LdapCherry - Directory Management Interface

Release 0.1.0

August 10, 2015

Contents

1 Install **1**
- 1.1 From the sources . . . 1
- 1.2 From Pypi . . . 1
- 1.3 Installed files . . . 1

2 Deploy **3**
- 2.1 Launch . . . 3
- 2.2 Roles and Attributes Configuration . . . 3
- 2.3 Main Configuration . . . 7

3 Backends **11**
- 3.1 Backend id prefix . . . 11
- 3.2 Common backend parameters . . . 11
- 3.3 Ldap Backend . . . 11
- 3.4 Active Directory Backend . . . 13
- 3.5 Demo Backend . . . 14

4 Full Configuration **17**
- 4.1 Main ini configuration file . . . 17
- 4.2 Yaml Attributes configuration file . . . 20
- 4.3 Yaml Roles configuration file . . . 22

5 Implementing cutom backends **25**
- 5.1 API . . . 25
- 5.2 Configuration . . . 26
- 5.3 Exceptions . . . 27
- 5.4 Example . . . 27

6 Implementing password policy modules **31**
- 6.1 API . . . 31
- 6.2 Configuration . . . 31
- 6.3 Example . . . 32

7 Changelog **33**
- 7.1 Dev . . . 33
- 7.2 Version 0.1.0 . . . 33
- 7.3 Version 0.0.1 . . . 33

8 Some Goodies **35**

8.1 Init Script 35
8.2 Apache Vhost 37
8.3 Nginx Vhost 37
8.4 Lighttpd Vhost 37

9 Screenshots **39**

10 LdapCherry **41**

11 Presentation **43**

12 Screenshots **45**

13 License **47**

14 Discussion / Help / Updates **49**

Python Module Index **51**

CHAPTER 1

Install

1.1 From the sources

Download the latest release from GitHub.

```
$ tar -xf ldapcherry*.tar.gz
$ cd ldapcherry*
$ python setup.py install
```

1.2 From Pypi

```
$ pip install ldapcherry
```

or

```
$ easy_install ldapcherry
```

1.3 Installed files

ldapCherry install directories are:

- **/etc/ldapcherry/** (configuration)
- **dist-package** or **site-packages** of your distribution (LdapCherry modules)
- **/usr/share/ldapcherry/** (static content (css, js, images...) and templates)

These directories can be changed by exporting the following variables before launching the install command:

```
#optional, default sys.prefix + 'share' (/usr/share/ on most Linux)
$ export DATAROOTDIR=/usr/local/share/
#optional, default /etc/
$ export SYSCONFDIR=/usr/local/etc/
```

CHAPTER 2

Deploy

LdapCherry aims to be as simple as possible to deploy. The Application is constituted of:

- ldapcherryd: the daemon to lauch LdapCherry.
- one ini file (ldapcherry.ini by default): the entry point for the configuration, containing all the "technical" attributes.
- two yaml files (roles.yml and attributes by default): the files containing the roles and attributes definition.

The default configuration directory is **/etc/ldapcherry/**.

2.1 Launch

LdapCherry is launched using the internal cherrypy server:

```
# ldapcherryd help
$ ldapcherryd -h

# launching ldapcherryd in the forground
$ ldapcherryd -c /etc/ldapcherry/ldapcherry.ini

# launching ldapcherryd as a daemon
$ ldapcherryd -c /etc/ldapcherry/ldapcherry.ini -p /var/run/ldapcherry/ldapcherry.pid -d
```

2.2 Roles and Attributes Configuration

2.2.1 Entry point in main configuration

The main configuration file (**ldapcherry.ini** by default) contains two parameters locating the roles and attributes configuration files:

Parameter	Section	Description	Values
attributes.file	attributes	Attributes configuration file	Path to conf file
roles.file	roles	Roles configuration file	Path to conf file

2.2.2 Attributes Configuration

The attributes configuration is done in a yaml file (**attributes.yml** by default).

Mandatory parameters

The mandatory parameters for an attribute, and their format are the following:

```
<attr id>:
    description: <Human readable description of the attribute>                    # (free text)
    display_name: <Display name in LdapCherry forms>                               # (free text)
    weight: <weight controlling the display order of the attributes, lower is first> # (integer)
    type: <type of the attributes>                                                 # (in ['int', 's
    backends:                                                                      # (list of backe
        - <backend id 1>: <backend 1 attribute name>
        - <backend id 2>: <backend 2 attribute name>
```

Warning: <attr id> (the attribute id) must be unique, LdapCherry won't start if it's not.

Warning: <backend id> (the backend id) must be defined in main ini configuration file. LdapCherry won't start if it's not.

Type stringlist values

If **type** is set to **stringlist** the parameter **values** must be filled with the list of possible values:

```
<attr id>:
    description: <Human readable description of the attribute>
    display_name: <Display name in LdapCherry forms>
    weight: <weight controlling the display order of the attributes)

    type: stringlist
    values:
        - value1
        - value2
        - value3

    backends:
        - <backend id>: <backend attribute name>
```

Key attribute:

One attribute must be used as a unique key across all backends:

To set the key attribute, you must set **key** to **True** on this attribute.

Example:

```
uid:
    description: "UID of the user"
    display_name: "UID"
    search_displayed: True
    key: True                       # defining the attribute as "key"
    type: string
    weight: 50
    backends:
        ldap: uid
        ad: sAMAccountName
```

Authorize self modification

A user can modify some of his attributes (self modification). In such case, the parameter **self** must set to **True**:

```
<attr id>:
    description: <Human readable description of the attribute>
    display_name: <Display name in LdapCherry forms>
    weight: <weight controlling the display order of the attributes)
    type: <type of the attributes>

    self: True

    backends:
        - <backend id 1>: <backend 1 attribute name>
        - <backend id 2>: <backend 2 attribute name>
```

Autofill

LdapCherry has the possibility to auto-fill fields from other fields, to use this functionnality **autofill** must be set.

Example:

```
gidNumber:
    description: "Group ID Number of the user"
    display_name: "GID Number"
    weight: 70
    type: int

    autofill:
        function: lcUidNumber # name of the function to call
        args:                 # list of arguments
            - $first-name     #
            - $name
            - '10000'
            - '40000'

    backends:
        ldap: gidNumber
```

Arguments of the **autofill** function work as follow:

- if argument starts with **$**, for example **$my_field**, the value of form input **my_field** will be passed to the function.
- otherwise, it will be treated as a fixed argument.

Available **autofill** functions:

- lcUid: generate 8 characters ascii uid from 2 other fields (first letter of the first field, 7 first letters of the second):

```
autofill:
    function: lcUid
    args:
        - $first-name
        - $name
```

- lcDisplayName: concatenate two fields (with a space as separator):

```
autofill:
    function: lcDisplayName
    args:
```

```
        - $first-name
        - $name
```

- lcMail: generate an email address from 2 other fields and a domain (<uid>+domain):

```
autofill:
    function: lcMail
    args:
        - $first-name
        - $name
        - '@example.com'
```

- lcUidNumber: generate an uid number from 2 other fields and between a minimum and maximum value:

```
autofill:
    function: lcUidNumber
    args:
        - $first-name
        - $name
        - '10000'
        - '40000'
```

- lcHomeDir: generate an home directory from 2 other fields and a root (<root>+<uid>):

```
autofill:
    function: lcHomeDir
    args:
        - $first-name
        - $name
        - /home/
```

2.2.3 Roles Configuration

The roles configuration is done in a yaml file (**roles.yml** by default).

Mandatory parameters

Roles are seen as an aggregate of groups:

```
<role id>:
    display_name: <role display name in LdapCherry>
    description: <human readable role description>
    backends_groups:                                # list of backends
        <backend id 1>:                             # list of groups in backend
            - <b1 group 1>
            - <b1 group 2>
        <backend id 2>:
            - <b2 group 1>
            - <b2 group 2>
```

Warning: <role id> must be unique, LdapCherry won't start if it's not

Defining LdapCherry Administrator role

At least one of the declared roles must be tagged to be LdapCherry administrators.

Doing so is done by setting **LC_admins** to **True** for the selected role:

```
<role id>:
    display_name: <Role display name in LdapCherry>
    description: <human readable role description>

    LC_admins: True

    backends_groups:                                # list of backends
        <backend id 1>:                             # list of groups in backend
            - <b1 group 1>
            - <b1 group 2>
        <backend id 2>:
            - <b2 group 1>
            - <b2 group 2>
```

Nesting roles

LdapCherry handles roles nesting:

```
parent_role:
    display_name: Role parent
    description: The parent role
    backends_groups:
        backend_id_1:
            - b1_group_1
            - b1_group_2
        backend_id_2:
            - b2_group_1
            - b2_group_2
    subroles:
        child_role_1:
            display_name: Child role 1
            description: The first Child Role
            backends_groups:
                backend_id_1:
                    - b1_group_3
        child_role_2:
            display_name: Child role 2
            description: The second Child Role
            backends_groups:
                backend_id_1:
                    - b1_group_4
```

In that case, child_role_1 and child_role_2 will contain all groups of parent_role plus their own specific groups.

2.3 Main Configuration

2.3.1 Webserver

LdapCherry uses the embedded http server of CherryPy, however it has some limitations:

- no listening on port 80/443 (unless run as root, which is strongly discourage)
- no https

The simpler way to properly deploy LdapCherry is to run it listening only on localhost with a port above 1024 and put it behind an http server like nginx, apache or lighttpd acting as a reverse http(s) proxy.

Parameter	Section	Description	Values	Comment
server.socket_host	global	Listening IP	IP on which to listen	Use '0.0.0.0' to listen on any interfaces.
server.socket_port	global	Listening Port	TCP Port	
server.thread_pool	global	Number of threads created by the CherryPy server	Number of threads threads	
tools.staticdir.on	/static	Serve static files through LdapCherry	True, False	These files could be server directly by an HTTP server for better performance.
tools.staticdir.dir	/static	Directory containing LdapCherry static resources (js, css, img...)	Path to static resources	

example:

```
[global]

# listing interface
server.socket_host = '127.0.0.1'
# port
server.socket_port = 8080
# number of threads
server.thread_pool = 8

# enable cherrypy static handling
# to comment if static content are handled otherwise
[/static]
tools.staticdir.on = True
tools.staticdir.dir = '/usr/share/ldapcherry/static/'
```

2.3.2 Backends

Backends are configure in the **backends** section, the format is the following:

```
[backends]

# backend python module path
<backend id>.module = <python.module.path>

# display name of the backend in forms
<backend id>.display_name = <display name of the backend>

# parameters of the module instance for backend <backend id>.
<backend id>.<param> = <value>
```

It's possible to instanciate the same module several times.

2.3.3 Authentication and sessions

LdapCherry supports several authentication modes:

Parameter	Section	Description	Values	Comment
auth.mode	auth	Authentication mode	• 'and' (user must auth on all backends) • 'or' (user must auth on one of the backends) • 'none' (disable auth) • 'custom' (use custom auth module)	
auth.module	auth	Custom auth module	python class path to module	only used if auth.mode='custom'
tools.sessions.timeout	global	Session timeout in minutes	Number of minutes	

Different session backends can also be configured (see CherryPy documentation for details)

```
[global]
# session configuration
# activate session
tools.sessions.on = True
# session timeout in minutes
tools.sessions.timeout = 10
# file session storage(to use if multiple processes,
# default is in RAM and per process)
#tools.sessions.storage_type = "file"
# session
#tools.sessions.storage_path = "/var/lib/ldapcherry/sessions"

[auth]
# Auth mode
# * and: user must authenticate on all backends
# * or:  user must authenticate on one of the backend
# * none: disable authentification
# * custom: custom authentification module (need auth.module param)
auth.mode = 'or'

# custom auth module to load
#auth.module = 'ldapcherry.auth.modNone'
```

2.3.4 Logging

LdapCherry has two loggers, one for errors and applicative actions (login, del/add, logout...) and one for access logs.

Each logger can be configured to log to syslog, file or be disabled.

Logging parameters:

Parameter	Section	Description	Values	Comment
log.access_handler	global	Logger type for access log	'syslog', 'file', 'none'	
log.error_handler	global	Logger type for applicative log	'syslog', 'file', 'none'	
log.access_file	global	log file for access log	path to log file	only used if log.access_handler='file'
log.error_file	global	log file for applicative log	path to log file	only used if log.error_handler='file'
log.level	global	log level of LdapCherry	'debug', 'info', 'warning', 'error', 'critical'	

Example:

```
[global]

# logger syslog for access log
log.access_handler = 'syslog'
# logger syslog for error and ldapcherry log
log.error_handler = 'syslog'
# log level
log.level = 'info'
```

2.3.5 Custom javascript

It's possible to add custom javascript to LdapCherry, mainly to add custom autofill functions.

Configuration:

Parameter	Section	Description	Values	Comment
tools.staticdir.on	/custom	Serve custom js files through LdapCherry	True, False	These files could be server directly by an HTTP server for better performance.
tools.staticdir.dir	/custom	Directory containing custom js files	Path to static resources	• custom js files must be put at the root if the directory • only files ending with ".js" are taken into account

2.3.6 Other LdapCherry parameters

Parameter	Section	Description	Values
template_dir	resources	LdapCherry template directory	path to template dir

```
# resources parameters
[resources]
# templates directory
template_dir = '/usr/share/ldapcherry/templates/'
```

CHAPTER 3

Backends

3.1 Backend id prefix

Each parameter of a backend instance must be prefixed by a backend id. This backend id must be unique.

For example:

```
[backends]

# configuration of the bk1 backend
bk1.module = 'my.backend.module'
bk1.display_name = 'My backend module'
bk1.param = 'value'
```

Warning: For the rest of the backends documentation, this prefix is infered.

3.2 Common backend parameters

Every backend instance systematicaly has two parameters:

Parameter	Section	Description	Values	Comment
module	backends	Library path to the module	Python library path	
display_name	backends	Display_name of the backend	Free text	

3.3 Ldap Backend

3.3.1 Class path

The class path for the ldap backend is **ldapcherry.backend.backendLdap**.

3.3.2 Configuration

The ldap backend exposes the following parameters:

Parameter	Section	Description	Values	Comment
uri	backends	The ldap uri to access	ldap uri	• use ldap:// for clear/starttls • use ldaps:// for ssl • custom port: ldap://<host>:<port>
ca	backends	Path to the CA file	file path	optional
starttls	backends	Use starttls	'on' or 'off'	optional
checkcert	backends	Check the server certificat	'on' or 'off'	optional
binddn	backends	The bind dn to use	ldap dn	This dn must have read/write permissions
password	backends	The password of the bind dn	password	
timeout	backends	Ldap connexion timeout	integer (second)	
password	backends	The password of the bind dn	password	
groupdn	backends	The ldap dn where groups are	ldap dn	
userdn	backends	The ldap dn where users are	ldap dn	
user_filter_tmpl	backends	The search filter template to recover a given user	ldap search filter template	The user identifier is passed through the **username** variable (*%(username)s*).
group_filter_tmpl	backends	The search filter template to recover the groups of a given user	ldap search filter template	The following variables are usable: * **username**: the user key attribute * **userdn**: the user ldap dn
group_attr.<member attr>	backends	Member attribute template value	template	• <member attr> is the member attribute in groups dn entries • every user attributes are exposed in the template • multiple attributes can be set
objectclasses	backends	list of object classes for users	comma separated list	
dn_user_attr	backends	attribute used in users dn	dn attribute	

3.3.3 Example

```
[backends]

# name of the module
ldap.module = 'ldapcherry.backend.backendLdap'
# display name of the ldap
ldap.display_name = 'My Ldap Directory'

# uri of the ldap directory
ldap.uri = 'ldap://ldap.ldapcherry.org'
# ca to use for ssl/tls connexion
#ldap.ca = '/etc/dnscherry/TEST-cacert.pem'
# use start tls
#ldap.starttls = 'off'
# check server certificate (for tls)
#ldap.checkcert = 'off'
# bind dn to the ldap
ldap.binddn = 'cn=dnscherry,dc=example,dc=org'
# password of the bind dn
ldap.password = 'password'
# timeout of ldap connexion (in second)
ldap.timeout = 1

# groups dn
ldap.groupdn = 'ou=group,dc=example,dc=org'
# users dn
ldap.userdn = 'ou=people,dc=example,dc=org'
# ldapsearch filter to get a user
ldap.user_filter_tmpl = '(uid=%(username)s)'
# ldapsearch filter to get groups of a user
ldap.group_filter_tmpl = '(member=uid=%(username)s,ou=People,dc=example,dc=org)'
# filter to search users
ldap.search_filter_tmpl = '(|(uid=%(searchstring)s*)(sn=%(searchstring)s*))'

# ldap group attributes and how to fill them
ldap.group_attr.member = "%(dn)s"
#ldap.group_attr.memberUid = "%(uid)s"
# object classes of a user entry
ldap.objectclasses = 'top, person, posixAccount, inetOrgPerson'
# dn entry attribute for an ldap user
ldap.dn_user_attr = 'uid'
```

3.4 Active Directory Backend

3.4.1 Class path

The class path for the ldap backend is **ldapcherry.backend.backendAD**.

3.4.2 Configuration

Parameter	Section	Description	Values	Comment
uri	backends	The ldap uri to access	ldap uri	• use ldap:// for clear/starttls • use ldaps:// for ssl • custom port: ldap://<host>:<port>
ca	backends	Path to the CA file	file path	optional
starttls	backends	Use starttls	'on' or 'off'	optional
checkcert	backends	Check the server certificat	'on' or 'off'	optional
domain	backends	Name of the domain	AD domain	
login	backends	login used for connecting to AD	login	user used must have sufficient rights
password	backends	password if binding user	password	

3.4.3 Example

```
[backends]

# Name of the backend
ad.module = 'ldapcherry.backend.backendAD'
# display name of the ldap
ad.display_name = 'My Active Directory'
# ad domain
ad.domain = 'dc.ldapcherry.org'
# ad login
ad.login  = 'administrator'
# ad password
ad.password = 'qwertyP455'
# ad uri
ad.uri = 'ldap://ad.ldapcherry.org'

## ca to use for ssl/tls connexion
#ad.ca = '/etc/dnscherry/TEST-cacert.pem'
## use start tls
#ad.starttls = 'off'
## check server certificate (for tls)
#ad.checkcert = 'off'
```

3.5 Demo Backend

Warning: This backend is only meant for demo.

3.5.1 Class path

The class path for the ldap backend is **ldapcherry.backend.backendDemo**.

3.5.2 Configuration

Parameter	Section	Description	Values	Comment
admin.user	backends	Login for default admin	string	optional, default: 'admin'
admin.password	backends	Password for default admin	string	optional, default: 'admin'
admin.groups	backends	Groups for default admin	comma separated list	
basic.user	backends	Login for default user	string	optional, default: 'user'
basic.password	backends	Password for default user	string	optional, default: 'user'
basic.groups	backends	Groups for default user	comma separated list	
pwd_attr	backends	Password attribute name	string	
search_attributes	backends	Attributes used for search	comma separated list	

3.5.3 Example

```
[backends]

# path to the module
demo.module = 'ldapcherry.backend.backendDemo'
# display name of the module
demo.display_name  = 'Demo Backend'

## admin user login (optional, default: 'admin')
#demo.admin.user = 'admin'
## admin user password (optional: default 'admin')
#demo.admin.password = 'admin'
# groups for the default admin user (comma separated)
demo.admin.groups  = 'DnsAdmins'

## basic user login (optional, default: 'user')
#demo.basic.user = 'user'
## admin user password (optional: default 'user')
#demo.basic.password = 'user'
# groups for the default basic user (comma separated)
demo.basic.groups  = 'Test 2, Test 1'

# password attribute used for auth
demo.pwd_attr = 'userPassword'
# attributes to search on
demo.search_attributes = 'cn, sn, givenName, uid'
```

CHAPTER 4

Full Configuration

4.1 Main ini configuration file

```
# global parameters
[global]

# listing interface
server.socket_host = '127.0.0.1'
# port
server.socket_port = 8080
# number of threads
server.thread_pool = 8
#don't show traceback on error
request.show_tracebacks = False

# log configuration
# /!\ you can't have multiple log handlers
#####################################
#   configuration to log in files   #
#####################################
## logger 'file' for access log
#log.access_handler = 'file'
## logger syslog for error and ldapcherry log
#log.error_handler = 'file'
## access log file
#log.access_file = '/tmp/ldapcherry_access.log'
## error and ldapcherry log file
#log.error_file = '/tmp/ldapcherry_error.log'

#####################################
#  configuration to log in syslog   #
#####################################
# logger syslog for access log
#log.access_handler = 'syslog'
## logger syslog for error and ldapcherry log
log.error_handler = 'syslog'

#####################################
#  configuration to not log at all  #
#####################################
# logger none for access log
log.access_handler = 'none'
```

```
# logger none for error and ldapcherry log
#log.error_handler = 'none'

# log level
log.level = 'info'

# session configuration
# activate session
tools.sessions.on = True
# session timeout
tools.sessions.timeout = 10
# file session storage(to use if multiple processes,
# default is in RAM and per process)
#tools.sessions.storage_type = "file"
# session
#tools.sessions.storage_path = "/var/lib/ldapcherry/sessions"

[attributes]

# file discribing form content
attributes.file = '/etc/ldapcherry/attributes.yml'

[roles]

# file listing roles
roles.file = '/etc/ldapcherry/roles.yml'

[backends]

#####################################
#   configuration of ldap backend   #
#####################################

# name of the module
ldap.module = 'ldapcherry.backend.backendLdap'
# display name of the ldap
ldap.display_name = 'My Ldap Directory'

# uri of the ldap directory
ldap.uri = 'ldap://ldap.ldapcherry.org'
# ca to use for ssl/tls connexion
#ldap.ca = '/etc/dnscherry/TEST-cacert.pem'
# use start tls
#ldap.starttls = 'off'
# check server certificate (for tls)
#ldap.checkcert = 'off'
# bind dn to the ldap
ldap.binddn = 'cn=dnscherry,dc=example,dc=org'
# password of the bind dn
ldap.password = 'password'
# timeout of ldap connexion (in second)
ldap.timeout = 1

# groups dn
ldap.groupdn = 'ou=group,dc=example,dc=org'
# users dn
ldap.userdn = 'ou=people,dc=example,dc=org'
# ldapsearch filter to get a user
```

```
ldap.user_filter_tmpl = '(uid=%(username)s)'
# ldapsearch filter to get groups of a user
ldap.group_filter_tmpl = '(member=uid=%(username)s,ou=People,dc=example,dc=org)'
# filter to search users
ldap.search_filter_tmpl = '(|(uid=%(searchstring)s*)(sn=%(searchstring)s*))'

# ldap group attributes and how to fill them
ldap.group_attr.member = "%(dn)s"
#ldap.group_attr.memberUid = "%(uid)s"
# object classes of a user entry
ldap.objectclasses = 'top, person, posixAccount, inetOrgPerson'
# dn entry attribute for an ldap user
ldap.dn_user_attr = 'uid'

#####################################
#   configuration of ad backend     #
#####################################

## Name of the backend
#ad.module = 'ldapcherry.backend.backendAD'
## display name of the ldap
#ldap.display_name = 'My Active Directory'
## ad domain
#ad.domain = 'dc.ldapcherry.org'
## ad login
#ad.login  = 'administrator'
## ad password
#ad.password = 'qwertyP455'
## ad uri
#ad.uri = 'ldap://ldap.ldapcherry.org'

## ca to use for ssl/tls connexion
#ad.ca = '/etc/dnscherry/TEST-cacert.pem'
## use start tls
#ad.starttls = 'off'
## check server certificate (for tls)
#ad.checkcert = 'off'

[ppolicy]

# password policy module
ppolicy.module = 'ldapcherry.ppolicy.simple'

# parameters of the module
min_length = 8
min_upper = 1
min_digit = 1

# authentification parameters
[auth]

# Auth mode
# * and: user must authenticate on all backends
# * or:  user must authenticate on one of the backend
# * none: disable authentification
# * custom: custom authentification module (need auth.module param)
auth.mode = 'or'
```

```
# custom auth module to load
#auth.module = 'ldapcherry.auth.modNone'

# resources parameters
[resources]
# templates directory
templates.dir = '/usr/share/ldapcherry/templates/'

[/static]
# enable serving static file through ldapcherry
# set to False if files served directly by an
# http server for better performance
tools.staticdir.on = True
# static resources directory (js, css, images...)
tools.staticdir.dir = '/usr/share/ldapcherry/static/'

## custom javascript files
#[/custom]
#
## enable serving static file through ldapcherry
## set to False if files served directly by an
## http server for better performance
#tools.staticdir.on = True

## path to directory containing js files
## use it to add custom auto-fill functions
#tools.staticdir.dir = '/etc/ldapcherry/custom_js/'
```

4.2 Yaml Attributes configuration file

```
cn:
    description: "First Name and Display Name"
    display_name: "Display Name"
    type: string
    weight: 30
    autofill:
        function: lcDisplayName
        args:
            - $first-name
            - $name
    backends:
        ldap: cn
#        ad: CN
first-name:
    description: "First name of the user"
    display_name: "First Name"
    search_displayed: True
    type: string
    weight: 20
    backends:
        ldap: givenName
#        ad: givenName
name:
    description: "Family name of the user"
    display_name: "Name"
    search_displayed: True
```

```
    weight: 10
    type: string
    backends:
        ldap: sn
#        ad: sn
email:
    description: "Email of the user"
    display_name: "Email"
    search_displayed: True
    type: email
    weight: 40
    autofill:
        function: lcMail
        args:
            - $first-name
            - $name
            - '@example.com'
    backends:
        ldap: mail
#        ad: mail
uid:
    description: "UID of the user"
    display_name: "UID"
    search_displayed: True
    key: True
    type: string
    weight: 50
    autofill:
        function: lcUid
        args:
            - $first-name
            - $name
    backends:
        ldap: uid
#        ad: sAMAccountName
uidNumber:
    description: "User ID Number of the user"
    display_name: "UID Number"
    weight: 60
    type: int
    autofill:
        function: lcUidNumber
        args:
            - $first-name
            - $name
    backends:
        ldap: uidNumber
#        ad: UIDNumber
gidNumber:
    description: "Group ID Number of the user"
    display_name: "GID Number"
    weight: 70
    type: int
    default: 10000
    backends:
        ldap: gidNumber
#        ad: GIDNumber
shell:
```

```
    description: "Shell of the user"
    display_name: "Shell"
    weight: 80
    self: True
    type: stringlist
    values:
        - /bin/bash
        - /bin/zsh
        - /bin/sh
    backends:
        ldap: loginShell
#        ad: LOGINSHEL
home:
    description: "Home user path"
    display_name: "Home"
    weight: 90
    type: string
    autofill:
        function: lcHomeDir
        args:
            - $first-name
            - $name
            - /home/
    backends:
        ldap: homeDirectory
#        ad: HOMEDIRECTORY
password:
    description: "Password of the user"
    display_name: "Password"
    weight: 31
    self: True
    type: password
    backends:
        ldap: userPassword
#        ad: PASSWORD

#logscript:
#    description: "Windows login script"
#    display_name: "Login script"
#    weight: 100
#    type: fix
#    value: login1.bat
#    backends:
#        ad: scriptPath
```

4.3 Yaml Roles configuration file

```
admin-lv3:
    display_name: Administrators Level 3
    description: Super administrators of the system
    backends_groups:
        ldap:
            - cn=dns admins,ou=Group,dc=example,dc=org
            - cn=nagios admins,ou=Group,dc=example,dc=org
            - cn=puppet admins,ou=Group,dc=example,dc=org
            - cn=users,ou=Group,dc=example,dc=org
```

```
#        ad:
#            - Administrators
#            - Domain Controllers
#            - Domain Users

admin-lv2:
    display_name: Administrators Level 2
    description: Basic administrators of the system
    LC_admins: True
    backends_groups:
        ldap:
            - cn=nagios admins,ou=Group,dc=example,dc=org
            - cn=users,ou=Group,dc=example,dc=org
#        ad:
#            - Domain Users
#            - Domain Controllers

developpers:
    display_name: Developpers
    description: Developpers of the system
    backends_groups:
        ldap:
            - cn=developpers,ou=Group,dc=example,dc=org
            - cn=users,ou=Group,dc=example,dc=org

users:
    display_name: Simple Users
    description: Basic users of the system
    backends_groups:
        ldap:
            - cn=users,ou=Group,dc=example,dc=org
#        ad:
#            - Domain Users
```

CHAPTER 5

Implementing cutom backends

5.1 API

The backend modules must respect the following API:

class `ldapcherry.backend.`**`Backend`**(*config*, *logger*, *name*, *attrslist*, *key*)

Bases: `object`

`__init__`(*config*, *logger*, *name*, *attrslist*, *key*)

Initialize the backend

Parameters

- **`config`** (*dict {'config key': 'value'}*) – the configuration of the backend
- **`logger`** (*python logger*) – the cherrypy error logger object
- **`name`** (*string*) – id of the backend
- **`attrslist`** (*list of strings*) – list of the backend attributes
- **`key`** (*string*) – the key attribute

`add_to_groups`(*username*, *groups*)

Add a user to a list of groups

Parameters

- **`username`** (*string*) – 'key' attribute of the user
- **`groups`** (*list of strings*) – list of groups

`add_user`(*attrs*)

Add a user to the backend

Parameters **`attrs`** (*dict ({<attr>: <value>})*) – attributes of the user

> **Warning:** raise UserAlreadyExists if user already exists

`auth`(*username*, *password*)

Check authentication against the backend

Parameters

- **`username`** (*string*) – 'key' attribute of the user
- **`password`** (*string*) – password of the user

Return type boolean (True is authentication success, False otherwise)

del_from_groups (*username*, *groups*)
Delete a user from a list of groups

Parameters

- **username** (*string*) – 'key' attribute of the user
- **groups** (*list of strings*) – list of groups

> **Warning:** raise GroupDoesntExist if group doesn't exist

del_user (*username*)
Delete a user from the backend

Parameters **username** (*string*) – 'key' attribute of the user

get_groups (*username*)
Get a user's groups

Parameters **username** (*string*) – 'key' attribute of the user

Return type list of groups

get_user (*username*)
Get a user's attributes

Parameters **username** (*string*) – 'key' attribute of the user

Return type dict ({<attr>: <value>})

> **Warning:** raise UserDoesntExist if user doesn't exist

search (*searchstring*)
Search backend for users

Parameters **searchstring** (*string*) – the search string

Return type dict of dict ({<user attr key>: {<attr>: <value>}})

set_attrs (*username*, *attrs*)
Set a list of attributes for a given user

Parameters

- **username** (*string*) – 'key' attribute of the user
- **attrs** (*dict ({<attr>: <value>})*) – attributes of the user

5.2 Configuration

Configuration for your backend is declared in the main ini file, inside [backends] section:

For example with the configuration:

```
[backends]

# class path to module
b_id.module = "my.backend.module"
```

```
b_id.param1 = "my value 1"
b_id.param2 = "my value 2"
```

Note: One module can be instanciated several times, the prefix b_id permits to differenciate instances and their specific configuration.

The following hash will be passed as configuration to the module constructor as parameter config:

```
{
    'param1': "my value 1",
    'param2': "my value 2",
}
```

After having set **self.config** to **config** in the constructor, parameters can be recovered by **self.get_param**:

class `ldapcherry.backend.`**`Backend`**(*config*, *logger*, *name*, *attrslist*, *key*)

Bases: `object`

`get_param`(*param*, *default=None*)

Get a parameter in config (handle default value)

Parameters

- **`param`** (*string*) – name of the parameter to recover
- **`default`** (*string or None*) – the default value, raises an exception if param is not in configuration and default is None (which is the default value).

Return type the value of the parameter or the default value if not set in configuration

5.3 Exceptions

The following exception can be used in your module

exception `ldapcherry.exceptions.`**`UserDoesntExist`**(*user*, *backend*)

Bases: `exceptions.Exception`

exception `ldapcherry.exceptions.`**`UserAlreadyExists`**(*user*, *backend*)

Bases: `exceptions.Exception`

exception `ldapcherry.exceptions.`**`GroupDoesntExist`**(*group*, *backend*)

Bases: `exceptions.Exception`

These exceptions permit a nicer error handling and avoid a generic message to be thrown at the user.

5.4 Example

Here is the ldap backend module that comes with LdapCherry:

```
# -*- coding: utf-8 -*-
# vim:set expandtab tabstop=4 shiftwidth=4:
#
# The MIT License (MIT)
# LdapCherry
# Copyright (c) 2014 Carpentier Pierre-Francois

# This is a demo backend
```

```
from ldapcherry.exceptions import MissingParameter
from sets import Set
import ldapcherry.backend
import re


class Backend(ldapcherry.backend.Backend):

    def __init__(self, config, logger, name, attrslist, key):
        """ Initialize the backend

        :param config: the configuration of the backend
        :type config: dict {'config key': 'value'}
        :param logger: the cherrypy error logger object
        :type logger: python logger
        :param name: id of the backend
        :type name: string
        :param attrslist: list of the backend attributes
        :type attrslist: list of strings
        :param key: the key attribute
        :type key: string
        """
        self.config = config
        self._logger = logger
        self.users = {}
        self.backend_name = name
        admin_user = self.get_param('admin.user', 'admin')
        admin_password = self.get_param('admin.password', 'admin')
        admin_groups = Set(re.split('\W+', self.get_param('admin.groups')))
        basic_user = self.get_param('basic.user', 'user')
        basic_password = self.get_param('basic.password', 'user')
        basic_groups = Set(re.split('\W+', self.get_param('basic.groups')))
        pwd_attr = self.get_param('pwd_attr')
        self.search_attrs = Set(
            re.split('\W+', self.get_param('search_attributes')),
            )
        self.pwd_attr = pwd_attr
        self.admin_user = admin_user
        self.basic_user = basic_user
        self.key = key
        self.users[admin_user] = {
                key: admin_user,
                pwd_attr: admin_password,
                'groups': admin_groups,
                }
        self.users[basic_user] = {
                key: basic_user,
                pwd_attr: basic_password,
                'groups': basic_groups,
                }

    def _check_fix_users(self, username):
        if self.admin_user == username or self.basic_user == username:
            raise Exception('User cannot be modified')

    def auth(self, username, password):
        """ Check authentication against the backend
```

```
        :param username: 'key' attribute of the user
        :type username: string
        :param password: password of the user
        :type password: string
        :rtype: boolean (True is authentication success, False otherwise)
        """
        if username not in self.users:
            return False
        elif self.users[username][self.pwd_attr] == password:
            return True
        return False

    def add_user(self, attrs):
        """ Add a user to the backend

        :param attrs: attributes of the user
        :type attrs: dict ({<attr>: <value>})

        .. warning:: raise UserAlreadyExists if user already exists
        """
        username = attrs[self.key]
        if username in self.users:
            raise UserAlreadyExists(username, self.backend_name)
        self.users[username] = attrs
        self.users[username]['groups'] = Set([])

    def del_user(self, username):
        """ Delete a user from the backend

        :param username: 'key' attribute of the user
        :type username: string

        """
        self._check_fix_users(username)
        del self.users[username]

    def set_attrs(self, username, attrs):
        """ Set a list of attributes for a given user

        :param username: 'key' attribute of the user
        :type username: string
        :param attrs: attributes of the user
        :type attrs: dict ({<attr>: <value>})
        """
        self._check_fix_users(username)
        for attr in attrs:
            self.users[username][attr] = attrs[attr]

    def add_to_groups(self, username, groups):
        """ Add a user to a list of groups

        :param username: 'key' attribute of the user
        :type username: string
        :param groups: list of groups
        :type groups: list of strings
        """
        self._check_fix_users(username)
        current_groups = self.users[username]['groups']
```

```
        new_groups = current_groups | Set(groups)
        self.users[username]['groups'] = new_groups

    def del_from_groups(self, username, groups):
        """ Delete a user from a list of groups

        :param username: 'key' attribute of the user
        :type username: string
        :param groups: list of groups
        :type groups: list of strings

        .. warning:: raise GroupDoesntExist if group doesn't exist
        """
        self._check_fix_users(username)
        current_groups = self.users[username]['groups']
        new_groups = current_groups - Set(groups)
        self.users[username]['groups'] = new_groups

    def search(self, searchstring):
        """ Search backend for users

        :param searchstring: the search string
        :type searchstring: string
        :rtype: dict of dict ( {<user attr key>: {<attr>: <value>}} )
        """
        ret = {}
        for user in self.users:
            match = False
            for attr in self.search_attrs:
                if attr not in self.users[user]:
                    pass
                elif re.search(searchstring + '.*', self.users[user][attr]):
                    match = True
            if match:
                ret[user] = self.users[user]
        return ret

    def get_user(self, username):
        """ Get a user's attributes

        :param username: 'key' attribute of the user
        :type username: string
        :rtype: dict ( {<attr>: <value>} )

        .. warning:: raise UserDoesntExist if user doesn't exist
        """
        return self.users[username]

    def get_groups(self, username):
        """ Get a user's groups

        :param username: 'key' attribute of the user
        :type username: string
        :rtype: list of groups
        """
        return self.users[username]['groups']
```

CHAPTER 6

Implementing password policy modules

6.1 API

The password policy modules must respect following API:

class `ldapcherry.ppolicy.`**`PPolicy`**(*config*, *logger*)

`__init__`(*config*, *logger*)
Password policy constructor

Parameters

- **`config`** (*dict {'config key': 'value'}*) – the configuration of the ppolicy
- **`logger`** (*python logger*) – the cherrypy error logger object

`check`(*password*)
Check if a password match the ppolicy

Parameters **`password`** (*string*) – the password to check

Return type dict with keys 'match' a boolean (True if ppolicy matches, False otherwise) and 'reason', an explaination string

`info`()
Give information about the ppolicy

Return type a string describing the ppolicy

6.2 Configuration

Parameters are declared in the main configuration file, inside the **ppolicy** section.

After having set **self.config** to **config** in the constructor, parameters can be recovered by **self.get_param**:

class `ldapcherry.ppolicy.`**`PPolicy`**(*config*, *logger*)

`get_param`(*param*, *default=None*)
Get a parameter in config (handle default value)

Parameters

- **`param`** (*string*) – name of the parameter to recover

- **default** (*string or None*) – the default value, raises an exception if param is not in configuration and default is None (which is the default value).

Return type the value of the parameter or the default value if not set in configuration

6.3 Example

Here is the simple default ppolicy module that comes with LdapCherry:

```
# -*- coding: utf-8 -*-
# vim:set expandtab tabstop=4 shiftwidth=4:
#
# The MIT License (MIT)
# LdapCherry
# Copyright (c) 2014 Carpentier Pierre-Francois

import ldapcherry.ppolicy
import re


class PPolicy(ldapcherry.ppolicy.PPolicy):

    def __init__(self, config, logger):
        self.config = config
        self.min_length = self.get_param('min_length')
        self.min_upper = self.get_param('min_upper')
        self.min_digit = self.get_param('min_digit')

    def check(self, password):
        if len(password) < self.min_length:
            return {'match': False, 'reason': 'password too short'}
        if len(re.findall(r'[A-Z]', password)) < self.min_upper:
            return {
                'match': False,
                'reason': 'not enough upper case characters'
                }
        if len(re.findall(r'[0-9]', password)) < self.min_digit:
            return {'match': False, 'reason': 'not enough digits'}
        return {'match': True, 'reason': 'password ok'}

    def info(self):
        return \
            "* Minimum length: %(len)n\n" \
            "* Minimum number of uppercase characters: %(upper)n\n" \
            "* Minimum number of digits: %(digit)n" % {
                'upper': self.min_upper,
                'len': self.min_length,
                'digit': self.min_digit
                }
```

CHAPTER 7

Changelog

7.1 Dev

7.2 Version 0.1.0

- add demo backend
- add custom javascript hook
- add documentation for backends
- add the Active Directory backend
- add display name parameter for backends
- fix many encoding error in LDAP backend
- fix dn renaming of an entry in LDAP backend
- turn-off configuration monitoring
- better exception handling and debugging logs

7.3 Version 0.0.1

- first release

CHAPTER 8

Some Goodies

Here are some goodies that might help deploying LdapCherry

They are located in the **goodies/** directory.

8.1 Init Script

Sample init script for Debian:

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:        ldapcherryd
# Required-Start:    $remote_fs $network $syslog
# Required-Stop:    $remote_fs $network $syslog
# Default-Start:    2 3 4 5
# Default-Stop:
# Short-Description:    ldapcherry
### END INIT INFO

PIDFILE=/var/run/ldapcherryd/ldapcherryd.pid
CONF=/etc/ldapcherry/ldapcherry.ini
USER=www-data
GROUP=www-data
BIN=/usr/local/bin/ldapcherryd
OPTS="-d -c $CONF -p $PIDFILE"

. /lib/lsb/init-functions

if [ -f /etc/default/ldapcherryd ]; then
    . /etc/default/ldapcherryd
fi

start_ldapcherryd(){
    log_daemon_msg "Starting ldapcherryd" "ldapcherryd" || true
    pidofproc -p $PIDFILE $BIN >/dev/null
    status="$?"
    if [ $status -eq 0 ]
    then
        log_end_msg 1
        log_failure_msg \
        "ldapcherryd already started"
```

```
        return 1
    fi
    mkdir -p `dirname $PIDFILE` -m 750
    chown $USER:$GROUP `dirname $PIDFILE`
    if start-stop-daemon -c $USER:$GROUP --start \
        --quiet --pidfile $PIDFILE \
        --oknodo --exec $BIN -- $OPTS
    then
        log_end_msg 0 || true
        return 0
    else
        log_end_msg 1 || true
        return 1
    fi

}

stop_ldapcherryd(){
    log_daemon_msg "Stopping ldapcherryd" "ldapcherryd" || true
    if start-stop-daemon --stop --quiet \
        --pidfile $PIDFILE
    then
        log_end_msg 0 || true
        return 0
    else
        log_end_msg 1 || true
        return 1
    fi
}

case "$1" in
  start)
    start_ldapcherryd
    exit $?
    ;;
  stop)
    stop_ldapcherryd
    exit $?
    ;;
  restart)
    stop_ldapcherryd
    while pidofproc -p $PIDFILE $BIN >/dev/null
    do
        sleep 0.5
    done
    start_ldapcherryd
    exit $?
    ;;
  status)
    status_of_proc -p $PIDFILE $BIN "ldapcherryd" \
        && exit 0 || exit $?
    ;;
  *)
    log_action_msg \
    "Usage: /etc/init.d/ldapcherryd {start|stop|restart|status}" \
    || true
    exit 1
esac
```

```
exit 0
```

This init script is available in **goodies/init-debian**.

8.2 Apache Vhost

```
<VirtualHost *:80>

    <Location />
        ProxyPass http://127.0.0.1:8080/
        ProxyPassReverse http://127.0.0.1:8080/
    </Location>

</VirtualHost>
```

8.3 Nginx Vhost

```
server {
    listen 80 default_server;

    server_name $hostname;
    #access_log /var/log/nginx/dnscherry_access_log;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-for $proxy_add_x_forwarded_for;
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Proto $remote_addr;
    }
}
```

8.4 Lighttpd Vhost

```
server.modules += ("mod_proxy")

$HTTP["host"] == "ldapcherry.kakwa.fr" {
    proxy.server  = ( "" =>
        (( "host" => "127.0.0.1", "port" => 8080 ))
    )
}
```

CHAPTER 9

Screenshots

CHAPTER 10

LdapCherry

Nice and simple application to manage users and groups in multiple directory services.

Doc ldapcherry documentation on ReadTheDoc

Dev ldapcherry source code on GitHub

PyPI ldapcherry package on Pypi

License MIT

Author Pierre-Francois Carpentier - copyright © 2015

CHAPTER 11

Presentation

LdapCherry is a CherryPY application to manage users and groups in multiple directory services.

It's main features are:

- manage multiple directories/databases backends in an unified way
- roles management (as in "groups of groups")
- autofill forms
- password policy
- self modification of some selected fields by normal (non administrator) users
- nice bootstrap interface
- modular through pluggable authentication, password policy and backend modules

LdapCherry is not limited to ldap, it can handle virtually any user backend (ex: SQL database, htpasswd file, etc) through the proper plugin (provided that it is implemented ^^).

LdapCherry also aims to be as simple as possible to deploy: no crazy dependencies, few configuration files, extensive debug logs and full documentation.

CHAPTER 12

Screenshots

Screenshots.

CHAPTER 13

License

LdapCherry is published under the MIT Public License.

CHAPTER 14

Discussion / Help / Updates

- IRC: Freenode `#ldapcherry` channel
- Bugtracker: Github

Python Module Index

l

ldapcherry.exceptions, 27

Symbols

__init__() (ldapcherry.backend.Backend method), 25
__init__() (ldapcherry.ppolicy.PPolicy method), 31

A

add_to_groups() (ldapcherry.backend.Backend method), 25
add_user() (ldapcherry.backend.Backend method), 25
auth() (ldapcherry.backend.Backend method), 25

B

Backend (class in ldapcherry.backend), 25, 27

C

check() (ldapcherry.ppolicy.PPolicy method), 31

D

del_from_groups() (ldapcherry.backend.Backend method), 26
del_user() (ldapcherry.backend.Backend method), 26

G

get_groups() (ldapcherry.backend.Backend method), 26
get_param() (ldapcherry.backend.Backend method), 27
get_param() (ldapcherry.ppolicy.PPolicy method), 31
get_user() (ldapcherry.backend.Backend method), 26
GroupDoesntExist, 27

I

info() (ldapcherry.ppolicy.PPolicy method), 31

L

ldapcherry.exceptions (module), 27

P

PPolicy (class in ldapcherry.ppolicy), 31

S

search() (ldapcherry.backend.Backend method), 26
set_attrs() (ldapcherry.backend.Backend method), 26

U

UserAlreadyExists, 27
UserDoesntExist, 27