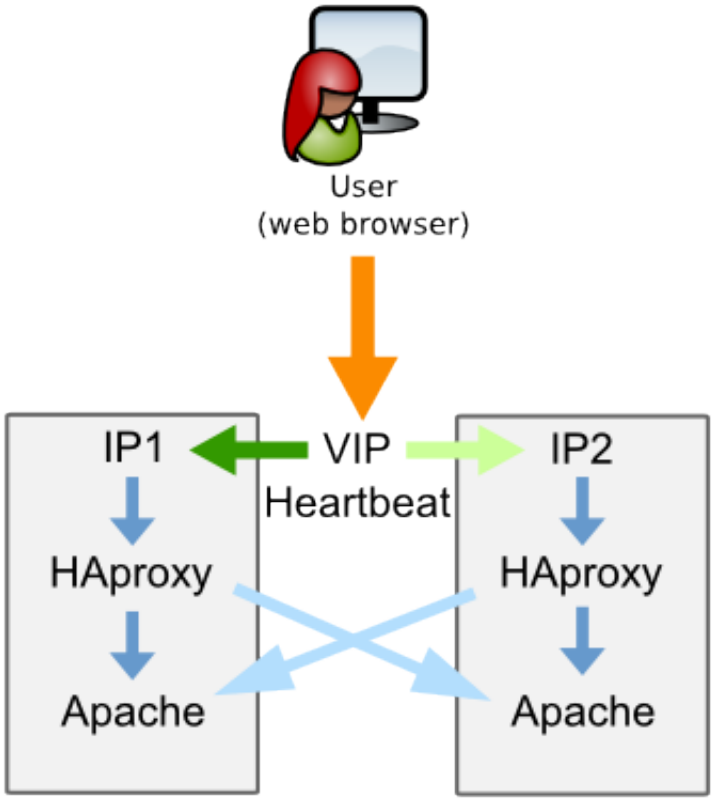
→ Some URL parameters contain forbidden characters.

18.7 High availability

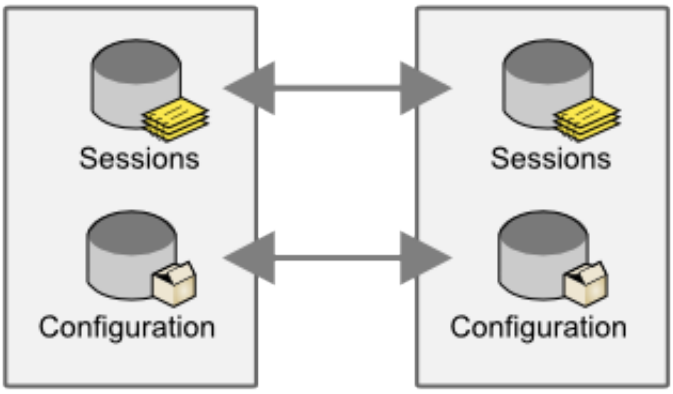
LemonLDAP::NG is highly scalable, so easy to insert behind a load-balancer:

- Portal does not store any data outside the session database, so you can have many portal servers using the same HTTP host name
- All handlers download the whole configuration, so many servers can serve the same virtual hosts

You can for example set up a fail-over cluster with Heartbeat and HAproxy, like this:



You just have to share configuration and sessions databases between those servers:


Sessions
Configuration
Sessions
Configuration

CHAPTER

NINETEEN

DEVELOPMENT

19.1 How to report a bug

If you don't know if the problem is a bug, first try to contact us on lemonldap-ng-users@ow2.org.

19.1.1 Before reporting bug

First, verify that the bug isn't known; the list is here: Known Lemonldap::NG bugs.

Bug reports must provide enough information for developers. So here are the steps:

- set log level to debug:
  - in lemonldap-ng.ini, section [all]
  - for Apache users, set also "LogLevel perl:debug" or "LogLevel debug" in your httpd.conf
- restart the web server
- replay the sequence that fails

19.1.2 Report the bug

- Go to https://gitlab.ow2.org/lemonldap-ng/lemonldap-ng/issues
- Create an account if you don't have one
- Create the the bug using "New issue" button
- Fill the form:
  - don't forget to set the version you're using
  - explain the sequence that generates the bug
  - attach the log file *(or part of it if it's enough)*
- Submit the form

Do you want to contribute to LemonLdap::NG project ?

19.2 Contribute to Project

LemonLDAP::NG is mostly written in Perl and Javascript. Community applies the following rules:

- Perl:
  - code must be written in modern object-oriented code (using Mouse) *(except handler and Apache::Session inheritance)*
  - code must be formatted using perltidy version 20181120 *(from Debian/buster)*
- Javascript:
  - code must be written in CoffeeScript (in `<component>/site/coffee`): `make minify` will generate JS files

19.2.1 Configure SSH

On Debian developper station :

```
ssh-keygen -o -t rsa -b 4096 -C "your@email"
```

Go to your gitlab account : https://gitlab.ow2.org/profile/keys

```
cat ~/.ssh/id_rsa.pub
```

Copy id_rsa.pub content to key section and enter a name into "Title" and click "Add key" button. Test ssh connexion :

```
ssh -T git@gitlab.com
```

Accept messages

19.2.2 Install basic tools

Debian

As *root :*

```
apt install aptitude
aptitude install vim make devscripts yui-compressor git git-gui libjs-uglify␣
↪coffeescript cpanminus autopkgtest pkg-perl-autopkgtest
aptitude install libauth-yubikey-webclient-perl libnet-smtp-server-perl libtime-fake-
↪perl libtest-output-perl libtest-pod-perl libtest-leaktrace-perl

cpanm Authen::U2F Authen::U2F::Tester Crypt::U2F::Server::Simple

curl -sL https://deb.nodesource.com/setup_9.x | bash -
apt-get install -y nodejs

npm install -g protractor # end-2-end tests
webdriver-manager update # install/update selenium driver
```

Configure Git

As *user :*

```
git config --global user.name "Name Surname"
git config --global user.email "your@mail"
git config --global core.editor vim
git config --global merge.tool vimdiff
git config --global color.ui true
git config --list
```

Import Project and using Git

As *user*, create directory in directory:

```
git clone git@gitlab.ow2.org://user///lemonldap-ng.git
cd lemonldap-ng/
git log
git checkout master # go to master branch
git remote add upstream https://gitlab.ow2.org/lemonldap-ng/lemonldap-ng.git # to␣
↪connect to remote branch
git fetch upstream  # import branch
git checkout v2.0 # to change branch
git fetch upstream
```

Import version branch on linux station:

```
git checkout v2.0
git fetch upstream
git rebase upstream/v2.0 # to align to parent project remote branch
```

On gitlab, create working branch, one per thematic on linux station:

```
git checkout workingbranch
git log
git status
git commit -am "explanations (#number gitlab ticket)"
git commit --amend file(s) # to modify a commit
git rebase v2.0 # align local working branch to local 2.0
git checkout -- file(s) # revert
git push # to send on remote working branch ! Only after doing some commits !
```

On gitlab, submit merge request when tests are corrects.

19.2.3 Install dependencies

```
aptitude install libapache-session-perl libcache-cache-perl libclone-perl libconfig-
↪inifiles-perl libconvert-pem-perl libcrypt-openssl-bignum-perl libcrypt-openssl-rsa-
↪perl libcrypt-openssl-x509-perl libcrypt-rijndael-perl libdbi-perl libdigest-hmac-perl␣
↪libemail-sender-perl libgd-securityimage-perl libhtml-template-perl libio-string-perl␣
↪libjson-perl libmime-tools-perl libmouse-perl libnet-ldap-perl libplack-perl libregexp-
↪assemble-perl libregexp-common-perl libsoap-lite-perl libstring-random-perl libtext-
↪unidecode-perl libunicode-string-perl liburi-perl libwww-perl libxml-simple-perl␣
↪libxml-libxslt-perl libcrypt-urandom-perl libconvert-base32-perl cpanminus
aptitude install apache2 libapache2-mod-fcgid libapache2-mod-perl2  # install Apache
aptitude install nginx nginx-extras  # install Nginx
cpanm perltidy@20181120
```

For SAML:

```
aptitude install liblasso-perl libglib-perl
```

19.2.4 Working Project

Configure hosts file

```
echo '127.0.0.1       auth.example.com manager.example.com test1.example.com test2.
↪example.com' >> /etc/hosts
```

Unit tests

Launch unit tests:

```
make test # or manager_test, portal_test, ... to launch unit tests
```

Same tests launched on a simulated install

```
make autopkgtest # or autopkg_portal, autopkg_manager, ... to launch unit tests
```

Execute an unit test :

```
# Building project
cd ~/lemonldap-ng/; make
# Go to parent test directory
cd ~/lemonldap-ng/lemonldap-ng-portal
# and execute the unit test:
prove -v t/67-CheckUser.t
```

Launch tests with LDAP backend, for example with OpenLDAP LTB package (https://ltb-project.org/documentation):

```
make LLNGTESTLDAP=1 LLNGTESTLDAP_SLAPD_BIN=/usr/local/openldap/libexec/slapd␣
↪LLNGTESTLDAP_SLAPADD_BIN=/usr/local/openldap/sbin/slapadd LLNGTESTLDAP_SCHEMA_DIR=/usr/
↪local/openldap/etc/openldap/schema/ test
```

Other commands

```
make start_web_server # TESTUSESSL=1 to enable SSL engine (only available for Apache)
make start_web_server TESTWEBSERVER=nginx # to use Nginx web server
make stop_web_server
make reload_web_server # to reload LL:NG conf
make clean # to clean test files
make minify # to minify and compile coffeescript
make json # to build conf and manager tree
make manifest # to update manifest
make tidy # to magnify perl files (perl best pratices)
```

Documentation

Install dependencies:

```
apt install python3-sphinx python3-sphinx-bootstrap-theme
```

Then edit sources in doc/sources/admin.

You can check the result with:

```
make documentation
firefox doc/pages/documentation/current/start.html
```

19.3 Handler libraries architecture

Handlers are build on rows of modules:

- Applications or launchers that get the request and choose the right type *(Main, AuthBasic, ZimbraPreAuth,...)* and launch it *(may not inherits from other Handler::* modules)*
- Wrappers that call "type" library and platform "Main" //(may all inherits from Platform::Main
- library types if needed *(may inherits from Main)*
- Main: the main handler library

19.3.1 Overview of Handler packages

Usage	Platform	Wrapper	Types	Main
Apache2 protection	ApacheMP2	ApacheMP2::<type>	Lib::<type>	Main
Plack servers protection or Nginx/*SSOaaS* FastCGI/uWSGI server	Server	Server::<type>		
Self protected applications	PSGI	PSGI::<type>		

Types are:

- *(Main)*: link between Main and platform
- *AuthBasic*
- *CDA*

- *DevOps*
- *DevOps+ServiceToken*
- *OAuth2*
- *SecureToken (not available for PSGI)*
- *Service Token (server to server)*
- *ZimbraPreAuth (not available for PSGI)*

19.4 Write a custom plugin

19.4.1 Presentation

Standard entry points

You can now write a custom portal plugin that will hook in the authentication process:

- `beforeAuth`: method called before authentication process
- `betweenAuthAndData`: method called after authentication and before setting "sessionInfo" provisionning
- `afterData`: method called after "sessionInfo" provisionning
- `endAuth`: method called when session is validated (after cookie build)
- `authCancel`: method called when user click on "cancel" during auth process
- `forAuthUser`: method called for already authenticated users
- `beforeLogout`: method called before logout

Extended entry points

If you need to call a method just after any standard method in authentication process, then use `afterSub`, for example:

```
use constant afterSub => {
    getUser => 'mysub',
};
sub mysub {
    my ( $self ,$req ) = @_;
    # Do something
    return PE_OK;
}
```

If you need to call a method instead any standard method in authentication process, then use `aroundSub`, for example:

```
use constant aroundSub => {
    getUser => 'mysub',
};
sub mysub {
    my ( $self, $sub, $req ) = @_;
    # Do something before
    my $ret = $sub->($req);
    # Do something after
```

(continues on next page)

(continued from previous page)

```
    return $ret;
}
```

Hooks

New in version 2.0.10.

Your plugin can also register itself to be called at some points of interest within the main LemonLDAP::NG code.

Available plugin hooks

OpenID Connect Issuer hooks

oidcGotRequest

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG received an authorization request on the */oauth2/authorize* endpoint.

The hook's parameter is a hash containing the authorization request parameters.

Sample code:

```
use constant hook => {
    oidcGotRequest               => 'addScopeToRequest',
};

sub addScopeToRequest {
    my ( $self, $req, $oidc_request ) = @_;
    $oidc_request->{scope} = $oidc_request->{scope} . " my_hooked_scope";

    return PE_OK;
}
```

oidcGotClientCredentialsGrant

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG successfully authorized a *Client Credentials Grant*.

The hook's parameters are:

- A hash of the current session info
- the configuration key of the relying party which is being identified

Sample code:

```
use constant hook => {
    oidcGotClientCredentialsGrant => 'addSessionVariable',
};
```

(continues on next page)

(continued from previous page)

```
sub addSessionVariable {
    my ( $self, $req, $info, $rp ) = @_;
    $info->{is_client_credentials} = 1;

    return PE_OK;
}
```

oidcGenerateCode

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is about to generate an Authorization Code for a Relying Party.

The hook's parameters are:

- A hash of the parameters for the OIDC Authorize request, which you can modify
- the configuration key of the relying party which will receive the token
- A hash of the session keys for the (internal) Authorization Code session

Sample code:

```
use constant hook => {
    oidcGenerateCode              => 'modifyRedirectUri',
};

sub modifyRedirectUri {
    my ( $self, $req, $oidc_request, $rp, $code_payload ) = @_;
    my $original_uri = $oidc_request->{redirect_uri};
    $oidc_request->{redirect_uri} = "$original_uri?hooked=1";
    return PE_OK;
}
```

oidcGenerateUserInfoResponse

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG is about to send a UserInfo response to a relying party on the */oauth2/userinfo* endpoint.

The hook's parameter is a hash containing all the claims that are about to be released.

Sample code:

```
use constant hook => {
    oidcGenerateUserInfoResponse => 'addClaimToUserInfo',
};

sub addClaimToUserInfo {
    my ( $self, $req, $userinfo ) = @_;
    $userinfo->{"userinfo_hook"} = 1;
    return PE_OK;
}
```

oidcGenerateIDToken

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG is generating an ID Token.

The hook's parameters are:

- A hash of the claims to be contained in the ID Token
- the configuration key of the relying party which will receive the token

Sample code:

```
use constant hook => {
    oidcGenerateIDToken          => 'addClaimToIDToken',
};

sub addClaimToIDToken {
    my ( $self, $req, $payload, $rp ) = @_;
    $payload->{"id_token_hook"} = 1;
    return PE_OK;
}
```

oidcGenerateAccessToken

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is generating an JWT-formatted Access Token

The hook's parameters are:

- A hash of the claims to be contained in the Access Token
- the configuration key of the relying party which will receive the token

Sample code:

```
use constant hook => {
    oidcGenerateAccessToken          => 'addClaimToAccessToken',
};

sub addClaimToAccessToken {
    my ( $self, $req, $payload, $rp ) = @_;
    $payload->{"access_token_hook"} = 1;
    return PE_OK;
}
```

oidcResolveScope

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is resolving scopes.

The hook's parameters are:

- An array ref of currently granted scopes, which you can modify
- The configuration key of the requested RP

Sample code:

```
use constant hook => {
    oidcResolveScope          => 'addHardcodedScope',
};

sub addHardcodedScope{
    my ( $self, $req, $scopeList, $rp ) = @_;
    push @{$scopeList}, "myscope";
    return PE_OK;
}
```

SAML Issuer hooks

samlGotAuthnRequest

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG has received a SAML login request

The hook's parameter is the Lasso::Login object

Sample code:

```
use constant hook => {
   samlGotAuthnRequest => 'gotRequest',
};

sub gotRequest {
    my ( $self, $res, $login ) = @_;

    # Your code here
}
```

samlBuildAuthnResponse

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG is about to build a response to the SAML login request

The hook's parameter is the Lasso::Login object

Sample code:

```
use constant hook => {
   samlBuildAuthnResponse => 'buildResponse',
};

sub buildResponse {
    my ( $self, $res, $login ) = @_;

    # Your code here
}
```

samlGotLogoutRequest

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG has received a SAML logout request

The hook's parameter is the Lasso::Logout object

Sample code:

```
use constant hook => {
   samlGotLogoutRequest => 'gotLogout',
};

sub gotLogout {
    my ( $self, $res, $logout ) = @_;

    # Your code here
}
```

samlGotLogoutResponse

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG has received a SAML logout response

The hook's parameter is the Lasso::Logout object

Sample code:

```
use constant hook => {
   samlGotLogoutResponse => 'gotLogoutResponse',
};

sub gotLogoutResponse {
```

(continues on next page)

(continued from previous page)

```
    my ( $self, $res, $logout ) = @_;

    # Your code here
}
```

samlBuildLogoutResponse

New in version 2.0.10.

This hook is triggered when LemonLDAP::NG is about to generate a SAML logout response

The hook's parameter is the Lasso::Logout object

Sample code:

```
use constant hook => {
   samlBuildLogoutResponse => 'buildLogoutResponse',
};

sub buildLogoutResponse {
    my ( $self, $res, $logout ) = @_;

    # Your code here
}
```

CAS Issuer hooks

casGotRequest

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG received an CAS authentication request on the */cas/login* endpoint.

The hook's parameter is a hash containing the CAS request parameters.

Sample code:

```
use constant hook => {
    casGotRequest                => 'filterService'
};

sub filterService {
    my ( $self, $req, $cas_request ) = @_;
    if ( $cas_request->{service} eq "http://auth.sp.com/" ) {
        return PE_OK;
    }
    else {
        return 999;
    }
}
```

casGenerateServiceTicket

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is about to generate a Service Ticket for a CAS application

The hook's parameters are:

- A hash of the parameters for the CAS request, which you can modify
- the configuration key of the cas application which will receive the ticket
- A hash of the session keys for the (internal) CAS session

Sample code:

```
use constant hook => {
    'casGenerateServiceTicket'    => 'changeRedirectUrl',
};

sub changeRedirectUrl {
    my ( $self, $req, $cas_request, $app, $Sinfos ) = @_;
    $cas_request->{service} .= "?hooked=1";
    return PE_OK;
}
```

casGenerateValidateResponse

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is about to send a CAS response to an application on the */cas/serviceValidate* endpoint.

The hook's parameters are:

- The username (CAS principal)
- A hash of modifiable attributes to be sent

Sample code:

```
use constant hook => {
    casGenerateValidateResponse    => 'addAttributes',
};

sub addAttributes {
    my ( $self, $req, $username, $attributes ) = @_;
    $attributes->{hooked} = 1;
    return PE_OK;
}
```

Password change hooks

passwordBeforeChange

New in version 2.0.12.

This hook is triggered when LemonLDAP::NG is about to change or reset a user's password. Returning an error will cancel the password change operation

The hook's parameters are:

- The main user identifier
- The new password
- The old password, if relevant

Sample code:

```
use constant hook => {
    passwordBeforeChange => 'blacklistPassword',
};

sub blacklistPassword {
    my ( $self, $req, $user, $password, $old ) = @_;
    if ( $password eq "12345" ) {
        $self->logger->error("I've got the same combination on my luggage");
        return PE_PP_INSUFFICIENT_PASSWORD_QUALITY;
    }
    return PE_OK;
}
```

passwordAfterChange

New in version 2.0.12.

This hook is triggered after LemonLDAP::NG has changed the user's password successfully in the underlying password database

The hook's parameters are:

- The main user identifier
- The new password
- The old password, if relevant

Sample code:

```
use constant hook => {
    passwordAfterChange  => 'logPasswordChange',
};

sub logPasswordChange {
    my ( $self, $req, $user, $password, $old ) = @_;
    $old ||= "";
    $self->userLogger->info("Password changed for $user: $old -> $password")
```

(continues on next page)

(continued from previous page)

```
    return PE_OK;
}
```

Routes

The plugin can also define new routes and call actions on them.

See also `Lemonldap::NG::Portal::Main::Plugin` man page.

19.4.2 Example

Plugin Perl module

Create for example the MyPlugin module:

```
vi /usr/share/perl5/Lemonldap/NG/Portal/MyPlugin.pm
```

Tip: If you do not want to mix files from the distribution with your own work, put your own code in `/usr/local/lib/site_perl/Lemonldap/NG/Portal/MyPlugin.pm`

```
package Lemonldap::NG::Portal::MyPlugin;

use Mouse;
use Lemonldap::NG::Portal::Main::Constants;
extends 'Lemonldap::NG::Portal::Main::Plugin';

use constant beforeAuth => 'verifyIP';

sub init {
          my ($self) = @_;
          $self->addUnauthRoute( mypath => 'hello', [ 'GET', 'PUT' ] );
          $self->addAuthRoute( mypath => 'welcome', [ 'GET', 'PUT' ] );
          return 1;
}
sub verifyIP {
          my ($self, $req) = @_;
          return PE_ERROR if($req->address !~ /^10/);
          return PE_OK;
}
sub hello {
          my ($self, $req) = @_;
          ...
          return $self->p->sendJSONresponse($req, { hello => 1 });
}
sub welcome {
          my ($self, $req) = @_;

          my $userid = $req->user;
          $self->p->logger->debug("Call welcome for $userid");
```

(continues on next page)

(continued from previous page)

```
        ...
        return $self->p->sendHtml($req, 'template', params => { WELCOME => 1 });
}
1;
```

Configuration

Declare the plugin in lemonldap-ng.ini:

```
vi /etc/lemonldap-ng/lemonldap-ng.ini
```

```
[portal]
customPlugins = Lemonldap::NG::Portal::MyPlugin
;customPlugins = Lemonldap::NG::Portal::MyPlugin1, Lemonldap::NG::Portal::MyPlugin2, ...
```

Since 2.0.7, it can also be configured in Manager, in General Parameters > Plugins > Custom Plugins.

CHAPTER

TWENTY

PRESENTATION

CHAPTER

TWENTYONE

INSTALLATION

21.1 Before installation

- *Prerequisites and dependencies*
- *Upgrade notes*

21.2 Installation

- *Installation from the tarball*
- *Installation on Debian/Ubuntu with packages*
- *Installation on RHEL/CentOS with packages*
- *Installation on Suse Linux Enterprise Server with packages*
- *Run in LemonLDAP::NG in Docker*
- *Node.js handler* NEW

21.3 After installation

- *Deploy Nginx configuration* *(recommended configuration)*
- *Deploy Apache configuration*
- *Deploy LemonLDAP::NG on Plack servers family* *(Twiggy, Starman, Corona,...)* NEW

CHAPTER

TWENTYTWO

CONFIGURATION

22.1 First steps

- *Configuration overview*
- *Configure Single Sign On cookie and portal URL*
- *Parameter redirections*
- *Set exported variables*
- *Manage virtual hosts*
- *Configure sessions specificities*

22.2 Portal

- *Presentation*
- *Portal customization*
- *Portal menu*
- *REST/SOAP servers*
- *Captcha*
- *Public pages*

22.2.1 Authentication, users and password databases

Official Backends	Authentication	Users	Password
Active Directory	✓	✓	✓
Apache (Basic, NTLM, OTP, . . .)	✓		
CAS	✓	NEW	
SQL Databases	✓	✓	✓
Demonstration	✓	✓	✓
Facebook	✓	✓	
GitHub NEW [1]	✓		
GPG NEW [2]	✓		
Kerberos NEW	✓		
LDAP	✓	✓	✓
LinkedIn	✓		
Null	✓	✓	✓
OpenID Connect	✓	✓	
PAM NEW	✓		
Proxy LL::NG	✓	✓	
Radius	✓		
REST NEW	✓	✓	✓
SAML 2.0 / Shibboleth	✓	✓	
Slave	✓	✓	
SSL	✓		
Twitter	✓		
WebID	✓	✓	
Yubikey DEPRECATED	*Replaced by Yubikey Second Factor*		
Custom modules NEW	✓	✓	✓

Combo Backends	Authentication	Users	Password
Choice by users	✓	✓	✓
Combination of auth schemes NEW	✓	✓	✓ (since *2.0.10*)
Multiple backends stack DEPRECATED	*Replaced by Combination*		

Obsolete Backends	Authentication	Users	Password
OpenID	✓	✓	
Remote LL::NG	✓	✓	

[1] *GitHub authentication* is available with LLNG 2.0.8
[2] *GPG authentication* is available with LLNG 2.0.2

Second factor (*documentation*)	Authentication
TOTP-or-U2F NEW	✓
U2F NEW	✓
TOTP (Google Authenticator,...) NEW	✓
E-mail Second Factor NEW	✓
External Second Factor (OTP, SMS,...) NEW	✓
Radius Second Factor NEW [3]	✓
REST Second Factor NEW	✓
Yubikey NEW	✓
Additional second factors NEW [4]	✓

Auth addons	Authentication
Auto Signin NEW	✓

22.2.2 Identity provider

Tip:

- All identity provider protocols can be used simultaneously
- LemonLDAP::NG can be used as a *proxy between those protocols*

Protocol	Service Provider	Identity Provider
CAS 1.0 / 2.0 / 3.0	✓	✓
SAML 2.0 / Shibboleth	✓	✓
OpenID 2.0 (obsolete)	✓	✓
OpenID Connect	✓	✓
Get parameters provider (for poor applications)		✓

Tip:

- Issuers timeout : Delay for issuers to submit their authentication requests
- To avoid a bad/expired token and lose redirection to the SP protected application after authentication if IdP URLs are served by different load balancers, you can force Issuer tokens to be stored into Global Storage by editing `lemonldap-ng.ini` in section [portal]:

```
[portal]
forceGlobalStorageIssuerOTT = 1
```

[3] *Radius second factor* is available with LLNG 2.0.6
[4] *Check DevOps file plugin* are available with LLNG 2.0.12

22.2.3 Attacks and Protection

Tip: To learn or find out more about security, go to *Security* documentation

Attack	LLNG protection	System Integrator protection
Brute Force	✓	✓
Page Content	✓	
CSRF	✓	
Deny of Service		✓
Invisible iFrame	✓	✓
Man-in-the-Middle		✓
Software Exploit		✓
SSO by-passing		✓
XSS	✓	

22.2.4 Plugins

Name	Description
Auto Signin NEW	Auto Signin Addon
Brute Force protection NEW	User must wait to log in after some failed login attempts
CDA	Cross Domain Authentication
Check DevOps[5] NEW	Check DevOps handler file plugin
Check state NEW	Check state plugin (test page)
Check user[6] NEW	Check access rights, transmitted headers and session attibutes for a specific user and URL
Configuration viewer NEW	Edit WebSSO configuration in Read Only mode
Context switching[7] NEW	Switch context other users
CrowdSec[8] NEW	CrowdSec bouncer
Custom	Write a custom plugin
Decrypt value[9] BETA	Decrypt ciphered values
Display login history	Display Success/Fails logins
Force Authentication	Force authentication to access to Portal
Global Logout[10]	Suggest to close all opened sessions at logout
Grant Sessions	Rules to apply before allowing a user to open a session
Impersonation[11] NEW	Allow users to use another identity
Find user[12] NEW	Search for user account
Notifications system	DIsplay a message during log in process
Portal Status	Experimental portal status page
Public pages	Enable public pages system
Refresh session API[13]	Plugin that provides an API to refresh a user session
Reset password by mail	Send a mail to reset its password
Reset certificate by mail[14] NEW	Allow users to reset their certificate
REST services NEW	REST server for *Proxy*
SOAP services DEPRECATED	SOAP server for *Proxy*
Stay connected NEW	Enable persistent connection on same browser
Upgrade session NEW	This plugin explains to an already authenticated user that a higher authentication level is required to access the URL instead of reject him

[5] *Additional second factors* are available with LLNG 2.0.6
[6] *Check user plugin* is available with LLNG 2.0.3
[7] *Context switching plugin* is available with LLNG 2.0.6
[8] *CrowdSec bouncer* is available with LLNG 2.0.12
[9] *Decrypt value plugin* is available with LLNG 2.0.7
[10] *Global Logout plugin* is available with LLNG 2.0.7
[11] *Impersonation plugin* is available with LLNG 2.0.3
[12] *Find user plugin* is available with LLNG 2.0.11
[13] *Refresh session API plugin* is available with LLNG 2.0.7
[14] *Reset certificate by mail plugin* is available with LLNG 2.0.7

22.3 Handlers

Handlers are software control agents to be installed on your web servers *(Nginx, Apache, PSGI like Plack based servers or Node.js)*.

Handler type	Apache	LLNG FastCGI/uWSGI server (Nginx, or *SSOaaS*)	Plack servers	Node.js (express apps or *SSOaaS*)	*Self protected apps*	Comment
Main *(default handler)*	✓	✓	✓	*Partial* **[15] **	✓	
AuthBasic	✓	✓	✓		✓	Designed for some server-to-server applications
CDA	✓	✓	✓		✓	For Cross Domain Authentication
DevOps (*SSOaaS*) NEW	✓	✓	✓	✓		Allows application developers to define their own rules and headers inside their applications
DevOpsST (*SSOaaS*) NEW	✓	✓	✓	✓		Enables both *DevOps* and *Service Token*
OAuth2[16] NEW	✓	✓	✓		✓	Uses OpenID Connect/OAuth2 access token to check authentication and authorization, can be used to protect Web Services
Secure Token	✓	✓	✓			Designed to secure exchanges between a LLNG reverse-proxy and a remote app
Service Token NEW *(Server-to-Server)*	✓	✓	✓	✓	✓	Designed to permit underlying requests *(API-Based Infrastructure)*
Zimbra PreAuth	✓	✓	✓			

[15] *Node.js handler* has not yet reached the same level of functionalities
[16] *OAuth2 Handler* is available with LLNG 2.0.4

22.4 LLNG databases

22.4.1 Configuration database

LL::NG needs a storage system to store its own configuration (managed by the manager). Choose one in the following list:

Backend	Share-able	Comment
File (JSON)		Not shareable between servers except if used in conjunction with *REST* or with a shared file system (NFS,...). Selected by default during installation.
YAML NEW		Same as *File* but in YAML format instead of JSON
SQL (RDBI/CDBI)	✓	**Recommended for large-scale systems**. Prefer CDBI.
LDAP	✓	
MongoDB	✓	
SOAP DEPRECATED	✓	Proxy backend to be used in conjunction with another configuration backend. **Can be used to secure another backend** for remote servers.
REST NEW	✓	Proxy backend to be used in conjunction with another configuration backend. **Can be used to secure another backend** for remote servers.
Local NEW		Use only lemonldap-ng.ini parameters.

Tip: You can not start with an empty configuration, so read *how to change configuration backend* to convert your existing configuration into another one.

22.4.2 Sessions database

Sessions are stored using Apache::Session modules family. All Apache::Session style modules are usable except for some features.

Attention: If you plan to use LLNG in a large-scale system, take a look at *Performance Test* to choose the right backend. A *Browseable SQL backend* is generally a good choice.

Back-end	Share-able	*Session explorer*	*Session restrictions*	Session expiration	Comment
File		✓	✓	✓	Not shareable between servers except if used in conjunction with *REST session backend* or with a shared file system (NFS,...). Selected by default during installation.
PgJSON	✓	✓	✓	✓	**Recommended backend for production installations**
Browseable MySQL	✓	✓	✓	✓	Recommended for those who prefer MySQL
Browseable LDAP	✓	✓	✓	✓	
Redis	✓	✓	✓	✓	The fastest. Must be secured by network access control.
MongoDB	✓	✓	✓	✓	Must be secured by network access control.
SQL	✓	✓	✓	✓	Unoptimized for *session explorer* and *single session* features.
REST NEW	✓	✓	✓	✓	Proxy backend to be used in conjunction with another session backend.
SOAP DEPRECATED	✓	✓	✓	✓	Proxy backend to be used in conjunction with another session backend.

Tip: You can migrate from one session backend to another using the *session conversion script*. (NEW *since 2.0.7*)

CHAPTER

TWENTYTHREE

APPLICATIONS PROTECTION

- *Writing rules and headers*
- *Variables that can be used in rules and headers*
- *Integrate vendor applications*
- *Integrate self-made applications*
- *Form replay*
- *Custom Handlers*
- *WebServices / API*

23.1 Well known compatible applications

Note: Here is a list of well known applications that are compatible with LL::NG. A full list is available on *vendor applications page*.


GLPI


LIFERAY
Enterprise. Open Source. For Life.


MediaWiki


Nextcloud


simpleSAMLphp


XWIKI
Zimbra

CHAPTER

TWENTYFOUR

ADVANCED FEATURES

- *SMTP server setup*
- *Notifications system*
- *Store password in session*
- *Cross Domain Authentication (CDA)*
- *Role Based Access Control (RBAC)*
- *Use custom functions*
- *Use extended functions*
- *Reset password by mail (self service)*
- *Create an account (self service)*
- *Forward logout to applications*
- *Secure Token Handler*
- *AuthBasic Handler*
- *SSO as a Service (SSOaaS)* NEW
- *Handling server webservice calls* NEW
- LemonLDAP::NG kubernetes controller
- *Safe jail*
- *Login history*
- *Fast CGI support*
- *Advanced PSGI usage*
- *Ignore some manager tests*
- *See full parameters list*

CHAPTER

TWENTYFIVE

MINI HOWTOS

- *Command Line Interface (lemonldap-ng-cli) examples*
- *Modify Manager protection*
- *Configuration and sessions in MySQL*
- *Configuration and sessions in LDAP*
- *Configuration and sessions access by REST*
- *Integration in Active Directory (LDAP and Kerberos)*
- *Create a protocol proxy (SAML to OpenID, CAS to SAML ,. . .)*
- *Convert HTTP header into environment variable*
- *Connect to Renater Federation* NEW
- *Run LemonLDAP::NG components behind a reverse proxy*
- *Configure LL::NG to use an outgoing proxy*

CHAPTER

TWENTYSIX

EXPLOITATION

- *Performances*
- *Security*
- *SELinux*
- *Handler status page*
- *Portal state check* *(health check for fail-over)* NEW
- *Monitoring*
- *Logs settings*
- *Error messages*
- *High Availability*

CHAPTER

TWENTYSEVEN

BUG REPORT

See *How to report a bug*.

CHAPTER

TWENTYEIGHT

DEVELOPER CORNER

To contribute, see :

- *Contribute to project*

To develop an handler, see:

- *Handler architecture*
- *Custom handlers*

To develop a portal plugin, see manpages:

- Lemonldap::NG::Portal
- Lemonldap::NG::Portal::Auth
- Lemonldap::NG::Portal::UserDB
- Lemonldap::NG::Portal::Main::SecondFactor
- Lemonldap::NG::Portal::Main::Issuer
- Lemonldap::NG::Portal::Main::Plugin
- Lemonldap::NG::Portal::Main::Request *(the request object)*

To add a new language:

- Join us on https://www.transifex.com/lemonldapng/lemonldapng/dashboard/
- translate the 3 files
- then we will append them in sources.

If you don't want to publish your translation (`XX` must be replaced by your language code):

- Manager: translate `lemonldap-ng-manager/site/htdocs/static/languages/en.json` in `lemonldap-ng-manager/site/htdocs/static/languages/XX.json` and enable it in "lemonldap-ng.ini" file
- Portal: translate `lemonldap-ng-portal/site/htdocs/static/languages/en.json` in `lemonldap-ng-portal/site/htdocs/static/languages/XX.json` and enable it in "lemonldap-ng.ini" file
- Portal Mails: translate `lemonldap-ng-portal/site/templates/common/mail/en.json` in `lemonldap-ng-portal/site/templates/common/mail/XX.json`