openwisp-radius Documentation

Release 1.0.1

Fiorella De Luca

May 16, 2022

CONTENTS:

OpenWISP-RADIUS is Django reusable app that provides an admin interface to a freeradius database.

Note: If you're building a public wifi service, we suggest to take a look at openwisp-wifi-login-pages, which is built to work with openwisp-radius.

CHAPTER

ONE

SETUP

1.1 Deploy it in production

An automated installer is available at ansible-openwisp2.

1.2 Create a virtual environment

Please use a python virtual environment. It keeps everybody on the same page, helps reproducing bugs and resolving problems.

We highly suggest to use **virtualenvwrapper**, please refer to the official virtualenvwrapper installation page and come back here when ready to proceed.

```
# create virtualenv
mkvirtualenv radius
```

Note: If you encounter an error like `Python could not import the module virtualenvwrapper`, add `VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3` and run `source virtualenvwrapper.sh` again :)

1.3 Install required system packages

Install packages required by Weasyprint for your OS:

- Linux
- MacOS
- Windows

1.4 Install stable version from pypi

Install from pypi:

```
# REQUIRED: update base python packages
pip install -U pip setuptools wheel
# install openwisp-radius
pip install openwisp-radius
```

1.5 Install development version

Install tarball:

```
# REQUIRED: update base python packages
pip install -U pip setuptools wheel
# install openwisp-radius
pip install https://github.com/openwisp/openwisp-radius/tarball/master
```

Alternatively you can install via pip using git:

```
# REQUIRED: update base python packages
pip install -U pip setuptools wheel
# install openwisp-radius
pip install -e git+git://github.com/openwisp/openwisp-radius#egg=openwisp-radius
```

If you want to contribute, install your cloned fork:

```
# REQUIRED: update base python packages
pip install -U pip setuptools wheel
# install your forked openwisp-radius
git clone git@github.com:<your_fork>/openwisp-radius.git
cd openwisp-radius
pip install -e .
```

1.6 Setup (integrate in an existing django project)

The `settings.py` file of your project should have at least the following modules listed `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.humanize',
    # openwisp admin theme
    'openwisp_utils.admin_theme',
    # all-auth
    'django.contrib.sites',
```

(continues on next page)

(continued from previous page)

```
    'allauth',
    'allauth.account',
    # admin
    'django.contrib.admin',
    # rest framework
    'rest_framework',
    'django_filters',
    # registration
    'rest_framework.authtoken',
    'dj_rest_auth',
    'dj_rest_auth.registration',
    # openwisp radius
    'openwisp_radius',
    'openwisp_users',
    'private_storage',
    'drf_yasg',
]
```

These modules are optional, add them only if you need the *social login* feature:

```
INSTALLED_APPS += [
    # social login
    'allauth.socialaccount',
    'allauth.socialaccount.providers.facebook',
    'allauth.socialaccount.providers.google',
]
```

Add media locations in `settings.py`:

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
PRIVATE_STORAGE_ROOT = os.path.join(MEDIA_ROOT, 'private')
```

Also, add `AUTH_USER_MODEL`, `AUTHENTICATION_BACKENDS` and `SITE_ID` to your `settings.py`:

```
AUTH_USER_MODEL = 'openwisp_users.User'
SITE_ID = 1
AUTHENTICATION_BACKENDS = (
    'openwisp_users.backends.UsersAuthenticationBackend',
)
```

Add allowed freeradius hosts in `settings.py`:

```
OPENWISP_RADIUS_FREERADIUS_ALLOWED_HOSTS = ['127.0.0.1']
```

Note: Read more about *freeradius allowed hosts in settings page*.

Add the URLs to your main `urls.py`:

```
from openwisp_radius.urls import get_urls

urlpatterns = [
```

(continues on next page)

(continued from previous page)

```
    # ... other urls in your project ...

    # django admin interface urls
    path('admin/', admin.site.urls),
    # openwisp-radius urls
    path('api/v1/', include('openwisp_utils.api.urls')),
    path('api/v1/', include('openwisp_users.api.urls')),
    path('accounts/', include('openwisp_users.accounts.urls')),
    path('', include('openwisp_radius.urls'))
]
```

Then run:

```
./manage.py migrate
```

1.7 Migrating an existing freeradius database

If you already have a freeradius 3 database with the default schema, you should be able to use it with openwisp-radius (and extended apps) easily:

1. first of all, back up your existing database;
2. configure django to connect to your existing database;
3. fake the first migration (which only replicates the default freeradius schema) and then launch the rest of migrations normally, see the examples below to see how to do this.

```
./manage.py migrate --fake openwisp-radius 0001_initial_freeradius
./manage.py migrate
```

1.8 Automated periodic tasks

Some periodic commands are required in production environments to enable certain features and facilitate database cleanup. There are two ways to automate these tasks:

1.8.1 1. Celery-beat (Recommended Method)

1. You need to create a celery configuration file as it's created in example file.
2. Add celery to `__init__.py` of your project:

```
from .celery import app as celery_app

__all__ = ['celery_app']
```

3. In the settings.py, configure the CELERY_BEAT_SCHEDULE. Some celery tasks take an argument, for instance `365` is given here for `delete_old_radacct` in the example settings. These arguments are passed to their respective management commands. More information about these parameters can be found at the management commands page.

Note: Celery tasks do not start with django server and need to be started seperately, please read about running celery and celery-beat tasks.

1.8.2 2. Crontab (Legacy Method)

Edit the crontab with:

```
crontab -e
```

Add and modify the following lines accordingly:

```
# This command deletes RADIUS accounting sessions older than 365 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_radacct 365

# This command deletes RADIUS post-auth logs older than 365 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_postauth 365

# This command closes stale RADIUS sessions that have remained open for 15 days
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py cleanup_stale_radacct␣
↪15

# This command deactivates expired user accounts which were created temporarily
# (eg: for en event) and have an expiration date set.
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py deactivate_expired_
↪users

# This command deletes users that have expired (and should have
# been deactivated by deactivate_expired_users) for more than
# 18 months (which is the default duration)
30 04 * * * <virtualenv_path>/bin/python <full/path/to>/manage.py delete_old_users
```

Be sure to replace <virtualenv_path> with the absolute path to the Python virtual environment.

Also, change <full/path/to> to the directory where manage.py is.

To get the absolute path to manage.py when openwisp-radius is installed for development, navigate to the base directory of the cloned fork. Then, run:

```
cd tests/
pwd
```

Note: More information can be found at the *management commands page*.

1.9 Installing for development

Install python3-dev and gcc:

```
sudo apt install python3-dev gcc
```

Install sqlite:

```
sudo apt install sqlite3 libsqlite3-dev libpq-dev
```

Install mysqlclient:

```
sudo apt install libmysqlclient-dev libssl-dev
```

Note: If you are on Debian 10 or 9 you may need to install `default-libmysqlclient-dev` instead

Install xmlsec1:

```
sudo apt install xmlsec1
```

Install your forked repo:

```
git clone git://github.com/<your_username>/openwisp-radius
cd openwisp-radius/
pip install -e .[saml,openvpn_status]
```

Install test requirements:

```
pip install -r requirements-test.txt
```

Create database:

```
cd tests/
./manage.py migrate
./manage.py createsuperuser
```

Launch development server:

```
./manage.py runserver
```

You can access the admin interface at http://127.0.0.1:8000/admin/.

Run tests with:

```
./runtests.py
```

1.10 Celery Usage

To run celery, you need to start redis-server. You can install redis on your machine or install docker and run redis inside docker container:

```
docker run -p 6379:6379 --name openwisp-redis -d redis:alpine
```

Run celery (it is recommended to use a tool like supervisord in production):

```
# Optionally, use ``--detach`` argument to avoid using multiple terminals
celery -A openwisp2 worker -l info
celery -A openwisp2 beat -l info
```

1.11 Troubleshooting

If you encounter any issue during installation, run:

```
pip install -e .[saml] -r requirements-test.txt
```

instead of `pip install -r requirements-test.txt`

CHAPTER

TWO

FREERADIUS SETUP FOR CAPTIVE PORTAL AUTHENTICATION

This guide explains how to install and configure freeradius 3 in order to make it work with OpenWISP RADIUS for Captive Portal authentication.

The guide is written for debian based systems, other linux distributions can work as well but the name of packages and files may be different.

Widely used solutions used with OpenWISP RADIUS are PfSense and Coova-Chilli, but other solutions can be used as well.

Note: Before users can authenticate through a captive portal, they will most likely need to sign up through a web page, or alternatively, they will need to perform social login or some other kind of Single Sign On (SSO).

The openwisp-wifi-login-pages web app is an open source solution which integrates with OpenWISP RADIUS to provide features like self user registration, social login, SSO/SAML login, SMS verification, simple username & password login using the *Radius User Token* method.

For more information see: openwisp-wifi-login-pages.

2.1 How to install freeradius 3

First of all, become root:

```
sudo -s
```

In order to **install a recent version of FreeRADIUS**, we recommend using the freeradius packages provided by NetworkRADIUS.

After having updated the APT sources list to pull the NetworkRADIUS packages, let's proceed to update the list of available packages:

```
apt update
```

These packages are always needed:

```
apt install freeradius freeradius-rest
```

If you use MySQL:

```
apt install freeradius-mysql
```

If you use PostgreSQL:

```
apt install freeradius-postgresql
```

Warning: You have to install and configure an SQL database like PostgreSQL, MySQL (SQLite can also work, but we won't treat it here) and make sure both OpenWISP RADIUS and Freeradius point to it.

The steps outlined above may not be sufficient to get the DB of your choice to run, please consult the documentation of your database of choice for more information on how to get it to run properly.

In the rest of this document we will mention PostgreSQL often because that is the database generally preferred by the Django community.

2.2 Configuring Freeradius 3

For a complete reference on how to configure freeradius please read the Freeradius wiki, configuration files and their configuration tutorial.

Note: The path to freeradius configuration could be different on your system. This article use the `/etc/freeradius/` directory that ships with recent debian distributions and its derivatives

Refer to the mods-available documentation for the available configuration values.

2.2.1 Enable the configured modules

First of all enable the `rest` and optionally the `sql` module:

```
ln -s /etc/freeradius/mods-available/rest /etc/freeradius/mods-enabled/rest
# optional
ln -s /etc/freeradius/mods-available/sql /etc/freeradius/mods-enabled/sql
```

2.2.2 Configure the REST module

Configure the rest module by editing the file `/etc/freeradius/mods-enabled/rest`, substituting <url> with your django project's URL, (for example, if you are testing a development environment, the URL could be `http://127.0.0.1:8000`, otherwise in production could be something like `https://openwisp2.mydomain.org`)-

Warning: Remember you need to add your freeradius server IP address in openwisp freeradius allowed hosts settings. If the freeradius server IP is not in allowed hosts, all requests to openwisp radius API will return `403`.

Refer to the rest module documentation for the available configuration values.

```
# /etc/freeradius/mods-enabled/rest

connect_uri = "<url>"

authorize {
    uri = "${..connect_uri}/api/v1/freeradius/authorize/"
```

(continues on next page)

(continued from previous page)

```
    method = 'post'
    body = 'json'
    data = '{"username": "%{User-Name}", "password": "%{User-Password}"}'
    tls = ${..tls}
}

# this section can be left empty
authenticate {}

post-auth {
    uri = "${..connect_uri}/api/v1/freeradius/postauth/"
    method = 'post'
    body = 'json'
    data = '{"username": "%{User-Name}", "password": "%{User-Password}", "reply": "%
↪{reply:Packet-Type}", "called_station_id": "%{Called-Station-ID}", "calling_station_id
↪": "%{Calling-Station-ID}"}'
    tls = ${..tls}
}

accounting {
    uri = "${..connect_uri}/api/v1/freeradius/accounting/"
    method = 'post'
    body = 'json'
    data = '{"status_type": "%{Acct-Status-Type}", "session_id": "%{Acct-Session-Id}",
↪"unique_id": "%{Acct-Unique-Session-Id}", "username": "%{User-Name}", "realm": "%
↪{Realm}", "nas_ip_address": "%{NAS-IP-Address}", "nas_port_id": "%{NAS-Port}", "nas_
↪port_type": "%{NAS-Port-Type}", "session_time": "%{Acct-Session-Time}", "authentication
↪": "%{Acct-Authentic}", "input_octets": "%{Acct-Input-Octets}", "output_octets": "%
↪{Acct-Output-Octets}", "called_station_id": "%{Called-Station-Id}", "calling_station_id
↪": "%{Calling-Station-Id}", "terminate_cause": "%{Acct-Terminate-Cause}", "service_type
↪": "%{Service-Type}", "framed_protocol": "%{Framed-Protocol}", "framed_ip_address": "%
↪{Framed-IP-Address}"}'
    tls = ${..tls}
}
```

2.2.3 Configure the SQL module

Note: The `sql` module is not extremely needed but we treat it here since it can be useful to implement custom behavior, moreover we treat it in this document also to show that OpenWISP RADIUS can integrate itself with other widely used FreeRADIUS modules.

Once you have configured properly an SQL server, e.g. PostgreSQL:, and you can connect with a username and password edit the file `/etc/freeradius/mods-available/sql` to configure Freeradius to use the relational database.

Change the configuration for `driver`, `dialect`, `server`, `port`, `login`, `password`, `radius_db` as you need to fit your SQL server configuration.

Refer to the sql module documentation for the available configuration values.

Example configuration using the PostgreSQL database:

```
# /etc/freeradius/mods-available/sql

driver = "rlm_sql_postgresql"
dialect = "postgresql"

# Connection info:
server = "localhost"
port = 5432
login = "<user>"
password = "<password>"
radius_db = "radius"
```

2.2.4 Configure the site

This section explains how to configure the FreeRADIUS site.

Please refer to *FreeRADIUS API Authentication* to understand the different possibilities with which FreeRADIUS can authenticate requests going to OpenWISP RADIUS so that OpenWISP RADIUS knows to which organization each request belongs.

If you are **not** using the method described in *Radius User Token*, you have to do the following:

- create one FreeRADIUS site for each organization
- uncomment the line which starts with `# api_token_header`
- substitute the occurrences of `<org_uuid>` and `<org_radius_api_token>` with the UUID & RADIUS API token of each organization, refer to the section *Organization UUID & RADIUS API Token* for finding these values.

If you are deploying a captive portal setup and can use the RADIUS User Token method, you can get away with having only one freeradius site for all the organizations and can simply copy the configuration shown below.

```
# /etc/freeradius/sites-enabled/default
# Remove `#` symbol from the line to uncomment it

server default {
    # if you are not using Radius Token authentication method, please uncomment
    # and set the values for <org_uuid> & <org_radius_api_token>
    # api_token_header = "Authorization: Bearer <org_uuid> <org_radius_api_token>"

    authorize {
        # if you are not using Radius Token authentication method, please uncomment the␣
↪following
        # update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
    }

    # this section can be left empty
    authenticate {}

    post-auth {
        # if you are not using Radius Token authentication method, please uncomment the␣
↪following
```

(continues on next page)

(continued from previous page)

```
        # update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest

        Post-Auth-Type REJECT {
            # if you are not using Radius Token authentication method, please uncomment␣
↪the following
            # update control { &REST-HTTP-Header += "${....api_token_header}" }
            rest
        }
    }

    accounting {
        # if you are not using Radius Token authentication method, please uncomment the␣
↪following
        # update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
    }
}
```

Please also ensure that `acct_unique` is present in the `pre-accounting` section:

```
preacct {
    # ...
    acct_unique
    # ...
}
```

2.2.5 Restart freeradius to make the configuration effective

Restart freeradius to load the new configuration:

```
service freeradius restart
# alternatively if you are using systemd
systemctl restart freeradius
```

In case of errors you can run freeradius in debug mode by running `freeradius -X` in order to find out the reason of the failure.

A common problem, especially during development and testing, is that the openwisp-radius application may not be running, in that case you can find out how to run the django development server in the *Install for development* section.

Also make sure that this server runs on the port specified in `/etc/freeradius/mods-enabled/rest`.

You may also want to take a look at the Freeradius documentation for further information that is freeradius specific.

2.2.6 Reconfigure the development environment using PostgreSQL

You'll have to reconfigure the development environment as well before being able to use openwisp-radius for managing the freeradius databases.

If you have installed for development, create a file `tests/local_settings.py` and add the following code to configure the database:

```
# openwisp-radius/tests/local_settings.py
  DATABASES = {
     'default': {
         'ENGINE': 'django.db.backends.postgresql_psycopg2',
         'NAME': '<db_name>',
         'USER': '<db_user>',
         'PASSWORD': '<db_password>',
         'HOST': '127.0.0.1',
         'PORT': '5432'
     },
  }
```

Make sure the database by the name <db_name> is created and also the role <db_user> with <db_password> as password.

2.3 Using Radius Checks for Authorization Information

Traditionally, when using an SQL backend with Freeradius, user authorization information such as User-Name and "known good" password can be stored using the *radcheck* table provided by Freeradius' default SQL schema.

OpenWISP RADIUS instead uses the FreeRADIUS rlm_rest module in order to take advantage of the built in user management and authentication capabilities of Django (for more information about these topics see *Configure the REST module* and User authentication in Django).

When migrating from existing FreeRADIUS deployments or in cases where it is preferred to use the FreeRADIUS *radcheck* table for storing user credentials it is possible to utilize rlm_sql in parallel with (or instead of) rlm_rest for authorization.

Note: Bypassing the REST API of openwisp-radius means that you will have to manually create the radius check entries for each user you want to authenticate with FreeRADIUS.

2.3.1 Configuration

To configure support for accessing user credentials with Radius Checks ensure the `authorize` section of your site as follows contains the `sql` module:

```
# /etc/freeradius/sites-available/default

authorize {
    # ...
    sql  # <-- the sql module
    # ...
}
```

2.4 Debugging

In this section we will explain how to debug your freeradius instance.

2.4.1 Start freeradius in debug mode

When debugging we suggest you to open up a dedicated terminal window to run freeradius in debug mode:

```
# we need to stop the main freeradius process first
service freeradius stop
# alternatively if you are using systemd
systemctl stop freeradius
# launch freeradius in debug mode
freeradius -X
```

2.4.2 Testing authentication and authorization

You can do this with `radtest`:

```
# radtest <username> <password> <host> 10 <secret>
radtest admin admin localhost 10 testing123
```

A successful authentication will return similar output:

```
Sent Access-Request Id 215 from 0.0.0.0:34869 to 127.0.0.1:1812 length 75
    User-Name = "admin"
    User-Password = "admin"
    NAS-IP-Address = 127.0.0.1
    NAS-Port = 10
    Message-Authenticator = 0x00
    Cleartext-Password = "admin"
Received Access-Accept Id 215 from 127.0.0.1:1812 to 0.0.0.0:0 length 20
```

While an unsuccessful one will look like the following:

```
Sent Access-Request Id 85 from 0.0.0.0:51665 to 127.0.0.1:1812 length 73
    User-Name = "foo"
    User-Password = "bar"
    NAS-IP-Address = 127.0.0.1
    NAS-Port = 10
    Message-Authenticator = 0x00
    Cleartext-Password = "bar"
Received Access-Reject Id 85 from 127.0.0.1:1812 to 0.0.0.0:0 length 20
(0) -: Expected Access-Accept got Access-Reject
```

Alternatively, you can use `radclient` which allows more complex tests; in the following example we show how to test an authentication request which includes `Called-Station-ID` and `Calling-Station-ID`:

```
user="foo"
pass="bar"
called="00-11-22-33-44-55:localhost"
```

(continues on next page)

(continued from previous page)

```
calling="00:11:22:33:44:55"
request="User-Name=$user,User-Password=$pass,Called-Station-ID=$called,Calling-Station-
↪ID=$calling"
echo $request | radclient localhost auth testing123
```

2.4.3 Testing accounting

You can do this with `radclient`, but first of all you will have to create a text file like the following one:

```
# /tmp/accounting.txt

Acct-Session-Id = "35000006"
User-Name = "jim"
NAS-IP-Address = 172.16.64.91
NAS-Port = 1
NAS-Port-Type = Async
Acct-Status-Type = Interim-Update
Acct-Authentic = RADIUS
Service-Type = Login-User
Login-Service = Telnet
Login-IP-Host = 172.16.64.25
Acct-Delay-Time = 0
Acct-Session-Time = 261
Acct-Input-Octets = 9900909
Acct-Output-Octets = 10101010101
Called-Station-Id = 00-27-22-F3-FA-F1:hostname
Calling-Station-Id = 5c:7d:c1:72:a7:3b
```

Then you can call `radclient`:

```
radclient -f /tmp/accounting.txt -x 127.0.0.1 acct testing123
```

You should get the following output:

```
Sent Accounting-Request Id 83 from 0.0.0.0:51698 to 127.0.0.1:1813 length 154
    Acct-Session-Id = "35000006"
    User-Name = "jim"
    NAS-IP-Address = 172.16.64.91
    NAS-Port = 1
    NAS-Port-Type = Async
    Acct-Status-Type = Interim-Update
    Acct-Authentic = RADIUS
    Service-Type = Login-User
    Login-Service = Telnet
    Login-IP-Host = 172.16.64.25
    Acct-Delay-Time = 0
    Acct-Session-Time = 261
    Acct-Input-Octets = 9900909
    Acct-Output-Octets = 1511075509
    Called-Station-Id = "00-27-22-F3-FA-F1:hostname"
    Calling-Station-Id = "5c:7d:c1:72:a7:3b"
Received Accounting-Response Id 83 from 127.0.0.1:1813 to 0.0.0.0:0 length 20
```

2.5 Customizing your configuration

You can further customize your freeradius configuration and exploit the many features of freeradius but you will need to test how your configuration plays with *openwisp-radius*.

CHAPTER

THREE

FREERADIUS SETUP FOR WPA ENTERPRISE (EAP-TTLS-PAP) AUTHENTICATION

This guide explains how to install and configure freeradius 3 in order to make it work with OpenWISP RADIUS for WPA Enterprise EAP-TTLS-PAP authentication.

The setup will allow users to authenticate via WiFi WPA Enterprise networks using their personal username and password of their django user accounts. Users can either be created manually via the admin interface, *generated*, *imported from CSV*, or can self register through a web page which makes use of the *registration REST API* (like openwisp-wifi-login-pages).

3.1 Prerequisites

Execute the steps explained in the following sections of the *freeradius guide for captive portal authentication*:

- How to install freeradius 3
- Enable the configured modules
- Configure the REST module

Then proceed with the rest of the document.

3.2 Freeradius configuration

3.2.1 Configure the sites

Main sites

In this scenario it is necessary to set up one FreeRADIUS site for each organization you want to support, each FreeRADIUS instance will therefore need two dedicated ports, one for authentication and one for accounting and a related inner tunnel configuration.

Let's create the site for an hypotethical organization called org-A.

Don't forget to substitute the occurrences of `<org_uuid>` and `<org_radius_api_token>` with the UUID & Radius API token of each organization, refer to the section *Organization UUID & RADIUS API Token* for finding these values.

```
# /etc/freeradius/sites-enabled/org_a

server org_a {
    listen {
```

(continues on next page)

(continued from previous page)

```
        type = auth
        ipaddr = *
        # ensure each org has its own port
        port = 1812
        # adjust these as needed
        limit {
          max_connections = 16
          lifetime = 0
          idle_timeout = 30
        }
    }

    listen {
        ipaddr = *
        # ensure each org has its own port
        port = 1813
        type = acct
        limit {}
    }

    # IPv6 configuration skipped for brevity
    # consult the freeradius default configuration if you need
    # to add the IPv6 configuration

    # Substitute the following variables with
    # the organization UUID and RADIUS API Token
    api_token_header = "Authorization: Bearer <org_uuid> <org_radius_api_token>"

    authorize {
        eap-org_a {
           ok = return
        }

        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
    }

    authenticate {
        Auth-Type eap-org_a {
            eap-org_a
        }
    }

    post-auth {
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest

        Post-Auth-Type REJECT {
            update control { &REST-HTTP-Header += "${....api_token_header}" }
            rest
        }
    }
```

(continues on next page)

(continued from previous page)

```
    accounting {
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest
    }
}
```

Please also ensure that `acct_unique` is present in the `pre-accounting` section:

```
preacct {
    # ...
    acct_unique
    # ...
}
```

Inner tunnels

You will need to set up one inner tunnel for each organization too.

Following the example for a hypotetical organization named org-A:

```
# /etc/freeradius/sites-enabled/inner-tunnel

server inner-tunnel_org_a {
    listen {
        ipaddr = 127.0.0.1
        # each org will need a dedicated port for their inner tunnel
        port = 18120
        type = auth
    }

    api_token_header = "Authorization: Bearer <org_uuid> <org_radius_api_token>"

    authorize {
        filter_username
        update control { &REST-HTTP-Header += "${...api_token_header}" }
        rest

        eap-org_a {
            ok = return
        }

        expiration
        logintime

        pap
    }

    authenticate {
        Auth-Type PAP {
            pap
        }
```

(continues on next page)

(continued from previous page)

```
        Auth-Type CHAP {
            chap
        }

        Auth-Type MS-CHAP {
            mschap
        }
        eap-org_a
    }

    session {}

    post-auth {
    }

    pre-proxy {}
    post-proxy {
        eap-org_a
    }
}
```

3.2.2 Configure the EAP modules

Note: Keep in mind these are basic sample configurations, once you get it working feel free to tweak it to make it more secure and fully featured.

You will need to set up one EAP module instance for each organization too.

Following the example for a hypotetical organization named org-A:

```
eap eap-org_a {
    default_eap_type = ttls
    timer_expire = 60
    ignore_unknown_eap_types = no
    cisco_accounting_username_bug = no
    max_sessions = ${max_requests}

    tls-config tls-common {
        # make sure to have a valid SSL certificate for production usage
        private_key_password = whatever
        private_key_file = /etc/ssl/private/ssl-cert-snakeoil.key
        certificate_file = /etc/ssl/certs/ssl-cert-snakeoil.pem
        ca_file = /etc/ssl/certs/ca-certificates.crt
        dh_file = ${certdir}/dh
        ca_path = ${cadir}
        cipher_list = "DEFAULT"
        cipher_server_preference = no
        ecdh_curve = "prime256v1"
```

(continues on next page)

(continued from previous page)

```
        cache {
            enable = no
        }

        ocsp {
            enable = no
            override_cert_url = yes
            url = "http://127.0.0.1/ocsp/"
        }
    }

    ttls {
        tls = tls-common
        default_eap_type = pap
        copy_request_to_tunnel = yes
        use_tunneled_reply = yes
        virtual_server = "inner-tunnel_org_a"
    }
}
```

3.3 Repeating the steps for more organizations

Let's say you don't have only the hypotetical org-A in your system but more organizations, in that case you simply have to repeat the steps explained in the previous sections, substituting the occurrences of org-A with the names of the other organizations.

So if you have an organization named ACME Systems, copy the files and substitute the occurrences `org_a` with `acme_systems`.

3.4 Final steps

Once the configurations are ready, you should *restart freeradius* and *then test/troubleshoot/debug your setup*.

3.5 Implementing other EAP scenarios

Implementing other setups like EAP-TLS requires additional development effort.

OpenWISP Controller already supports x509 certificates, so it would be a matter of integrating the django-x509 module into OpenWISP RADIUS and then implement mechanisms for the users to securely download their certificates.

If you're interested in this feature, let us know via the *support channels*.

CHAPTER

FOUR

AVAILABLE SETTINGS

4.1 Admin related settings

These settings control details of the administration interface of openwisp-radius.

4.1.1 OPENWISP_RADIUS_EDITABLE_ACCOUNTING

Default: `False`

Whether `radacct` entries are editable from the django admin or not.

4.1.2 OPENWISP_RADIUS_EDITABLE_POSTAUTH

Default: `False`

Whether `postauth` logs are editable from the django admin or not.

4.1.3 OPENWISP_RADIUS_GROUPCHECK_ADMIN

Default: `False`

Direct editing of group checks items is disabled by default because these can be edited through inline items in the Radius Group admin (Freeradius > Groups).

This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.

If for some reason you need to enable direct editing of group checks you can do so by setting this to `True`.

4.1.4 OPENWISP_RADIUS_GROUPREPLY_ADMIN

Default: `False`

Direct editing of group reply items is disabled by default because these can be edited through inline items in the Radius Group admin (Freeradius > Groups).

This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.

If for some reason you need to enable direct editing of group replies you can do so by setting this to `True`.

4.1.5 OPENWISP_RADIUS_USERGROUP_ADMIN

Default: `False`

Direct editing of user group items (`radusergroup`) is disabled by default because these can be edited through inline items in the User admin (Users and Organizations > Users).

This is done with the aim of simplifying the admin interface and avoid overwhelming users with too many options.

If for some reason you need to enable direct editing of user group items you can do so by setting this to `True`.

4.2 Model related settings

These settings control details of the openwisp-radius model classes.

4.2.1 OPENWISP_RADIUS_DEFAULT_SECRET_FORMAT

Default: `NT-Password`

The default encryption format for storing radius check values.

4.2.2 OPENWISP_RADIUS_DISABLED_SECRET_FORMATS

Default: `[]`

A list of disabled encryption formats, by default all formats are enabled in order to keep backward compatibility with legacy systems.

4.2.3 OPENWISP_RADIUS_BATCH_DEFAULT_PASSWORD_LENGTH

Default: `8`

The default password length of the auto generated passwords while batch addition of users from the csv.

4.2.4 OPENWISP_RADIUS_BATCH_DELETE_EXPIRED

Default: `18`

It is the number of months after which the expired users are deleted.

4.2.5 OPENWISP_RADIUS_BATCH_PDF_TEMPLATE

It is the template used to generate the pdf when users are being generated using the batch add users feature using the prefix.

The value should be the absolute path to the template of the pdf.

4.2.6 OPENWISP_RADIUS_EXTRA_NAS_TYPES

Default: `tuple()`

This setting can be used to add custom NAS types that can be used from the admin interface when managing NAS instances.

For example, you want a custom NAS type called `cisco`, you would add the following to your project `settings.py`:

```
OPENWISP_RADIUS_EXTRA_NAS_TYPES = (
    ('cisco', 'Cisco Router'),
)
```

4.2.7 OPENWISP_RADIUS_FREERADIUS_ALLOWED_HOSTS

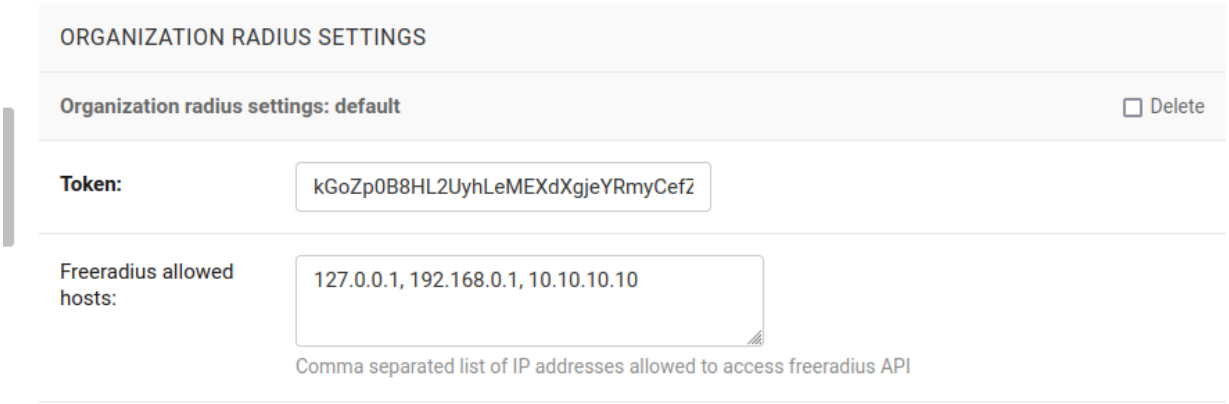
Default: `[]`

List of host IP addresses or subnets allowed to consume the freeradius API endpoints (Authorize, Accounting and Postauth), i.e the value of this option should be the IP address of your freeradius instance. Example: If your freeradius instance is running on the same host machine as OpenWISP, the value should be `127.0.0.1`. Similarly, if your freeradius instance is on a different host in the private network, the value should be the private IP of freeradius host like `192.0.2.50`. If your freeradius is on a public network, please use the public IP of your freeradius instance.

You can use subnets when freeradius is hosted on a variable IP, eg:

- `198.168.0.0/24` to allow the entire LAN.
- `0.0.0.0/0` to allow any address (useful for development / testing).

This value can be overridden per organization in the organization change page. You can skip setting this option if you intend to set it from organization change page for each organization.




```
OPENWISP_RADIUS_FREERADIUS_ALLOWED_HOSTS = ['127.0.0.1', '192.0.2.10', '192.168.0.0/24']
```

If this option and organization change page option are both empty, then all freeradius API requests for the organization will return `403`.

4.2.8 OPENWISP_RADIUS_MAX_CSV_FILE_SIZE

type:	`int`
default:	*5 * 1024 * 1024* (5 MB)

This setting can be used to set the maximum size limit for firmware images, eg:

```
OPENWISP_RADIUS_MAX_CSV_FILE_SIZE = 10 * 1024 * 1024  # 10MB
```

Note: The numeric value represents the size of files in bytes. Setting this to `None` will mean there's no max size.

4.2.9 OPENWISP_RADIUS_CALLED_STATION_IDS

Default: `{}`

This setting allows to specify the parameters to connect to the different OpenVPN management interfaces available for an organization. This setting is used by the *convert_called_station_id* command.

It should contain configuration in following format:

```
OPENWISP_RADIUS_CALLED_STATION_IDS = {
    # Slug of the organization for which settings are being specified
    # In this example 'default'
    '<organization_slug>': {
        'openvpn_config': [
            {
                # Host address of OpenVPN management
                'host': '<host>',
                # Port of OpenVPN management interface. Defaults to 7505 (integer)
                'port': 7506,
                # Password of OpenVPN management interface (optional)
                'password': '<management_interface_password>',
            }
        ],
        # List of CALLED STATION IDs that has to be converted,
        # These look like: 00:27:22:F3:FA:F1:gw1.openwisp.org
        'unconverted_ids': ['<called_station_id>'],
    }
}
```

4.2.10 OPENWISP_RADIUS_CONVERT_CALLED_STATION_ON_CREATE

Default: `False`

If set to `True`, "Called Station ID" of a RADIUS session will be converted (as per configuration defined in *OPENWISP_RADIUS_CALLED_STATION_IDS*) just after the RADIUS session is created.

4.2.11 OPENWISP_RADIUS_OPENVPN_DATETIME_FORMAT

Default: `u'%a %b %d %H:%M:%S %Y'`

Specifies the datetime format of OpenVPN management status parser used by the *convert_called_station_id* command.

4.3 API and user token related settings

These settings control details related to the API and the radius user token.

4.3.1 OPENWISP_RADIUS_API_URLCONF

Default: `None`

Changes the urlconf option of django urls to point the RADIUS API urls to another installed module, example, `myapp.urls` (useful when you have a seperate API instance.)

4.3.2 OPENWISP_RADIUS_API_BASEURL

Default: `/` (points to same server)

If you have a seperate instance of openwisp-radius API on a different domain, you can use this option to change the base of the image download URL, this will enable you to point to your API server's domain, example value: `https://myradius.myapp.com`.

4.3.3 OPENWISP_RADIUS_API

Default: `True`

Indicates whether the REST API of openwisp-radius is enabled or not.

4.3.4 OPENWISP_RADIUS_DISPOSABLE_RADIUS_USER_TOKEN

Default: `True`

Radius user tokens are used for authorizing users.

When this setting is `True` radius user tokens are deleted right after a successful authorization is performed. This reduces the possibility of attackers reusing the access tokens and posing as other users if they manage to intercept it somehow.

4.3.5 OPENWISP_RADIUS_API_AUTHORIZE_REJECT

Default: `False`

Indicates wether the *Authorize API view* will return `{"control:Auth-Type": "Reject"}` or not.

Rejecting an authorization request explicitly will prevent freeradius from attempting to perform authorization with other mechanisms (eg: radius checks, LDAP, etc.).

When set to `False`, if an authorization request fails, the API will respond with `None`, which will allow freeradius to keep attempting to authorize the request with other freeradius modules.

Set this to `True` if you are performing authorization exclusively through the REST API.

4.3.6 OPENWISP_RADIUS_API_ACCOUNTING_AUTO_GROUP

Default: `True`

When this setting is enabled, every accounting instance saved from the API will have its `groupname` attribute automatically filled in. The value filled in will be the `groupname` of the `RadiusUserGroup` of the highest priority among the RadiusUserGroups related to the user with the `username` as in the accounting instance. In the event there is no user in the database corresponding to the `username` in the accounting instance, the failure will be logged with `warning` level but the accounting will be saved as usual.

4.3.7 OPENWISP_RADIUS_ALLOWED_MOBILE_PREFIXES

Default: `[]`

This setting is used to specify a list of international mobile prefixes which should be allowed to register into the system via the *user registration API*.

That is, only users with phone numbers using the specified international prefixes will be allowed to register.

Leaving this unset or setting it to an empty list (`[]`) will effectively allow any international mobile prefix to register (which is the default setting).

For example:

```
OPENWISP_RADIUS_ALLOWED_MOBILE_PREFIXES = ['+44', '+237']
```

Using the setting above will only allow phone numbers from the UK (+44) or Cameroon (+237).

Note: This setting is applicable only for organizations which have *enabled the SMS verification option*.

4.3.8 OPENWISP_RADIUS_OPTIONAL_REGISTRATION_FIELDS

Default:

```
{
    'first_name': 'disabled',
    'last_name': 'disabled',
    'birth_date': 'disabled',
    'location': 'disabled',
}
```

This global setting is used to specify if the optional user fields (`first_name`, `last_name`, `location` and `birth_date`) shall be disabled (hence ignored), allowed or required in the *User Registration API*.

The allowed values are:

- `disabled`: (**default**) the field is disabled.
- `allowed`: the field is allowed but not mandatory.
- `mandatory`: the field is mandatory.

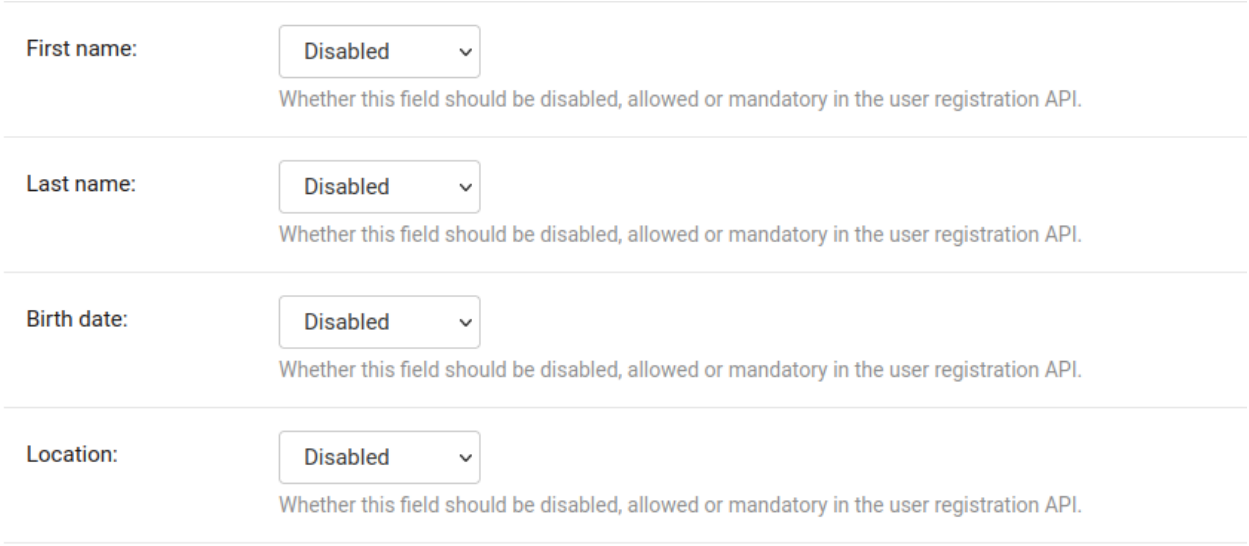
For example:

```
OPENWISP_RADIUS_OPTIONAL_REGISTRATION_FIELDS = {
    'first_name': 'disabled',
    'last_name': 'disabled',
    'birth_date': 'mandatory',
    'location': 'allowed',
}
```

Means:

- `first_name` and `last_name` fields are not required and their values if provided are ignored.
- `location` field is not required but its value will be saved to the database if provided.
- `birth_date` field is required and a `ValidationError` exception is raised if its value is not provided.

The setting for each field can also be overridden at organization level if needed, by going to `Home › Users and Organizations › Organizations > Edit organization` and then scrolling down to `ORGANIZATION RADIUS SETTINGS`.



By default the fields at organization level hold a `NULL` value, which means that the global setting specified in `settings.py` will be used.

4.3.9 OPENWISP_RADIUS_PASSWORD_RESET_URLS

Note: This setting can be overridden for each organization in the organization admin page, the setting implementation is left for backward compatibility but may be deprecated in the future.

Default:

```
{
    'default': 'https://{site}/{organization}/password/reset/confirm/{uid}/{token}'
}
```

A dictionary representing the frontend URLs through which end users can complete the password reset operation.

The frontend could be openwisp-wifi-login-pages or another in-house captive page solution.

Keys of the dictionary must be either UUID of organizations or `default`, which is the fallback URL that will be used in case there's no customized URL for a specific organization.

The password reset URL must contain the "{token}" and "{uid}" placeholders.

The meaning of the variables in the string is the following:

- `{site}`: site domain as defined in the django site framework (defaults to example.com and an be changed through the django admin)
- `{organization}`: organization slug
- `{uid}`: uid of the password reset request
- `{token}`: token of the password reset request

If you're using openwisp-wifi-login-pages, the configuration is fairly simple, in case the nodejs app is installed in the same domain of openwisp-radius, you only have to ensure the domain field in the main Site object is correct, if instead the nodejs app is deployed on a different domain, say `login.wifiservice.com`, the configuration should be simply changed to:

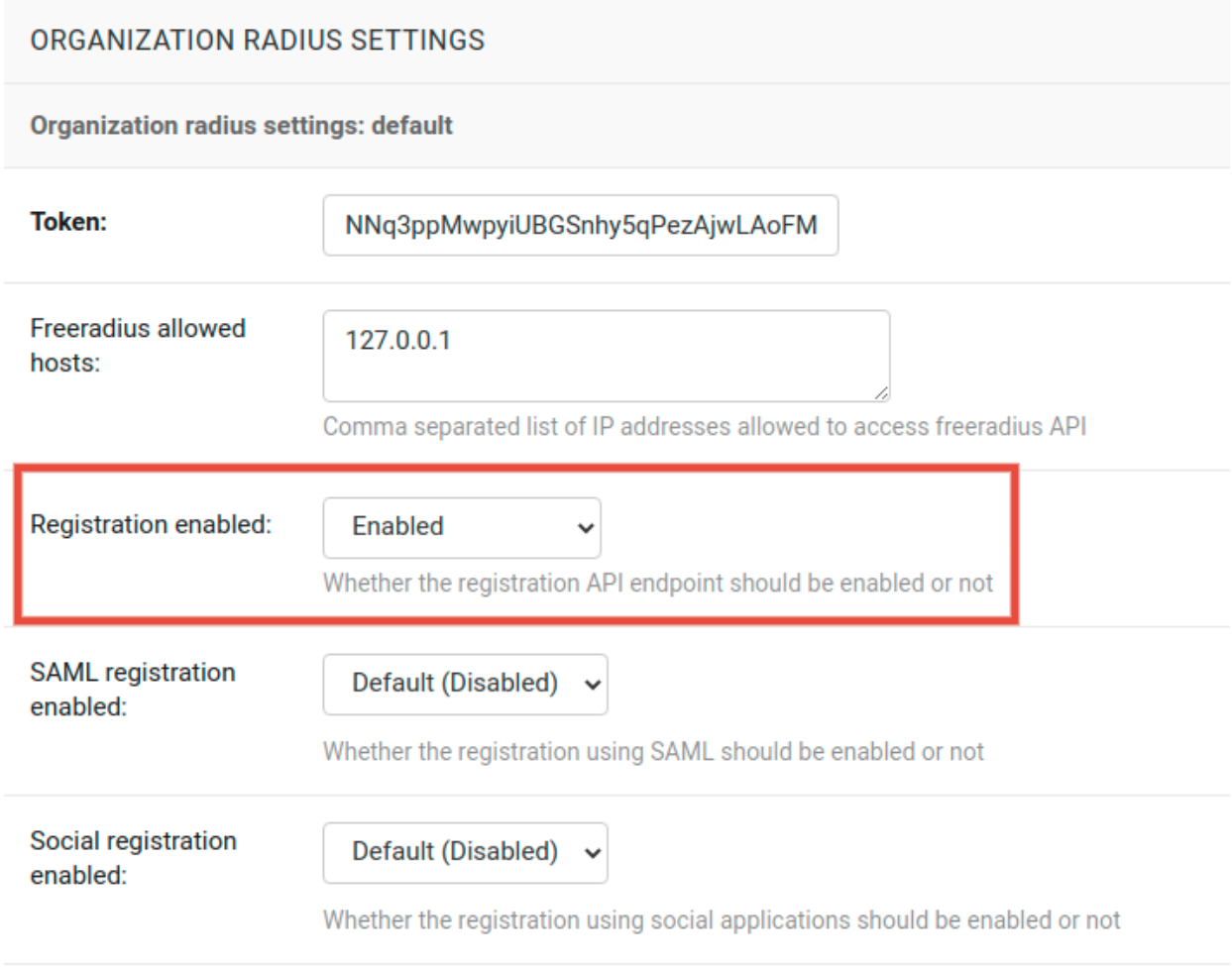
```
{
    'default': 'https://login.wifiservice.com/{organization}/password/reset/confirm/{uid}
↪/{token}'
}
```

4.3.10 OPENWISP_RADIUS_REGISTRATION_API_ENABLED

Default: `True`

Indicates whether the API registration view is enabled or not. When this setting is disabled (i.e. `False`), the registration API view is disabled.

This setting can be overridden in individual organizations via the admin interface, by going to *Organizations* then edit a specific organization and scroll down to *"Organization RADIUS settings"*, as shown in the screenshot below.



Note: We recommend using the override via the admin interface only when there are special organizations which need a different configuration, otherwise, if all the organization use the same configuration, we recommend changing the global setting.

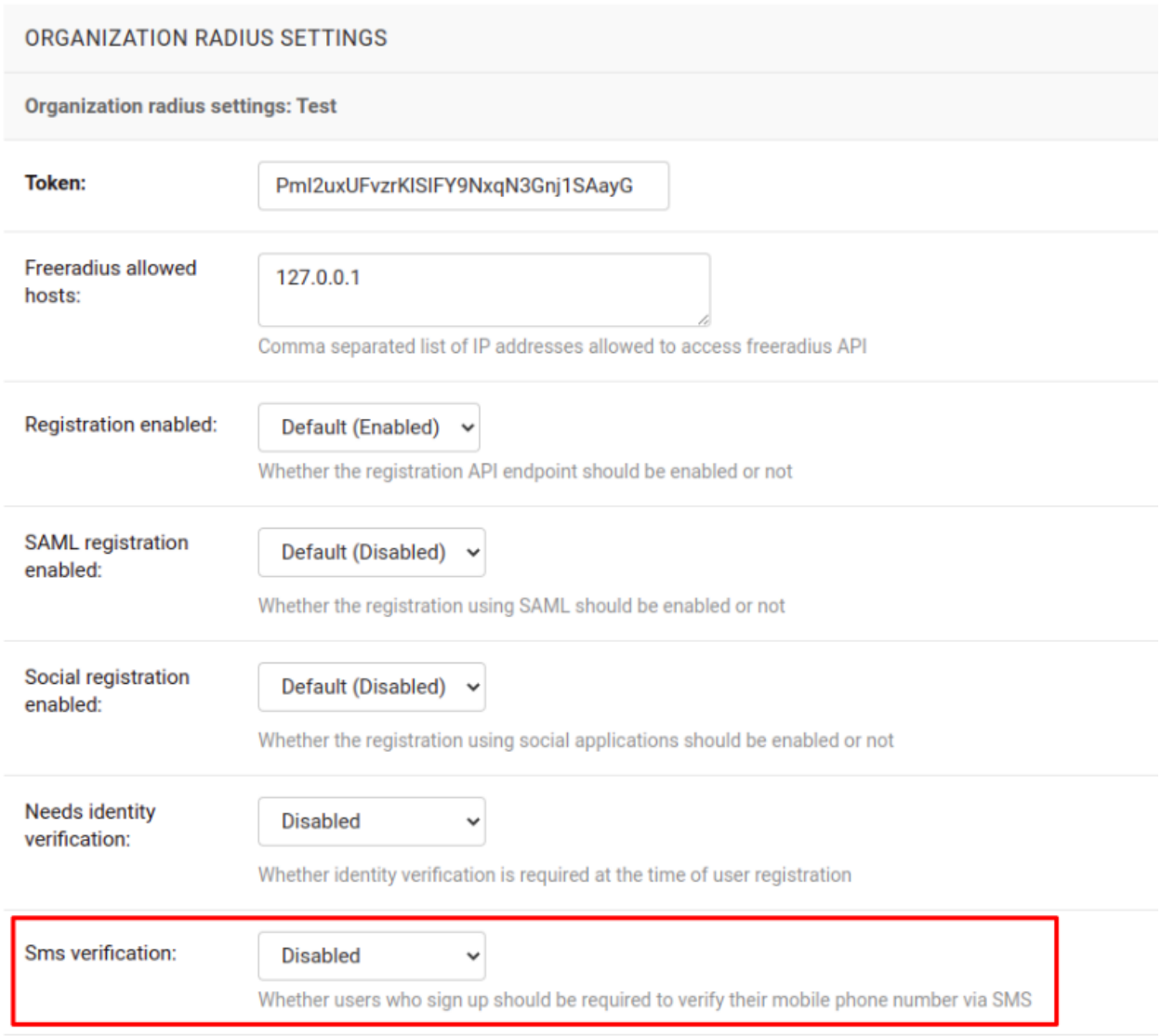
4.3.11 OPENWISP_RADIUS_SMS_VERIFICATION_ENABLED

Default: `False`

Note: If you're looking for instructions on how to configure SMS sending, see *SMS Token Related Settings*.

If *Identity verification is required*, this setting indicates whether users who sign up should be required to verify their mobile phone number via SMS.

This can be overridden for each organization separately via the organization radius settings section of the admin interface.



4.3.12 OPENWISP_RADIUS_NEEDS_IDENTITY_VERIFICATION

Default: `False`

Indicates whether organizations require a user to be verified in order to login. This can be overridden globally or for each organization separately via the admin interface.

If this is enabled, each registered user should be verified using a verification method. The following choices are available by default:

- `''` (empty string): unspecified
- `manual`: manually created
- `email`: Email (No Identity Verification)
- `mobile_phone`: Mobile phone number *verification via SMS*
- `social_login`: *social login feature*

Note: Of the methods listed above, `mobile_phone` is generally accepted as a legal and valid form of indirect identity verification in those countries who require to provide a valid ID document before buying a SIM card.

Organizations which are required by law to identify their users before allowing them to access the network (eg: ISPs) can restrict users to register only through this method and can configure the system to only *allow international mobile prefixes* of countries which require a valid ID document to buy a SIM card.

Disclaimer: these are just suggestions on possible configurations of OpenWISP RADIUS and must not be considered as legal advice.

Adding support for more registration/verification methods

For those who need to implement additional registration and identity verification methods, such as supporting a National ID card, new methods can be added or an existing method can be removed using the `register_registration_method` and `unregister_registration_method` functions respectively.

For example:

```
from openwisp_radius.registration import (
    register_registration_method,
    unregister_registration_method,
)

# Enable registering via national digital ID
register_registration_method('national_id', 'National Digital ID')

# Remove mobile verification method
unregister_registration_method('mobile_phone')
```

Note: Both functions will fail if a specific registration method is already registered or unregistered, unless the keyword argument `fail_loud` is passed as `False` (this useful when working with additional registration methods which are supported by multiple custom modules).

Pass `strong_identity` as `True` to to indicate that users who register using that method have indirectly verified their identity (eg: *SMS verification*, credit card, national ID card, etc).

Warning: If you need to implement a registration method that needs to grant limited internet access to unverified users so they can complete their verification process online on other websites which cannot be predicted and hence cannot be added to the walled garden, you can pass `authorize_unverified=True` to the `register_registration_method` function.

This is needed to implement payment flows in which users insert a specific 3D secure code in the website of their bank. Keep in mind that you should create a specific limited radius group for these unverified users.

Payment flows and credit/debit card verification are fully implemented in **OpenWISP Subscriptions**, a premium module available only to customers of the *commercial support offering of OpenWISP*.

4.4 Email related settings

Emails can be sent to users whose usernames or passwords have been auto-generated. The content of these emails can be customized with the settings explained below.

4.4.1 OPENWISP_RADIUS_BATCH_MAIL_SUBJECT

Default: `Credentials`

It is the subject of the mail to be sent to the users. Eg: `Login Credentials`.

4.4.2 OPENWISP_RADIUS_BATCH_MAIL_MESSAGE

Default: `username: {}, password: {}`

The message should be a string in the format `Your username is {} and password is {}`.

The text could be anything but should have the format string operator `{}` for `.format` operations to work.

4.4.3 OPENWISP_RADIUS_BATCH_MAIL_SENDER

Default: `settings.DEFAULT_FROM_EMAIL`

It is the sender email which is also to be configured in the SMTP settings. The default sender email is a common setting from the Django core settings under `DEFAULT_FROM_EMAIL`. Currently, `DEFAULT_FROM_EMAIL` is set to to `webmaster@localhost`.

4.5 Counter related settings

4.5.1 OPENWISP_RADIUS_COUNTERS

Default: depends on the database backend in use, see *How limits are enforced: counters* to find out what are the default counters enabled.

It's a list of strings, each representing the python path to a counter class.

It may be set to an empty list or tuple to disable the counter feature, eg:

```
OPENWISP_RADIUS_COUNTERS = []
```

If custom counters have been implemented, this setting should be changed to include the new classes, eg:

```
OPENWISP_RADIUS_COUNTERS = [
    # default counters for PostgreSQL, may be removed if not needed
    'openwisp_radius.counters.postgresql.daily_counter.DailyCounter',
    'openwisp_radius.counters.postgresql.daily_traffic_counter.DailyTrafficCounter',
    # custom counters
    'myproject.counters.CustomCounter1',
    'myproject.counters.CustomCounter2',
]
```

4.5.2 OPENWISP_RADIUS_TRAFFIC_COUNTER_CHECK_NAME

Default: `Max-Daily-Session-Traffic`

Used by *DailyTrafficCounter*, it indicates the check attribute which is looked for in the database to find the maximum amount of daily traffic which users having the default `users` radius group assigned can consume.

4.5.3 OPENWISP_RADIUS_TRAFFIC_COUNTER_REPLY_NAME

Default: `ChilliSpot-Max-Total-Octets`

Used by *DailyTrafficCounter*, it indicates the reply attribute which is returned to the NAS to indicate how much remaining traffic users which users having the default `users` radius group assigned can consume.

It should be changed according to the NAS software in use, for example, if using PfSense, this setting should be set to `pfSense-Max-Total-Octets`.

4.6 Social Login related settings

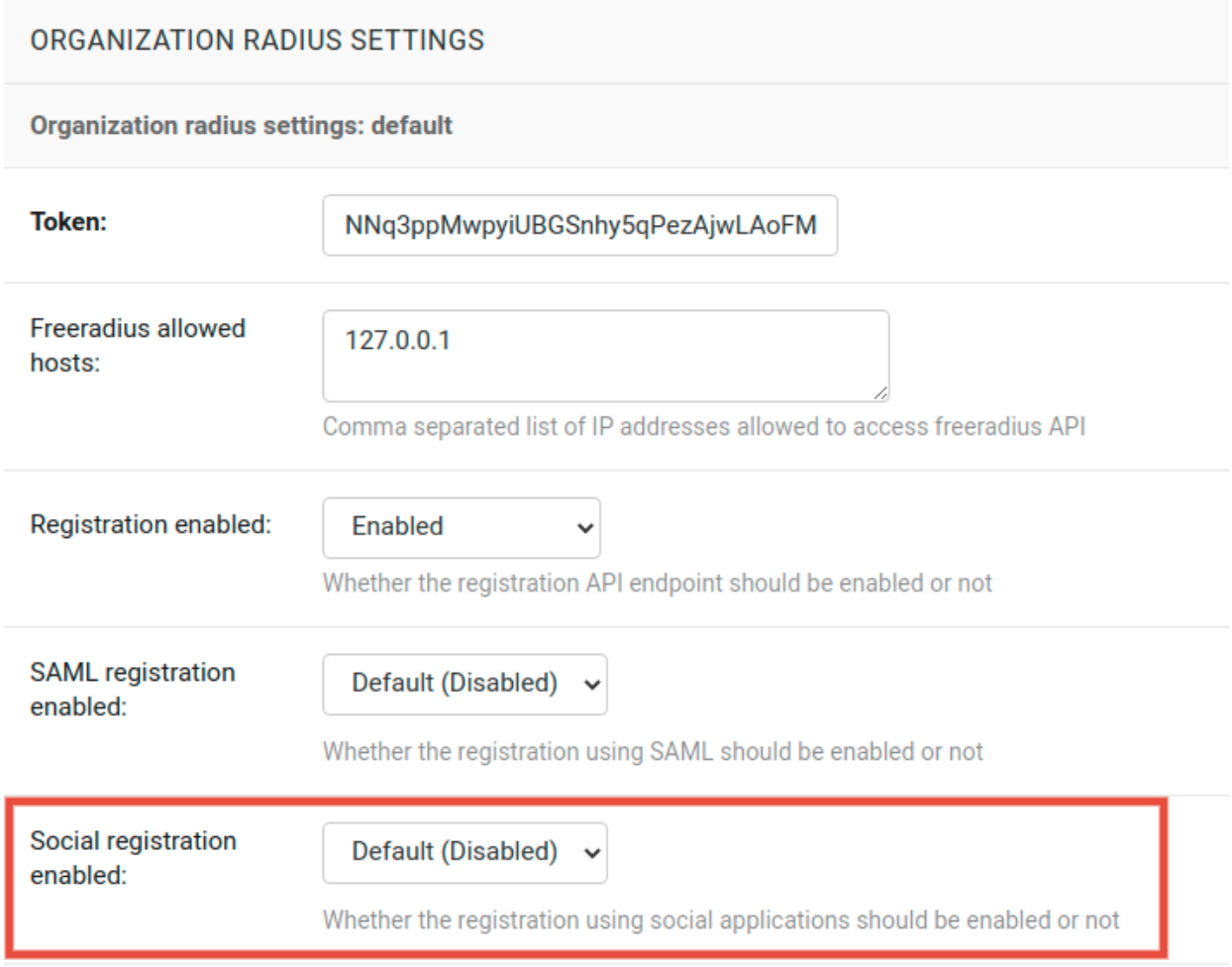
The following settings are related to the *social login feature*.

4.6.1 OPENWISP_RADIUS_SOCIAL_REGISTRATION_ENABLED

Default: `False`

Indicates whether the registration using social applications is enabled or not. When this setting is enabled (i.e. `True`), authentication using social applications is enabled for all organizations.

This setting can be overridden in individual organizations via the admin interface, by going to *Organizations* then edit a specific organization and scroll down to *"Organization RADIUS settings"*, as shown in the screenshot below.



Note: We recommend using the override via the admin interface only when there are special organizations which need a different configuration, otherwise, if all the organization use the same configuration, we recommend changing the global setting.

4.7 SAML related settings

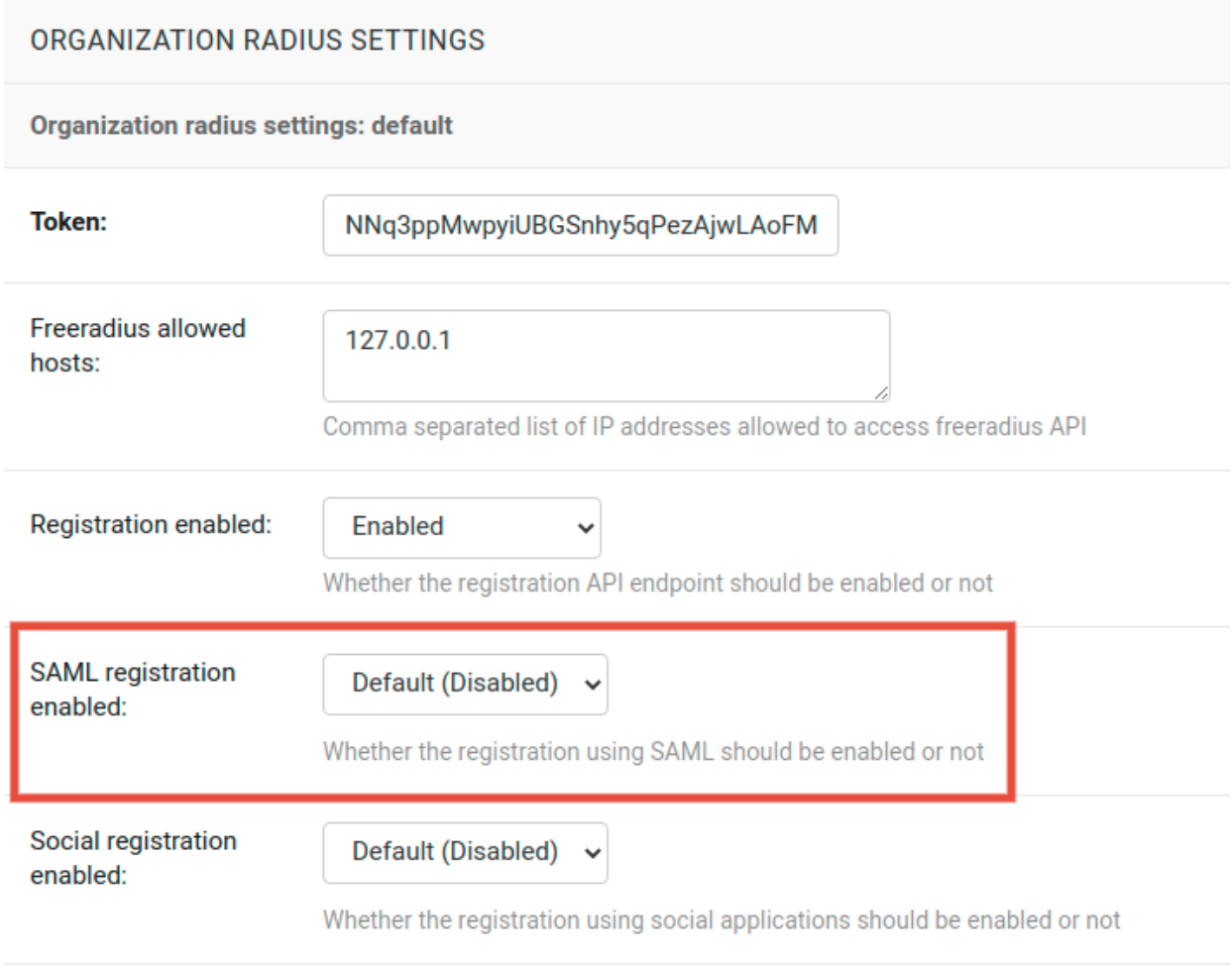
The following settings are related to the *SAML feature*.

4.7.1 OPENWISP_RADIUS_SAML_REGISTRATION_ENABLED

Default: `False`

Indicates whether registration using SAML is enabled or not. When this setting is enabled (i.e. `True`), authentication using SAML is enabled for all organizations.

This setting can be overridden in individual organizations via the admin interface, by going to *Organizations* then edit a specific organization and scroll down to *"Organization RADIUS settings"*, as shown in the screenshot below.



Note: We recommend using the override via the admin interface only when there are special organizations which need a different configuration, otherwise, if all the organization use the same configuration, we recommend changing the global setting.

4.7.2 OPENWISP_RADIUS_SAML_REGISTRATION_METHOD_LABEL

Default: `'Single Sign-On (SAML)'`

Sets the verbose name of SAML registration method.

4.7.3 OPENWISP_RADIUS_SAML_IS_VERIFIED

Default: `False`

Setting this to `True` will automatically flag user accounts created during SAML sign-in as verified users (`RegisteredUser.is_verified=True`).

This is useful when SAML identity providers can be trusted to be legally valid identity verifiers.

4.7.4 OPENWISP_RADIUS_SAML_UPDATES_PRE_EXISTING_USERNAME

Default: `False`

Allows updating username of a registered user with the value received from SAML Identity Provider. Read the *FAQs in SAML integration documentation* for details.

4.8 SMS token related settings

These settings allow to control aspects and limitations of the SMS tokens which are sent to users for the purpose of *verifying their mobile phone number*.

These settings are applicable only when *SMS verification is enabled*.

4.8.1 SENDSMS_BACKEND

This setting takes a python path which points to the django-sendsms backend which will be used by the system to send SMS messages.

The list of supported SMS services can be seen in the source code of the django-sendsms backends. Adding support for other SMS services can be done by subclassing the `BaseSmsBackend` and implement the logic needed to talk to the SMS service.

The value of this setting can point to any class on the python path, so the backend doesn't have to be necessarily shipped in django-sendsms but can be deployed in any other location.

4.8.2 OPENWISP_RADIUS_SMS_TOKEN_DEFAULT_VALIDITY

Default: `30`

For how many minutes the SMS token is valid for.

4.8.3 OPENWISP_RADIUS_SMS_TOKEN_LENGTH

Default: `6`

The length of the SMS token.

4.8.4 OPENWISP_RADIUS_SMS_TOKEN_HASH_ALGORITHM

Default: `'sha256'`

The hashing algorithm used to generate the numeric code.

4.8.5 OPENWISP_RADIUS_SMS_TOKEN_MAX_ATTEMPTS

Default: `5`

The max number of mistakes tolerated during verification, after this amount of mistaken attempts, it won't be possible to verify the token anymore and it will be necessary to request a new one.

4.8.6 OPENWISP_RADIUS_SMS_TOKEN_MAX_USER_DAILY

Default: `5`

The max number of SMS tokens a single user can request within a day.

4.8.7 OPENWISP_RADIUS_SMS_TOKEN_MAX_IP_DAILY

Default: `999`

The max number of tokens which can be requested from the same IP address during the same day.

CHAPTER

FIVE

MANAGEMENT COMMANDS

These management commands are necessary for enabling certain features and for database cleanup.

Example usage:

```
cd tests/
./manage.py <command> <args>
```

In this page we list the management commands currently available in **openwisp-radius**.

5.1 delete_old_radacct

This command deletes RADIUS accounting sessions older than <days>.

```
./manage.py delete_old_radacct <days>
```

For example:

```
./manage.py delete_old_radacct 365
```

5.2 delete_old_postauth

This command deletes RADIUS post-auth logs older than <days>.

```
./manage.py delete_old_postauth <days>
```

For example:

```
./manage.py delete_old_postauth 365
```

5.3 cleanup_stale_radacct

This command closes stale RADIUS sessions that have remained open for the number of specified <days>.

```
./manage.py cleanup_stale_radacct <days>
```

For example:

```
./manage.py cleanup_stale_radacct 15
```

5.4 deactivate_expired_users

Note: *Find out more about this feature in its dedicated page*

This command deactivates expired user accounts which were created temporarily (eg: for en event) and have an expiration date set.

```
./manage.py deactivate_expired_users
```

5.5 delete_old_users

This command deletes users that have expired (and should have been deactivated by `deactivate_expired_users`) for more than the specified <duration_in_months>.

```
./manage.py delete_old_users --older-than-months <duration_in_months>
```

Note that the default duration is set to 18 months.

5.6 delete_unverified_users

This command deletes unverified users that have been registered for more than specified duration and have no associated radius session. This feature is needed to delete users who have registered but never completed the verification process. **Staff users will not be deleted by this management command.**

```
./manage.py delete_unverified_users --older-than-days <duration_in_days>
```

Note that the default duration is set to 1 day.

It is also possible to exclude users that have registered using specified methods. You can specify multiple methods separated by comma(,). Following is an example:

```
./manage.py delete_unverified_users --older-than-days 1 --exclude-methods mobile_phone,
↪email
```

5.7 upgrade_from_django_freeradius

If you are upgrading from django-freeradius to openwisp-radius, there is an easy migration script that will import your freeradius database, sites, social website account users, users & groups to openwisp-radius instance:

```
./manage.py upgrade_from_django_freeradius
```

The management command accepts an argument `--backup`, that you can pass to give the location of the backup files, by default it looks in the `tests/` directory, eg:

```
./manage.py upgrade_from_django_freeradius --backup /home/user/django_freeradius/
```

The management command accepts another argument `--organization`, if you want to import data to a specific organization, you can give its UUID for the same, by default the data is added to the first found organization, eg:

```
./manage.py upgrade_from_django_freeradius --organization 900856da-c89a-412d-8fee-
↪45a9c763ca0b
```

Note: You can follow the tutorial to migrate database from django-freeradius.

Warning: It is not possible to export user credential data for radiusbatch created using prefix, please manually preserve the PDF files if you want to access the data in the future.

5.8 convert_called_station_id

If an installation uses a centralized captive portal, the value of "Called Station ID" of RADIUS Sessions will always show the MAC address of the captive portal instead of the access points.

This command will update the "Called Station ID" to reflect the MAC address of the access points using information from OpenVPN. It requires installing `openvpn_status`, which can be installed using the following command

```
pip install openwisp-radius[openvpn_status]
```

In order to work, this command requires to be configured via the *OPENWISP_RADIUS_CALLED_STATION_IDS* setting.

Use the following command if you want to perform this operation for all RADIUS sessions that meet criteria of `OPENWISP_RADIUS_CALLED_STATION_IDS` setting.

```
./manage.py convert_called_station_id
```

You can also convert the "Called Station ID" of a particular RADIUS session by replacing session's `unique_id` in the following command:

```
./manage.py convert_called_station_id --unique_id=<session_unique_id>
```

Note: If you encounter `ParseError` for datetime data, you can set the datetime format of the parser using *OPENWISP_RADIUS_OPENVPN_DATETIME_FORMAT* setting.

Note: `convert_called_station_id` command will only operate on open RADIUS sessions, i.e. the "stop_time" field is None.

But if you are converting a single RADIUS session, it will operate on it even if the session is closed.

CHAPTER

SIX

IMPORTING USERS

This feature can be used for importing users from a csv file. There are many features included in it such as:

- Importing users in batches: all of the users of a particular csv file would be stored in batches and can be retrieved/ deleted easily using the batch functions.
- Set an expiration date: Expiration date can be set for a batch after which the users would not able to authenticate to the RADIUS Server.
- Autogenerate usernames and passwords: The usernames and passwords are automatically generated if they aren't provided in the csv file. Usernames are generated from the email address whereas passwords are generated randomly and their lengths can be customized.
- Passwords are accepted in both cleartext and hash formats from the CSV.
- Send mails to users whose passwords have been generated automatically.

This operation can be performed via the admin interface, with a management command or via the REST API.

6.1 CSV Format

The CSV shall be of the format:

```
username,password,email,firstname,lastname
```

6.1.1 Imported users with hashed passwords

The hashes are directly stored in the database if they are of the django hash format.

For example, a password myPassword123, hashed using salted SHA1 algorithm, will look like:

```
pbkdf2_sha256$100000$cKdP39chT3pW$2EtVk4Hhm1V65GNfYAA5AHj0uyD60f2CmqumqiB/gRk=
```

So a full CSV line containing that password would be:

```
username,pbkdf2_sha256$100000$cKdP39chT3pW$2EtVk4Hhm1V65GNfYAA5AHj0uyD60f2CmqumqiB/gRk=,
↪email@email.com,firstname,lastname
```

6.1.2 Importing users with clear-text passwords

Clear-text passwords must be flagged with the prefix `cleartext$`.

For example, if we want to use the password `qwerty`, we must use: `cleartext$qwerty`.

6.1.3 Auto-generation of usernames and passwords

Email is the only mandatory field of the CSV file.

Other fields like username and password will be auto-generated if omitted.

Emails will be sent to users whose usernames or passwords have been auto-generated and contents of these emails can be customized too.

Here are some defined settings for doing that:

- *OPENWISP_RADIUS_BATCH_MAIL_SUBJECT*
- *OPENWISP_RADIUS_BATCH_MAIL_MESSAGE*
- *OPENWISP_RADIUS_BATCH_MAIL_SENDER*

6.2 Using the admin interface

Note: The CSV uploaded must follow the *CSV format described above*.

To generate users from the admin interface, go to `Home > Batch user creation operations > Add` (URL: `/admin/openwisp_radius/radiusbatch/add`), set `Strategy` to `Import from CSV`, choose the CSV file to upload and save.

6.3 Management command: `batch_add_users`

This command imports users from a csv file. Usage is as shown below.

```
./manage.py batch_add_users --name <name_of_batch> \
                            --organization=<organization-slug> \
                            --file <filepath> \
                            --expiration <expiration_date> \
                            --password-length <password_length>
```

Note: The expiration and password-length are optional parameters which default to never and 8 respectively.

6.4 REST API: Batch user creation

See *API documentation: Batch user creation*.

CHAPTER

SEVEN

GENERATING USERS

Many a times, a network admin might need to generate temporary users (eg: events).

This feature can be used for generating users by specifying a prefix and the number of users to be generated.

There are many features included in it such as:

- **Generating users in batches**: all of the users of a particular **prefix** would be stored in batches and can be retrieved/deleted easily using the batch functions.
- **Download user credentials in PDF format**: get the usernames and passwords generated outputted into a PDF.
- **Set an expiration date**: an expiration date can be set for a batch after which the users would not able to authenticate to the RADIUS Server.

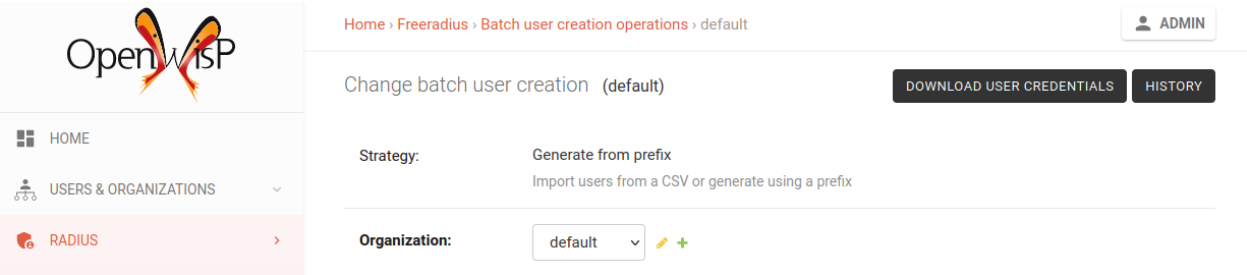
This operation can be performed via the admin interface, with a management command or via the REST API.

Note: Users imported or generated through this form will be flagged as verified if the organization requires identity verification, otherwise the generated users would not be able to log in. If this organization requires identity verification, make sure the identity of the users is verified before giving out the credentials.

7.1 Using the admin interface

To generate users from the admin interface, go to `Home > Batch user creation operations > Add` (URL: `/admin/openwisp_radius/radiusbatch/add`), set `Strategy` to `Generate from prefix`, fill in the remaining fields that are shown after the selection of the strategy and save.

Once the batch object has been created, a PDF containing the user credentials can be downloaded by using the "Download user credentials" button in the upper right corner of the page:



The contents of the PDF is in format of a table of users & their passwords:

Username	Password
sample1	ygJDRWt1
sample2	Ar1l2tZY

Usage Demonstration:

7.2 Management command: `prefix_add_users`

This command generates users whose usernames start with a particular prefix. Usage is as shown below.

```
./manage.py prefix_add_users --name <name_of_batch> \
                             --organization=<organization-slug> \
                             --prefix <prefix> \
                             --n <number_of_users> \
                             --expiration <expiration_date> \
                             --password-length <password_length> \
                             --output <path_to_pdf_file>
```

Note: The expiration, password-length and output are optional parameters. The options expiration and password-length default to never and 8 respectively. If output parameter is not provided, pdf file is not created on the server and can be accessed from the admin interface.

7.3 REST API: Batch user creation

See API documentation: *Batch user creation*.

CHAPTER

EIGHT

ENFORCING SESSION LIMITS

The default freeradius schema does not include a table where groups are stored, but openwisp-radius adds a model called `RadiusGroup` and alters the default freeradius schema to add some optional foreign-keys from other tables like:

- `radgroupcheck`
- `radgroupreply`
- `radusergroup`

These foreign keys make it easier to automate many synchronization and integrity checks between the `RadiusGroup` table and its related tables but they are not strictly mandatory from the database point of view: their value can be `NULL` and their presence and validation is handled at application level, this makes it easy to use existing freeradius databases.

For each group, checks and replies can be specified directly in the edit page of a Radius Group (`admin > groups > add group` or `change group`).

8.1 Default groups

Some groups are created automatically by **openwisp-radius** during the initial migrations:

- `users`: this is the default group which limits users sessions to 3 hours and 300 MB (daily)
- `power-users`: this group does not have any check, therefore users who are members of this group won't be limited in any way

You can customize the checks and the replies of these groups, as well as create new groups according to your needs and preferences.

Note on the default group: keep in mind that the group flagged as default will by automatically assigned to new users, it cannot be deleted nor it can be flagged as non-default: to set another group as default simply check that group as the default one, save and **openwisp-radius** will remove the default flag from the old default group.

8.2 How limits are enforced: counters

In Freeradius, this kind of feature is implemented with the rml_sqlcounter.

The problem with this FreeRADIUS module is that it doesn't know about OpenWISP, so it does not support multi-tenancy. This means that if multiple organizations are using the OpenWISP instance, it's possible that a user may be an end user of multiple organizations and hence have one radius group assigned for each, but the sqlcounter module will not understand the right group to choose when enforcing limits, with the result that the enforcing of limits will not work as expected, unless one FreeRADIUS site with different sqlcounter configurations is created for each organization using the system, which is doable but cumbersome to maintain.

For the reasons explained above, an alternative counter feature has been implemented in the authorize API endpoint of OpenWISP RADIUS.

The default counters available are described below.

8.2.1 DailyCounter

This check is used to limit the amount of time users can use the network every day. It works by checking whether the total session time of a user during a specific day is below the value indicated in the `Max-Daily-Session` group check attribute, sending the remaining session time with a `Session-Timeout` reply message or rejecting the authorization if the limit has been passed.

8.2.2 DailyTrafficCounter

This check is used to limit the amount of traffic users can consume every day. It works by checking whether the total amount of download plus upload octets (bytes consumed) is below the value indicated in the `Max-Daily-Session-Traffic` group check attribute, sending the remaining octets with a reply message or rejecting the authorization if the limit has been passed.

The attributes used for the check and or the reply message are configurable because it can differ from NAS to NAS, see *OPENWISP_RADIUS_TRAFFIC_COUNTER_CHECK_NAME* *OPENWISP_RADIUS_TRAFFIC_COUNTER_REPLY_NAME* for more information.

8.2.3 Database support

The counters described above are available for PostgreSQL, MySQL, SQLite and are enabled by default.

There's a different class of each counter for each database, because the query is executed with raw SQL defined on each class, instead of the classic django-ORM approach which is database agnostic.

It was implemented this way to ensure maximum flexibility and adherence to the FreeRADIUS sqlcounter implementation.

8.2.4 Django Settings

The settings available to control the behavior of counters are described in *Counter related settings*.

8.2.5 Writing custom counter classes

It is possible to write custom counter classes to satisfy any need.

The easiest way is to subclass `openwisp_radius.counters.base.BaseCounter`, then implement at least the following attributes:

- `counter_name`: name of the counter, used internally for debugging;
- `check_name`: attribute name used in the database lookup to the group check table;
- `reply_name`: attribute name sent in the reply message;
- `reset`: reset period, either `daily`, `weekly`, `monthly` or `never`;
- `sql`: the raw SQL query to execute;

- `get_sql_params`: a method which returns a list of the arguments passed to the interpolation of the raw sql query.

Please look at the source code of OpenWISP RADIUS to find out more.

- openwisp_radius.counters.base
- openwisp_radius.counters.postgresql

Once the new class is ready, you will need to add it to *OPENWISP_RADIUS_COUNTERS*.

It is also possible to implement a check class in a completely custom fashion (that is, not inheriting from `BaseCounter`), the only requirements are:

- the class must have a constructor (`__init__` method) identical to the one used in the `BaseCounter` class;
- the class must have a `check` method which doesn't need any required argument and returns the remaining counter value or raises `MaxQuotaReached` if the limit has been reached and the authorization should be rejected.

CHAPTER

NINE

REGISTRATION OF NEW USERS

openwisp-radius uses django-rest-auth which provides registration of new users via REST API so you can implement registration and password reset directly from your captive page.

The registration API endpoint is described in *API: User Registration*.

If you need users to self-register to a public wifi service, we suggest to take a look at openwisp-wifi-login-pages, which is built to work with openwisp-radius.

CHAPTER

TEN

SOCIAL LOGIN

Important: The social login feature is disabled by default.

In order to enable this feature you have to follow the *setup instructions* below and then activate it via *global setting or from the admin interface*.

Social login is supported by generating an additional temporary token right after users perform the social sign-in, the user is then redirected to the captive page with two querystring parameters: `username` and `token`.

The captive page must recognize these two parameters and automatically perform the submit action of the login form: `username` should obviously used for the username field, while `token` should be used for the password field.

The internal REST API of openwisp-radius will recognize the token and authorize the user.

This kind of implementation allows to implement the social login with any captive portal which already supports the RADIUS protocol because it's totally transparent for it, that is, the captive portal doesn't even know the user is signing-in with a social network.

Note: If you're building a public wifi service, we suggest to take a look at openwisp-wifi-login-pages, which is built to work with openwisp-radius.

10.1 Setup

Install `django-allauth`:

```
pip install django-allauth
```

Ensure your `settings.py` looks like the following (we will show how to configure of the facebook social provider):

```
INSTALLED_APPS = [
    # ... other apps ..
    # apps needed for social login
    'rest_framework.authtoken',
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
    # showing facebook as an example
    # to configure social login with other social networks
```

(continues on next page)

(continued from previous page)

```
    # refer to the django-allauth documentation
    'allauth.socialaccount.providers.facebook',
]

SITE_ID = 1

# showing facebook as an example
# to configure social login with other social networks
# refer to the django-allauth documentation
SOCIALACCOUNT_PROVIDERS = {
    'facebook': {
        'METHOD': 'oauth2',
        'SCOPE': ['email', 'public_profile'],
        'AUTH_PARAMS': {'auth_type': 'reauthenticate'},
        'INIT_PARAMS': {'cookie': True},
        'FIELDS': [
            'id',
            'email',
            'name',
            'first_name',
            'last_name',
            'verified',
        ],
        'VERIFIED_EMAIL': True,
    }
}
```

Ensure your main `urls.py` contains the `allauth.urls`:

```
urlpatterns = [
    # .. other urls ...
    path('accounts/', include('allauth.urls')),
]
```

10.2 Configure the social account application

Refer to the django-allauth documentation to find out how to complete the configuration of a sample facebook login app.

10.3 Captive page button example

Following the previous example configuration with facebook, in your captive page you will need an HTML button similar to the ones in the following examples.

This example needs the slug of the organization to assign the new user to the right organization:

```
<a href="https://openwisp2.mywifiproject.com/accounts/facebook/login/?next=%2Fradius
↪%2Fsocial-login%2Fdefault%2F%3Fcp%3Dhttps%3A%2F%2Fcaptivepage.mywifiproject.com%2F
↪%26last%3D"
```

(continues on next page)

(continued from previous page)

```
   class="button">Log in with Facebook
</a>
```

Substitute `openwisp2.mywifiproject.com`, `captivepage.mywifiproject.com` and `default` with the hostname of your openwisp-radius instance, your captive page and the organization slug respectively.

Alternatively, you can take a look at openwisp-wifi-login-pages, which provides buttons for Facebook, Google and Twitter by default.

10.4 Settings

See *social login related settings*.

CHAPTER

ELEVEN

SINGLE SIGN-ON (SAML)

Important: The SAML registration method is disabled by default.

In order to enable this feature you have to follow the *SAML setup instructions* below and then activate it via *global setting or from the admin interface*.

SAML is supported by generating an additional temporary token right after users authenticates via SSO, the user is then redirected to the captive page with 3 querystring parameters:

- `username`
- `token` (REST auth token)
- `login_method=saml`

The captive page must recognize these two parameters, validate the token and automatically perform the submit action of the captive portal login form: `username` should obviously used for the username field, while `token` should be used for the password field.

The third parameter, `login_method=saml`, is needed because it allows the captive page to remember that the user logged in via SAML, because it will need to perform the *SAML logout* later on.

The internal REST API of openwisp-radius will recognize the token and authorize the user.

This kind of implementation allows to support SAML with any captive portal which already supports the RADIUS protocol because it's totally transparent for it, that is, the captive portal doesn't even know the user is signing-in with a SSO.

Note: If you're building a public wifi service, we suggest to take a look at openwisp-wifi-login-pages, which is built to work with openwisp-radius.

11.1 Setup

Install required system dependencies:

```
sudo apt install xmlsec1
```

Install Python dependencies:

```
pip install openwisp-radius[saml]
```

Ensure your `settings.py` looks like the following:

```
INSTALLED_APPS = [
    # ... other apps ..
    # apps needed for SAML login
    'rest_framework.authtoken',
    'django.contrib.sites',
    'allauth',
    'allauth.account',
    'djangosaml2'
]

SITE_ID = 1

# Update AUTHENTICATION_BACKENDS
AUTHENTICATION_BACKENDS = (
    'openwisp_users.backends.UsersAuthenticationBackend',
    'openwisp_radius.saml.backends.OpenwispRadiusSaml2Backend', # <- add for SAML login
)

# Update MIDDLEWARE
MIDDLEWARE = [
    # ... other middlewares ...
    'djangosaml2.middleware.SamlSessionMiddleware',
]
```

Ensure your main `urls.py` contains the `openwisp_users.accounts.urls`:

```
urlpatterns = [
    # .. other urls ...
    path('accounts/', include('openwisp_users.accounts.urls')),
]
```

11.2 Configure the djangosaml2 settings

Refer to the djangosaml2 documentation to find out how to configure required settings for SAML.

11.3 Captive page button example

After successfully configuring SAML settings for your Identity Provider, you will need an HTML button similar to the one in the following example.

This example needs the slug of the organization to assign the new user to the right organization:

```
<a href="https://openwisp2.mywifiproject.com/radius/saml2/login/?RelayState=https://
↪captivepage.mywifiproject.com%3Forg%3Ddefault"
   class="button">
   Log in with SSO
</a>
```

Substitute `openwisp2.mywifiproject.com`, `https://captivepage.mywifiproject.com` and `default` with the hostname of your openwisp-radius instance, your captive page and the organization slug respectively.

Alternatively, you can take a look at openwisp-wifi-login-pages, which provides buttons for Single Sign-On (SAML) by default.

11.4 Logout

When logging out a user which logged in via SAML, the captive page should also call the SAML logout URL: `/radius/saml2/logout/`.

The openwisp-wifi-login-pages app supports this with minimal configuration, refer to the "Configuring SAML Login & Logout" section.

11.5 Settings

See *SAML related settings*.

11.6 FAQs

11.6.1 Preventing change in username of a registered user

The `djangosaml2` library requires configuring `SAML_DJANGO_USER_MAIN_ATTRIBUTE` setting which serves as the primary lookup value for User objects. Whenever a user logs in or registers through the SAML method, a database query is made to check whether such a user already exists. This lookup is done using the value of `SAML_DJANGO_USER_MAIN_ATTRIBUTE` setting. If a match is found, the details of the user are updated with the information received from SAML Identity Provider.

If a user (who has registered on OpenWISP with a different method from SAML) logs into OpenWISP with SAML, then the default behaviour of OpenWISP RADIUS prevents updating username of this user. Because, this operation could render the user's old credentials useless. If you want to update the username in such scenarios with details received from Identity Provider, set *OPENWISP_RADIUS_SAML_UPDATES_PRE_EXISTING_USERNAME* to `True`.

CHAPTER

TWELVE

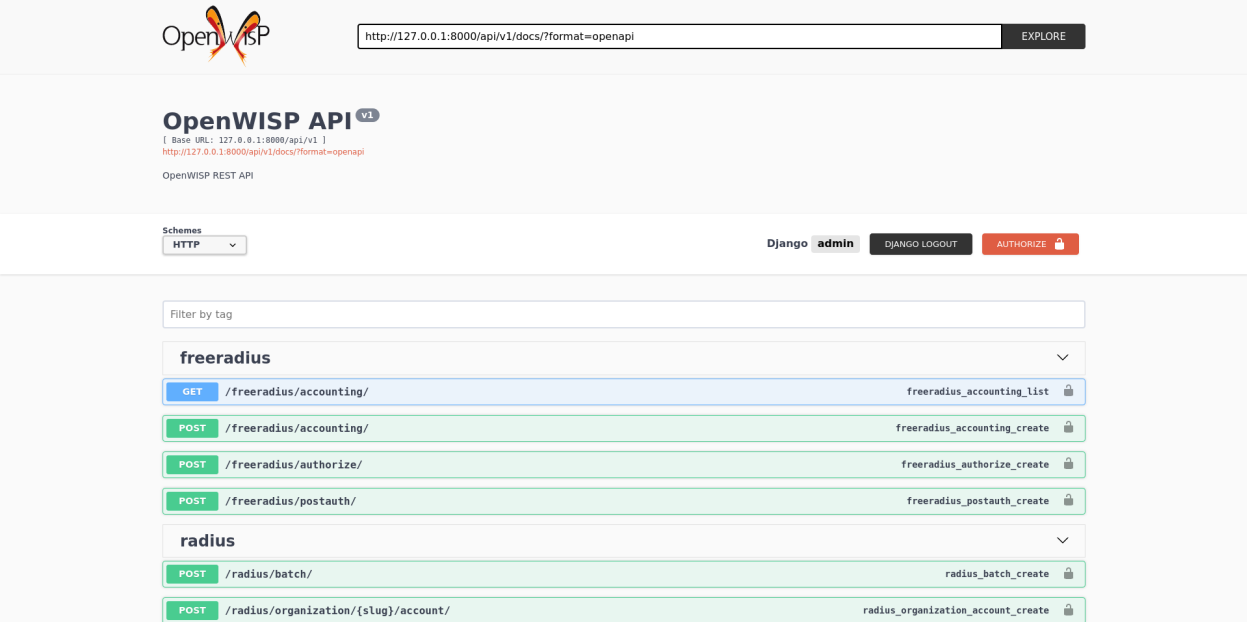
API DOCUMENTATION

Table of Contents:

- *Change phone number*
- *Batch user creation*
- *Batch CSV Download*

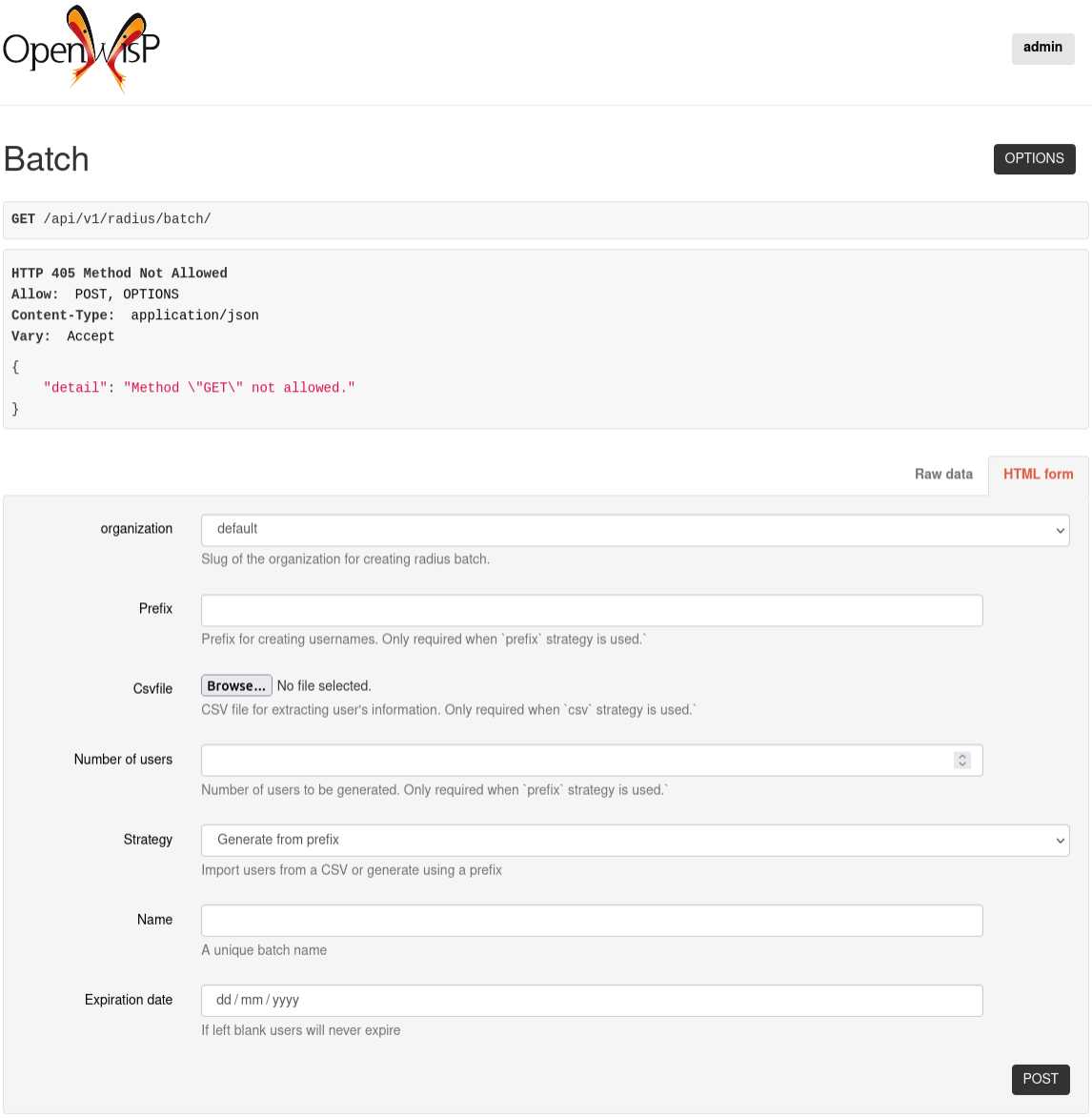
Important: The REST API of openwisp-radius is enabled by default and may be turned off by setting *OPENWISP_RADIUS_API* to `False`.

12.1 Live documentation



A general live API documentation (following the OpenAPI specification) at `/api/v1/docs/`.

12.2 Browsable web interface



Additionally, opening any of the endpoints *listed below* directly in the browser will show the browsable API interface of Django-REST-Framework, which makes it even easier to find out the details of each endpoint.

12.3 FreeRADIUS API Endpoints

The following section is dedicated to API endpoints that are designed to be consumed by FreeRADIUS (*Authorize*, *Post Auth*, *Accounting*).

Important: These endpoints can be consumed only by hosts which have been added to the *freeradius allowed hosts list*.

12.3.1 FreeRADIUS API Authentication

There are 3 different methods with which the FreeRADIUS API endpoints can authenticate incoming requests and understand to which organization these requests belong.

Radius User Token

This method relies on the presence of a special token which was obtained by the user when authenticating via the *Obtain Auth Token View*, this means the user would have to log in through something like a web form first.

The flow works as follows:

1. the user enters credentials in a login form belonging to a specific organization and submits, the credentials are then sent to the *Obtain Auth Token View*;
2. if credentials are correct, a **radius user token** associated to the user and organization is created and returned in the response;
3. the login page or app must then initiate the HTTP request to the web server of the captive portal, (the URL of the form action of the default captive login page) using the radius user token as password, example:

```
curl -X POST http://captive.projcect.com:8005/index.php?zone=myorg \
     -d "auth_user=<username>&auth_pass=<radius_token>"
```

This method is recommended if you are using multiple organizations in the same OpenWISP instance.

Note: By default, `<radius_token>` is valid for authentication for one request only and a new `<radius_token>` needs to be *obtained for each request*. However, if OPENWISP_RADIUS_DISPOSABLE_RADIUS_USER_TOKEN is set to `False`, the `<radius_token>` is valid for authentication as long as freeradius accounting `Stop` request is not sent or the token is not deleted.

Warning: If you are using Radius User token method, keep in mind that one user account can only authenticate with one organization at a time, i.e a single user account cannot consume services from multiple organizations simultaneously.

Bearer token

This other method allows to use the system without the need for a user to obtain a token first, the drawback is that one FreeRADIUS site has to be configured for each organization, the authorization credentials for the specific organization is sent in each request, see *Configure the site* for more information on the FreeRADIUS site configuration.

The (*Organization UUID and Organization RADIUS token*) are sent in the authorization header of the HTTP request in the form of a Bearer token, eg:

```
curl -X POST http://localhost:8000/api/v1/freeradius/authorize/ \
     -H "Authorization: Bearer <org-uuid> <token>" \
     -d "username=<username>&password=<password>"
```

This method is recommended if you are using only one organization and you have no need nor intention of adding more organizations in the future.

Querystring

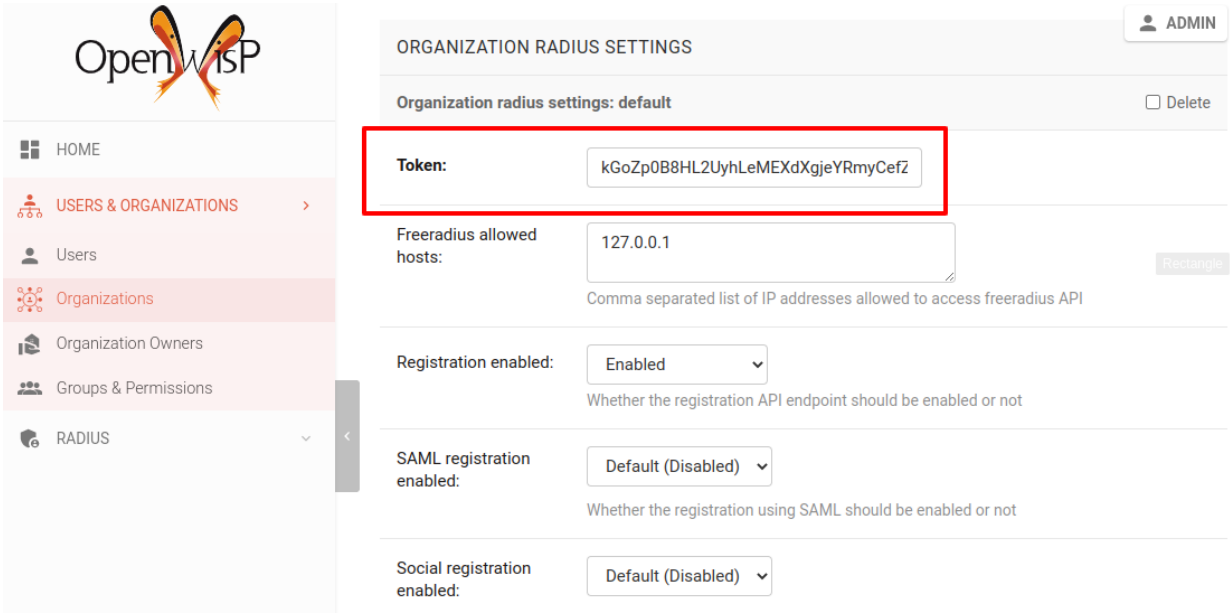
This method is identical to the previous one, but the credentials are sent in querystring parameters, eg:

```
curl -X POST http://localhost:8000/api/v1/freeradius/authorize/?uuid=<org-uuid>&token=
↪<token> \
     -d "username=<username>&password=<password>"
```

This method is not recommended for production usage, it should be used for testing and debugging only (because webservers can include the querystring parameters in their logs).

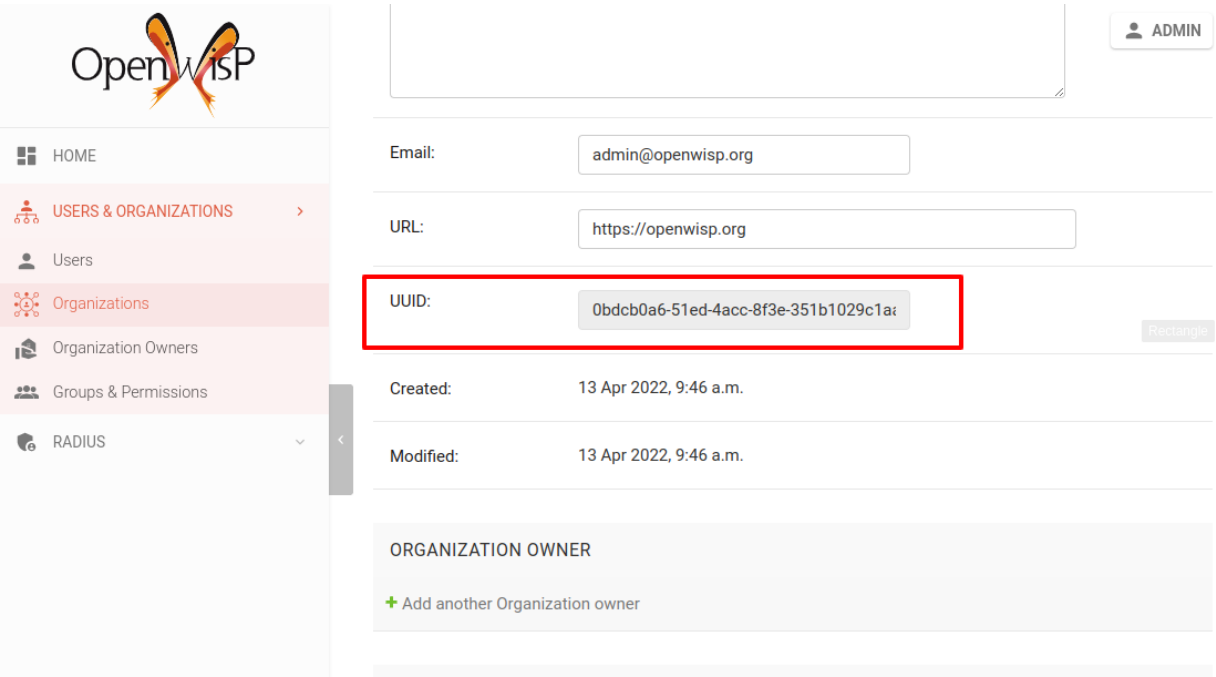
Organization UUID & RADIUS API Token

You can get (and set) the value of the OpenWISP RADIUS API token in the organization configuration page on the OpenWISP dashboard (select your organization in `/admin/openwisp_users/organization/`):



Note: It is highly recommended that you use a hard to guess value, longer than 15 characters containing both letters and numbers. Eg: `165f9a790787fc38e5cc12c1640db2300648d9a2`.

You will also need the UUID of your organization from the organization change page (select your organization in `/admin/openwisp_users/organization/`):



Requests authorizing with *bearer-token* or *querystring* method **must** contain organization UUID & token. If the tokens are missing or invalid, the request will receive a `403` HTTP error.

For information on how to configure FreeRADIUS to send the bearer tokens, see *Configure the site*.

12.3.2 API Throttling

To override the default API throttling settings, add the following to your `settings.py` file:

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [
        'rest_framework.throttling.ScopedRateThrottle',
    ],
    'DEFAULT_THROTTLE_RATES': {
        # None by default
        'authorize': None,
        'postauth': None,
        'accounting': None,
        'obtain_auth_token': None,
        'validate_auth_token': None,
        'create_phone_token': None,
        'validate_phone_token': None,
        # Relaxed throttling Policy
        'others': '400/hour',
```

(continues on next page)

(continued from previous page)

```
    },
}
```

The rate descriptions used in `DEFAULT_THROTTLE_RATES` may include `second`, `minute`, `hour` or `day` as the throttle period, setting it to `None` will result in no throttling.

12.3.3 List of Endpoints

Authorize

Use by FreeRADIUS to perform the `authorization` phase.

It's triggered when a user submits the form to login into the captive portal. The captive portal has to be configured to send the password to freeradius in clear text (will be encrypted with the freeradius shared secret, can be tunneled via TLS for increased security if needed).

FreeRADIUS in turn will send the username and password via HTTPs to this endpoint.

Responds to only **POST**.

```
/api/v1/freeradius/authorize/
```

Example:

```
POST /api/v1/freeradius/authorize/ HTTP/1.1 username=testuser&password=testpassword
```

Param	Description
username	Username for the given user
password	Password for the given user

If the authorization is successful, the API will return all group replies related to the group with highest priority assigned to the user.

If the authorization is unsuccessful, the response body can either be empty or it can contain an explicit rejection, depending on how the *OPENWISP_RADIUS_API_AUTHORIZE_REJECT* setting is configured.

Post Auth

API endpoint designed to be used by FreeRADIUS `postauth`.

Responds only to **POST**.

```
/api/v1/freeradius/postauth/
```

Param	Description
username	Username
password	Password (*)
reply	Radius reply received by freeradius
called_station_id	Called Station ID
calling_station_id	Calling Station ID

(*): the `password` is stored only on unsuccessful authorizations.

Returns an empty response body in order to instruct FreeRADIUS to avoid processing the response body.

Accounting

```
/api/v1/freeradius/accounting/
```

GET

Returns a list of accounting objects

```
GET /api/v1/freeradius/accounting/
```


```
[
  {
      "called_station_id": "00-27-22-F3-FA-F1:hostname",
      "nas_port_type": "Async",
      "groupname": null,
      "id": 1,
      "realm": "",
      "terminate_cause": "User_Request",
      "nas_ip_address": "172.16.64.91",
      "authentication": "RADIUS",
      "stop_time": null,
      "nas_port_id": "1",
      "service_type": "Login-User",
      "username": "admin",
      "update_time": null,
      "connection_info_stop": null,
      "start_time": "2018-03-10T14:44:17.234035+01:00",
      "output_octets": 1513075509,
      "calling_station_id": "5c:7d:c1:72:a7:3b",
      "input_octets": 9900909,
      "interval": null,
      "session_time": 261,
      "session_id": "35000006",
      "connection_info_start": null,
      "framed_protocol": "test",
      "framed_ip_address": "127.0.0.1",
      "unique_id": "75058e50"
  }
]
```

POST

Add or update accounting information (start, interim-update, stop); does not return any JSON response so that freeradius will avoid processing the response without generating warnings

Param	Description
session_id	Session ID
unique_id	Accounting unique ID
username	Username
groupname	Group name
realm	Realm
nas_ip_address	NAS IP address
nas_port_id	NAS port ID
nas_port_type	NAS port type
start_time	Start time
update_time	Update time
stop_time	Stop time
interval	Interval
session_time	Session Time
authentication	Authentication
connection_info_start	Connection Info Start
connection_info_stop	Connection Info Stop
input_octets	Input Octets
output_octets	Output Octets
called_station_id	Called station ID
calling_station_id	Calling station ID
terminate_cause	Termination Cause
service_type	Service Type
framed_protocol	Framed protocol
framed_ip_address	framed IP address

Pagination

Pagination is provided using a Link header pagination. Check here for more information about traversing with pagination.

```
{
  ....
  ....
  link: <http://testserver/api/v1/freeradius/accounting/?page=2&page_size=1>; rel=\"next\
↪",
        <http://testserver/api/v1/freeradius/accounting/?page=3&page_size=1>; rel=\"last\
↪"
  ....
  ....
}
```

Note: Default page size is 10, which can be overridden using the *page_size* parameter.

Filters

The JSON objects returned using the GET endpoint can be filtered/queried using specific parameters.

Filter Parameters	Description
username	Username
called_station_id	Called Station ID
calling_station_id	Calling Station ID
start_time	Start time (greater or equal to)
stop_time	Stop time (less or equal to)
is_open	If stop_time is null

12.4 User API Endpoints

These API endpoints are designed to be used by users (eg: creating an account, changing their password, obtaining access tokens, validating their phone number, etc.).

Note: The API endpoints described below do not require the *Organization API Token* described in the beginning of this document.

Some endpoints require the sending of the user API access token sent in the form of a "Bearer Token", example:

```
curl -H "Authorization: Bearer <user-token>" \
     'http://localhost:8000/api/v1/radius/organization/default/account/session/'
```

12.4.1 List of Endpoints

User Registration

Important: This endpoint is enabled by default but can be disabled either via a *global setting or from the admin interface*.

```
/api/v1/radius/organization/<organization-slug>/account/
```

Responds only to **POST**.

Parameters:

Param	Description
username	string
phone_number	string (*)
email	string
password1	string
password2	string
first_name	string (**)
last_name	string (**)
birth_date	string (**)
location	string (**)
method	string (***)

(*) `phone_number` is required only when the organization has enabled *SMS verification in its "Organization RADIUS Settings"*.

(**) `first_name`, `last_name`, `birth_date` and `location` are optional fields which are disabled by default to make the registration simple, but can be *enabled through configuration*.

(**) `method` must be one of the available *registration/verification methods*; if identity verification is disabled for a particular org, an empty string will be acceptable.

Registering to Multiple Organizations

An **HTTP 409** response will be returned if an existing user tries to register on a URL of a different organization (because the account already exists). The response will contain a list of organizations with which the user has already registered to the system which may be shown to the user in the UI. E.g.:

```
{
    "details": "A user like the one being registered already exists.",
    "organizations":[
        {"slug":"default","name":"default"}
    ]
}
```

The existing user can register with a new organization using the *login endpoint*. The user will also get membership of the new organization only if the organization has *user registration enabled*.

Reset password

This is the classic "password forgotten recovery feature" which sends a reset password token to the email of the user.

```
/api/v1/radius/organization/<organization-slug>/account/password/reset/
```

Responds only to **POST**.

Parameters:

Param	Description
input	string that can be an email, phone_number or username.

Confirm reset password

Allows users to confirm their reset password after having it requested via the *Reset password* endpoint.

```
/api/v1/radius/organization/<organization-slug>/account/password/reset/confirm/
```

Responds only to **POST**.

Parameters:

Param	Description
new_password1	string
new_password2	string
uid	string
token	string

Change password

Requires the user auth token (Bearer Token).

Allows users to change their password after using the *Reset password* endpoint.

```
/api/v1/radius/organization/<organization-slug>/account/password/change/
```

Responds only to **POST**.

Parameters:

Param	Description
current_password	string
new_password	string
confirm_password	string

Login (Obtain User Auth Token)

```
/api/v1/radius/organization/<organization-slug>/account/token/
```

Responds only to **POST**.

Returns:

- `radius_user_token`: the user radius token, which can be used to authenticate the user in the captive portal by sending it in place of the user password (it will be passed to freeradius which in turn will send it to the *authorize API endpoint* which will recognize the token as the user passsword)
- `key`: the user API access token, which will be needed to authenticate the user to eventual subsequent API requests (eg: change password)
- `is_active` if it's `false` it means the user has been banned
- `is_verified` when identity verification is enabled, it indicates whether the user has completed an indirect identity verification process like confirming their mobile phone number
- `method` registration/verification method used by the user to register, eg: `mobile_phone`, `social_login`, etc.
- `username`

- `email`
- `phone_number`
- `first_name`
- `last_name`
- `birth_date`
- `location`

If the user account is inactive or unverified the endpoint will send the data anyway but using the HTTP status code 401, this way consumers can recognize these users and trigger the appropriate response needed (eg: reject them or initiate account verification).

If an existing user account tries to authenticate to an organization of which they're not member of, then they would be automatically added as members (if registration is enabled for that org). Please refer to *"Registering to Multiple Organizations"*.

This endpoint updates the user language preference field according to the `Accept-Language` HTTP header.

Parameters:

Param	Description
username	string
password	string

Validate user auth token

Used to check whether the auth token of a user is valid or not.

Return also the radius user token and username in the response.

```
/api/v1/radius/organization/<organization-slug>/account/token/validate/
```

Responds only to **POST**.

Parameters:

Param	Description
token	the rest auth token to validate

The user information is returned in the response (similarly to *Obtain User Auth Token*), along with the following additional parameter:

- `response_code`: string indicating whether the result is successful or not, to be used for translation.

This endpoint updates the user language preference field according to the `Accept-Language` HTTP header.

User Radius Sessions

Requires the user auth token (Bearer Token).

Returns the radius sessions of the logged-in user and the organization specified in the URL.

```
/api/v1/radius/organization/<organization-slug>/account/session/
```

Responds only to **GET**.

Create SMS token

Note: This API endpoint will work only if the organization has enabled *SMS verification*.

Requires the user auth token (Bearer Token).

Used for SMS verification, sends a code via SMS to the phone number of the user.

```
/api/v1/radius/organization/<organization-slug>/account/phone/token/
```

Responds only to **POST**.

No parameters required.

Verify/Validate SMS token

Note: This API endpoint will work only if the organization has enabled *SMS verification*.

Requires the user auth token (Bearer Token).

Used for SMS verification, allows users to validate the code they receive via SMS.

```
/api/v1/radius/organization/<organization-slug>/account/phone/verify/
```

Responds only to **POST**.

Parameters:

Param	Description
code	string

Change phone number

Note: This API endpoint will work only if the organization has enabled *SMS verification*.

Requires the user auth token (Bearer Token).

Allows users to change their phone number, will flag the user as inactive and send them a verification code via SMS. The phone number of the user is updated only after this verification code has been *validated*.

```
/api/v1/radius/organization/<organization-slug>/account/phone/change/
```

Responds only to **POST**.

Parameters:

Param	Description
phone_number	string

Batch user creation

This API endpoint allows to use the features described in *Importing users* and *Generating users*.

```
/api/v1/radius/batch/
```

Note: This API endpoint allows to use the features described in *Importing users* and *Generating users*.

Responds only to **POST**, used to save a `RadiusBatch` instance.

It is possible to generate the users of the `RadiusBatch` with two different strategies: csv or prefix.

The csv method needs the following parameters:

Param	Description
name	Name of the operation
strategy	csv
csvfile	file with the users
expiration_date	date of expiration of the users
organization_slug	slug of organization of the users

These others are for the prefix method:

Param	Description
name	name of the operation
strategy	prefix
prefix	prefix for the generation of users
number_of_users	number of users
expiration_date	date of expiration of the users
organization_slug	slug of organization of the users

When using this strategy, in the response you can find the field `user_credentials` containing the list of users created (example: `[['username', 'password'], ['sample_user', 'BBuOb5sN']]`) and the field `pdf_link` which can be used to download a PDF file containing the user credentials.

Batch CSV Download

```
/api/v1/radius/organization/<organization-slug>/batch/<id>/csv/<filename>
```

Responds only to **GET**.

Parameters:

Param	Description
slug	string
id	string
filename	string

CHAPTER

THIRTEEN

SIGNALS

13.1 radius_accounting_success

Path: `openwisp_radius.signals.radius_accounting_success`

Arguments:

- `sender` : `AccountingView`
- `accounting_data` (`dict`): accounting information
- `view`: instance of `AccountingView`

This signal is emitted every time the accounting REST API endpoint completes successfully, just before the response is returned.

The `view` argument can also be used to access the `request` object i.e. `view.request`.

CHAPTER

FOURTEEN

EXTENDING OPENWISP-RADIUS

One of the core values of the OpenWISP project is Software Reusability, for this reason *openwisp-radius* provides a set of base classes which can be imported, extended and reused to create derivative apps.

In order to implement your custom version of *openwisp-radius*, you need to perform the steps described in this section.

When in doubt, the code in the test project and the sample app will serve you as source of truth: just replicate and adapt that code to get a basic derivative of *openwisp-radius* working.

If you want to add new users fields, please follow the tutorial to extend the openwisp-users. As an example, we have extended *openwisp-users* to *sample_users* app and added a field `social_security_number` in the sample_users/models.py.

Note: **Premise**: if you plan on using a customized version of this module, we suggest to start with it since the beginning, because migrating your data from the default module to your extended version may be time consuming.

14.1 1. Initialize your custom module

The first thing you need to do is to create a new django app which will contain your custom version of *openwisp-radius*.

A django app is nothing more than a python package (a directory of python scripts), in the following examples we'll call this django app `myradius`, but you can name it how you want:

```
django-admin startapp myradius
```

Keep in mind that the command mentioned above must be called from a directory which is available in your PYTHON_PATH so that you can then import the result into your project.

Now you need to add `myradius` to `INSTALLED_APPS` in your `settings.py`, ensuring also that `openwisp_radius` has been removed:

```
import os

INSTALLED_APPS = [
    # ... other apps ...
    # openwisp admin theme
    'openwisp_utils.admin_theme',
    # all-auth
    'django.contrib.sites',
    'allauth',
    'allauth.account',
```

(continues on next page)

(continued from previous page)

```
    'allauth.socialaccount',
    # admin
    'django.contrib.admin',
    # rest framework
    'rest_framework',
    'django_filters',
    # registration
    'rest_framework.authtoken',
    'dj_rest_auth',
    'dj_rest_auth.registration',
    # social login
    'allauth.socialaccount.providers.facebook',  # optional, can be removed if social␣
↪login is not needed
    'allauth.socialaccount.providers.google',  # optional, can be removed if social␣
↪login is not needed
    # SAML login
    'djangosaml2',  # optional, can be removed if SAML login is not needed
    # openwisp
    # 'myradius', <-- replace with your app-name here
    'openwisp_users',
    'private_storage',
    'drf_yasg'
]

SITE_ID = 1
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
PRIVATE_STORAGE_ROOT = os.path.join(MEDIA_ROOT, 'private')

AUTHENTICATION_BACKENDS = (
    'openwisp_users.backends.UsersAuthenticationBackend',
    'openwisp_radius.saml.backends.OpenwispRadiusSaml2Backend', # optional, can be␣
↪removed if SAML login is not needed
)
```

Important: Remember to include your radius app's name before proceeding.

Note: For more information about how to work with django projects and django apps, please refer to the django documentation.

14.2 2. Install openwisp-radius

Install (and add to the requirement of your project) openwisp-radius:

```
pip install openwisp-radius
```

Note: Use `pip install openwisp-radius[saml]` if you intend to use *Single Sign-On (SAML)* feature.

14.3 3. Add EXTENDED_APPS

Add the following to your `settings.py`:

```
EXTENDED_APPS = ('openwisp_radius',)
```

14.4 4. Add openwisp_utils.staticfiles.DependencyFinder

Add `openwisp_utils.staticfiles.DependencyFinder` to `STATICFILES_FINDERS` in your `settings.py`:

```
STATICFILES_FINDERS = [
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
    'openwisp_utils.staticfiles.DependencyFinder',
]
```

14.5 5. Add openwisp_utils.loaders.DependencyLoader

Add `openwisp_utils.loaders.DependencyLoader` to `TEMPLATES` in your `settings.py`, but ensure it comes before `django.template.loaders.app_directories.Loader`:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'OPTIONS': {
            'loaders': [
                'django.template.loaders.filesystem.Loader',
                'openwisp_utils.loaders.DependencyLoader',
                'django.template.loaders.app_directories.Loader',
            ],
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
```

(continues on next page)

(continued from previous page)

```
    }
]
```

14.6 6. Inherit the AppConfig class

Please refer to the following files in the sample app of the test project:

- sample_radius/__init__.py
- sample_radius/apps.py

You have to replicate and adapt that code in your project.

Note: For more information regarding the concept of `AppConfig` please refer to the "Applications" section in the django documentation.

14.7 7. Create your custom models

For the purpose of showing an example, we added a simple `details` field to the models of the sample app in the test project.

You can add fields in a similar way in your `models.py` file.

Note: For doubts regarding how to use, extend or develop models please refer to the "Models" section in the django documentation.

14.8 8. Add swapper configurations

Once you have created the models, add the following to your `settings.py`:

```
# Setting models for swapper module
OPENWISP_RADIUS_RADIUSREPLY_MODEL = 'myradius.RadiusReply'
OPENWISP_RADIUS_RADIUSGROUPREPLY_MODEL = 'myradius.RadiusGroupReply'
OPENWISP_RADIUS_RADIUSCHECK_MODEL = 'myradius.RadiusCheck'
OPENWISP_RADIUS_RADIUSGROUPCHECK_MODEL = 'myradius.RadiusGroupCheck'
OPENWISP_RADIUS_RADIUSACCOUNTING_MODEL = 'myradius.RadiusAccounting'
OPENWISP_RADIUS_NAS_MODEL = 'myradius.Nas'
OPENWISP_RADIUS_RADIUSUSERGROUP_MODEL = 'myradius.RadiusUserGroup'
OPENWISP_RADIUS_RADIUSPOSTAUTH_MODEL = 'myradius.RadiusPostAuth'
OPENWISP_RADIUS_RADIUSBATCH_MODEL = 'myradius.RadiusBatch'
OPENWISP_RADIUS_RADIUSGROUP_MODEL = 'myradius.RadiusGroup'
OPENWISP_RADIUS_RADIUSTOKEN_MODEL = 'myradius.RadiusToken'
OPENWISP_RADIUS_PHONETOKEN_MODEL = 'myradius.PhoneToken'
OPENWISP_RADIUS_ORGANIZATIONRADIUSSETTINGS_MODEL = 'myradius.OrganizationRadiusSettings'
OPENWISP_RADIUS_REGISTEREDUSER_MODEL = 'myradius.RegisteredUser'
```

(continues on next page)

(continued from previous page)

```
# You will need to change AUTH_USER_MODEL if you are extending openwisp_users
AUTH_USER_MODEL = 'openwisp_users.User'
```

Substitute `myradius` with the name you chose in step 1.

14.9 9. Create database migrations

Copy the migration files from the sample_radius's migration folder.

Now, create database migrations as per your custom application's requirements:

```
./manage.py makemigrations
```

If you are starting with a fresh database, you can apply the migrations:

```
./manage.py migrate
```

However, if you want *migrate an existing freeradius database please read the guide in the setup*.

Note: For more information, refer to the "Migrations" section in the django documentation.

14.10 10. Create the admin

Refer to the admin.py file of the sample app.

To introduce changes to the admin, you can do it in two main ways which are described below.

Note: For more information regarding how the django admin works, or how it can be customized, please refer to "The django admin site" section in the django documentation.

14.10.1 1. Monkey patching

If the changes you need to add are relatively small, you can resort to monkey patching.

For example:

```
from openwisp_radius.admin import (
    RadiusCheckAdmin,
    RadiusReplyAdmin,
    RadiusAccountingAdmin,
    NasAdmin,
    RadiusGroupAdmin,
    RadiusUserGroupAdmin,
    RadiusGroupCheckAdmin,
    RadiusGroupReplyAdmin,
    RadiusPostAuthAdmin,
    RadiusBatchAdmin,
```

(continues on next page)

(continued from previous page)

```
)
# NasAdmin.fields += ['example_field'] <-- Monkey patching changes example
```

14.10.2 2. Inheriting admin classes

If you need to introduce significant changes and/or you don't want to resort to monkey patching, you can proceed as follows:

```
from django.contrib import admin
from openwisp_radius.admin import (
    RadiusCheckAdmin as BaseRadiusCheckAdmin,
    RadiusReplyAdmin as BaseRadiusReplyAdmin,
    RadiusAccountingAdmin as BaseRadiusAccountingAdmin,
    NasAdmin as BaseNasAdmin,
    RadiusGroupAdmin as BaseRadiusGroupAdmin,
    RadiusUserGroupAdmin as BaseRadiusUserGroupAdmin,
    RadiusGroupCheckAdmin as BaseRadiusGroupCheckAdmin,
    RadiusGroupReplyAdmin as BaseRadiusGroupReplyAdmin,
    RadiusPostAuthAdmin as BaseRadiusPostAuthAdmin,
    RadiusBatchAdmin as BaseRadiusBatchAdmin,
)
from swapper import load_model
Nas = load_model('openwisp_radius', 'Nas')
RadiusAccounting = load_model('openwisp_radius', 'RadiusAccounting')
RadiusBatch = load_model('openwisp_radius', 'RadiusBatch')
RadiusCheck = load_model('openwisp_radius', 'RadiusCheck')
RadiusGroup = load_model('openwisp_radius', 'RadiusGroup')
RadiusPostAuth = load_model('openwisp_radius', 'RadiusPostAuth')
RadiusReply = load_model('openwisp_radius', 'RadiusReply')
PhoneToken = load_model('openwisp_radius', 'PhoneToken')
RadiusGroupCheck = load_model('openwisp_radius', 'RadiusGroupCheck')
RadiusGroupReply = load_model('openwisp_radius', 'RadiusGroupReply')
RadiusUserGroup = load_model('openwisp_radius', 'RadiusUserGroup')
OrganizationRadiusSettings = load_model('openwisp_radius', 'OrganizationRadiusSettings')
User = get_user_model()

admin.site.unregister(RadiusCheck)
admin.site.unregister(RadiusReply)
admin.site.unregister(RadiusAccounting)
admin.site.unregister(Nas)
admin.site.unregister(RadiusGroup)
admin.site.unregister(RadiusUserGroup)
admin.site.unregister(RadiusGroupCheck)
admin.site.unregister(RadiusGroupReply)
admin.site.unregister(RadiusPostAuth)
admin.site.unregister(RadiusBatch)

@admin.register(RadiusCheck)
class RadiusCheckAdmin(BaseRadiusCheckAdmin):
    # add your changes here
```

(continues on next page)

(continued from previous page)

```
@admin.register(RadiusReply)
class RadiusReplyAdmin(BaseRadiusReplyAdmin):
    # add your changes here

@admin.register(RadiusAccounting)
class RadiusAccountingAdmin(BaseRadiusAccountingAdmin):
    # add your changes here

@admin.register(Nas)
class NasAdmin(BaseNasAdmin):
    # add your changes here

@admin.register(RadiusGroup)
class RadiusGroupAdmin(BaseRadiusGroupAdmin):
    # add your changes here

@admin.register(RadiusUserGroup)
class RadiusUserGroupAdmin(BaseRadiusUserGroupAdmin):
    # add your changes here

@admin.register(RadiusGroupCheck)
class RadiusGroupCheckAdmin(BaseRadiusGroupCheckAdmin):
    # add your changes here

@admin.register(RadiusGroupReply)
class RadiusGroupReplyAdmin(BaseRadiusGroupReplyAdmin):
    # add your changes here

@admin.register(RadiusPostAuth)
class RadiusPostAuthAdmin(BaseRadiusPostAuthAdmin):
    # add your changes here

@admin.register(RadiusBatch)
class RadiusBatchAdmin(BaseRadiusBatchAdmin):
    # add your changes here
```

14.11 11. Setup Freeradius API Allowed Hosts

Add allowed freeradius hosts in `settings.py`:

```
OPENWISP_RADIUS_FREERADIUS_ALLOWED_HOSTS = ['127.0.0.1']
```

Note: Read more about *freeradius allowed hosts in settings page*.

14.12 12. Setup Periodic tasks

Some periodic commands are required in production environments to enable certain features and facilitate database cleanup:

1. You need to create a celery configuration file as it's created in example file.

2. In the settings.py, configure the CELERY_BEAT_SCHEDULE. Some celery tasks take an argument, for instance `365` is given here for `delete_old_radacct` in the example settings. These arguments are passed to their respective management commands. More information about these parameters can be found at the *management commands page*.

3. Add the following in your settings.py file:

```
CELERY_IMPORTS = ('openwisp_monitoring.device.tasks',)
```

Note: Celery tasks do not start with django server and need to be started seperately, please read about running *celery and celery-beat* tasks.

14.13 13. Create root URL configuration

The root `url.py` file should have the following paths (please read the comments):

```
from openwisp_radius.urls import get_urls
# Only imported when views are extended.
# from myradius.api.views import views as api_views
# from myradius.social.views import views as social_views
# from myradius.saml.views import views as saml_views

urlpatterns = [
    # ... other urls in your project ...
    path('admin/', admin.site.urls),
    # openwisp-radius urls
    path('accounts/', include('openwisp_users.accounts.urls')),
    path('api/v1/', include('openwisp_utils.api.urls')),
    # Use only when extending views (dicussed below)
    # path('', include((get_urls(api_views, social_views, saml_views), 'radius'), namespace=
↪'radius')),
    path('', include('openwisp_radius.urls', namespace='radius')), # Remove when␣
↪extending views
]
```

Note: For more information about URL configuration in django, please refer to the "URL dispatcher" section in the django documentation.

14.14 14. Import the automated tests

When developing a custom application based on this module, it's a good idea to import and run the base tests too, so that you can be sure the changes you're introducing are not breaking some of the existing features of *openwisp-radius*.

In case you need to add breaking changes, you can overwrite the tests defined in the base classes to test your own behavior.

See the tests of the sample app to find out how to do this.

You can then run tests with:

```
# the --parallel flag is optional
./manage.py test --parallel myradius
```

Substitute `myradius` with the name you chose in step 1.

14.15 Other base classes that can be inherited and extended

The following steps are not required and are intended for more advanced customization.

14.15.1 1. Extending the API Views

The API view classes can be extended into other django applications as well. Note that it is not required for extending *openwisp-radius* to your app and this change is required only if you plan to make changes to the API views.

Create a view file as done in API views.py.

Remember to use these views in root URL configurations in point 11. If you want only extend the API views and not social views, you can use `get_urls(api_views, None)` to get social_views from *openwisp_radius*.

Note: For more information about django views, please refer to the views section in the django documentation.

14.15.2 2. Extending the Social Views

The social view classes can be extended into other django applications as well. Note that it is not required for extending *openwisp-radius* to your app and this change is required only if you plan to make changes to the social views.

Create a view file as done in social views.py.

Remember to use these views in root URL configurations in point 11. If you want only extend the API views and not social views, you can use `get_urls(api_views, None)` to get social_views from *openwisp_radius*.

14.15.3 3. Extending the SAML Views

The SAML view classes can be extended into other django applications as well. Note that it is not required for extending *openwisp-radius* to your app and this change is required only if you plan to make changes to the SAML views.

Create a view file as done in saml views.py.

Remember to use these views in root URL configurations in point 11. If you want only extend the API views and social view but not SAML views, you can use `get_urls(api_views, social_views, None)` to get saml_views from *openwisp_radius*.

Note: For more information about django views, please refer to the views section in the django documentation.

CHAPTER

FIFTEEN

CAPTIVE PORTAL MOCK VIEWS

The development environment of openwisp-radius provides two URLs that mock the behavior of a captive portal, these URLs can be used when testing frontend applications like openwisp-wifi-login-pages during development.

Note: These views are meant to be used just for development and testing.

15.1 Captive Portal Login Mock View

- **URL**: `http://localhost:8000/captive-portal-mock/login/`.
- **POST fields**: `auth_pass` or `password`.

This view looks for `auth_pass` or `password` in the POST request data, and if it finds anything will try to look for any `RadiusToken` instance having its key equal to this value, and if it does find one, it makes a `POST` request to accouting view to create the radius session related to the user to which the radius token belongs, provided there's no other open session for the same user.

15.2 Captive Portal Logout Mock View

- **URL**: `http://localhost:8000/captive-portal-mock/logout/`.
- **POST fields**: `logout_id`.

This view looks for an entry in the `radacct` table with `session_id` equals to what is passed in the `logout_id` POST field and if it finds one, it makes a `POST` request to accounting view to flags the session as terminated by passing `User-Request` as termination cause.

CHAPTER

SIXTEEN

SUPPORT

The OpenWISP community is very active and offers best effort support through the official OpenWISP Support Channels.

CHAPTER

SEVENTEEN

CONTRIBUTING

Thank you for taking the time to contribute to openwisp-radius, please read the guide for contributing to openwisp repositories.

Follow these guidelines to speed up the process.

Table of Contents:

Note: **In order to have your contribution accepted faster**, please read the OpenWISP contributing guidelines and make sure to follow its guidelines.

17.1 Setup

Once you have chosen an issue to work on, *setup your machine for development*.

17.2 Ensure test coverage does not decrease

First of all, install the test requirements:

```
workon radius  # activate virtualenv
pip install --no-cache-dir -U -r requirements-test.txt
```

When you introduce changes, ensure test coverage is not decreased with:

```
coverage run --source=openwisp_radius runtests.py
```

17.3 Follow style conventions

First of all, install the test requirements:

```
workon radius  # activate virtualenv
pip install --no-cache-dir -U -r requirements-test.txt
npm install -g jslint
```

Before committing your work check that your changes are not breaking our coding style conventions:

```
# reformat the code according to the conventions
openwisp-qa-format
# run QA checks
./run-qa-checks
```

For more information, please see:

- OpenWISP Coding Style Conventions

17.4 Update the documentation

If you introduce new features or change existing documented behavior, please remember to update the documentation!

The documentation is located in the `/docs` directory of the repository.

To do work on the docs, proceed with the following steps:

```
workon radius  # activate virtualenv
pip install sphinx
cd docs
make html
```

17.5 Send pull request

Now is time to push your changes to github and open a pull request!

CHAPTER

EIGHTEEN

MOTIVATIONS AND GOALS

In this page we explain the goals of this project and the motivations that led us on this path.

18.1 Motivations

The old version of OpenWISP (which we call OpenWISP 1) had a freeradius module which provided several interesting features:

- user registration
- account verification with several methods
- user management
- password reset
- basic general statistics
- basic user account page with user's statistics

But it also had important problems:

- it was not written with automated testing in mind, so there was a lot of code which the maintainers didn't want to touch because of fear of breaking existing features
- it was not written with an international user-base in mind, it contained a great deal of code which was specific to a single country (Italy)
- it was hard to extend, even small changes required changing its core code
- the user management code was implemented in a different way compared to other openwisp1 modules, which added a lot of maintenance overhead
- it used outdated dependencies which over time became vulnerable and were hard to replace
- **it did not perform hashing of user passwords**
- the documentation did not explain how to properly install and configure the software

Similar problems were affecting other modules of OpenWISP 1, that's why over time we got convinced the best thing was to start fresh using best practices since the start.

18.2 Project goals

The main aim of this project is to offer a web application and documentation that helps people from all over the world to implement a wifi network that can use freeradius to authenticate its users, either via captive portal authentication or WPA2 enterprise, **BUT** this doesn't mean we want to lock the software to this use case: we want to keep the software generic enough so it can be useful to implement other use cases that are related to networking connectivity and network management; **Just keep in mind our main aim if you plan to contribute to openwisp-radius please**.

Other goals are listed below:

- replace the user management system of OpenWISP 1 by providing a similar feature set
- provide a web interface to manage a freeradius database
- provide abstract models and admin classes that can be imported, extended and reused in third party apps
- provide ways to extend the logic of openwisp-radius without changing its core
- ensure the code is written with an international audience in mind
- maintain a very good automated test suite
- reuse the django user management logic which is very robust and stable
- ensure passwords are hashed with strong algorithms and freeradius can authorize/authenticate using these hashes (that's why we recommend using the `rlm_rest` freeradius module with the REST API of openwisp-radius)
- integrate openwisp-radius with the rest of the openwisp2 ecosystem
- provide good documentation on how to install the project, configure it with freeradius and use its most important features

CHAPTER

NINETEEN

CHANGE LOG

19.1 Version 1.0.1 [2022-05-10]

- Fixed a bug in the organization radius settings form which was causing it to not display some default values correctly
- Fixed a bug in allowed mobile prefix implementation: the implementation was joining the globally allowed prefixes and the prefixes allowed at org level, with the result that disabling a prefix at org level was not possible
- Called-station-ID command: log with warning instead of `warn` or `error`: - warn > warning (warn is deprecated) - use warning instead of errors for more temporary connection issues cases

19.2 Version 1.0.0 [2022-04-18]

19.2.1 Features

- Allowed to login via API with email or phone number
- Allowed freeradius authorize with email or phone number
- Allowed the usage of subnets in OPENWISP_RADIUS_FREERADIUS_ALLOWED_HOSTS
- Made the fields containing personal data of users which are exposed in the registration API configurable (allowed, mandatory, disabled) via the OPENWISP_RADIUS_OPTIONAL_REGISTRATION_FIELDS setting or the admin interface
- Allow to disable registration API via the OPENWISP_RADIUS_REGISTRATION_API_ENABLED setting or the admin interface
- Added throttling of API requests
- Added OPENWISP_RADIUS_API_BASEURL setting
- Add identity verification feature, configurable via the OPENWISP_RADIUS_NEEDS_IDENTITY_VERIFICATION or via admin interface
- Added utilities for implementing new registration and identity verification methods
- Added captive portal mock views to ease development and debugging
- Add possibility to filter users by registration method in the admin interface
- Added SAML registration method to implement captive portal authentication via Single Sign On (SSO)
- Added management command and celery task to delete unverified users
- Added translations of user facing API responses in Italian, German, Slovenian and Furlan

- Added Convert RADIUS accounting CALLED-STATION-ID feature, celery task and management command, with the possibility of triggering it on accounting creation (see OPENWISP_RADIUS_CONVERT_CALLED_STATION_ON_CREATE)
- Added an equivalent of the FreeRADIUS sqlcounter feature to the REST API
- Added emission of django signal to FreeRADIUS accounting view: radius_accounting_success
- Added possibility to send email to the user an they start a new radius accounting session
- Added organization level settings and related admin interface functionality to enable/disable SAML and social login:
  - OPENWISP_RADIUS_SAML_REGISTRATION_ENABLED
  - OPENWISP_RADIUS_SOCIAL_REGISTRATION_ENABLED
- Added setting to avoid updating username from SAML: OPENWISP_RADIUS_SAML_UPDATES_PRE_EXISTING_USERNAM

19.2.2 Changes

Backward incompatible changes

- Updated prefixes of REST API URLs:
  - API endpoints dedicated to FreeRADIUS have moved to `/api/v1/freeradius/`
  - the rest of the API endpoints have moved to `/api/v1/radius/`
- Allowed `username` and `phone_number` in password reset API, the endpoint now accepts the "input" parameter instead of "email"
- Removed customizations for checks and password hashing because they are unmaintained, any user needing these customizations is advised to implement them as a third party app
- Improved REST API to change password: inherited `PasswordChangeView` of openwisp-users to add support for the current-password field in password change view

Dependencies

- Added support for Django 3.2 and 4.0
- Dropped support for Django 2.2
- Upgraded celery to 5.2.x
- Updated and tested Django REST Framework to 3.13.0
- Added support for Python 3.8, 3.9
- Removed support for Python 3.6

Other changes

- Moved AccountingView to freeradius endpoints
- Relaxed default values for the SMS token settings
- Switched to new navigation menu and new OpenWISP theme
- Allowed users to sign up to multiple organizations
- Update username when phone number is changed if username is equal to the phone number
- Update stop time and termination to `None` if `status_type` is `Interim-Update`
- Send password reset emails using HTML theme: leverage the new openwisp-utils send_email function to send an HTML version of the reset password email based on the configurable email HTML theme of OpenWISP
- Save the user preferred language in obtain and validate token views
- Added validation check to prevent invalid username in batch user creation
- Allowed to set the Password Reset URL setting via the admin interface
- Added soft limits to celery tasks for background operations
- Generalized the implementation of the fallback model fields which allow overriding general settings for each organization

19.2.3 Bugfixes

- Fixed login template of openwisp-admin-theme
- Fixed swagger API docs collision with openwisp-users
- Ensured each user can be member of a group only once
- Radius check and reply should check for organization membership
- `ValidateAuthTokenView`: show `phone_number` as `null` if `None`
- Freeradius API: properly handle interaction between multiple orgs: an user trying to authorize using the authorization data of an org for which they are not member of must be rejected
- Fixed radius user group creation with multiple orgs
- Added validation of phone number uniqueness in the registration API
- Fixed issues with translatable strings:
  - we don't translate log lines anymore because these won't be shown to end users
  - `gettext` does not work with fstrings, therefore the use of `str.format()` has been restored
  - improved some user facing strings
- Fixed Accounting-On and Accounting-Of accounting requests with blank usernames
- Delete any cached radius token key on phone number change
- Fixed handling of interim-updates for closed sessions: added handling of "Interim-Updates" for RadiusAccounting sessions that are closed by OpenWISP when user logs into another organization
- Flag user as verified in batch user creation
- Added validation which prevents the creation of duplicated check/reply attributes

19.3 Version 0.2.1 [2020-12-14]

19.3.1 Changes

- Increased openwisp-users and openwisp-utils versions to be consistent with the OpenWISP 2020-12 release
- Increased dj-rest-auth to 2.1.2 and weasyprint to 52

19.4 Version 0.2.0 [2020-12-11]

19.4.1 Features

- Changing the phone number via the API now keeps track of previous phone numbers used by the user to comply with ISP legal requirements

19.4.2 Changes

- Obtain Auth Token View API endpoint: added `is_active` attribute to response
- Obtain Auth Token View API endpoint: if the user attempting to authenticate is inactive, the API will return HTTP status code 401 along with the auth token and `is_active` attribute
- Validate Auth Token View API endpoint: added `is_active`, `phone_number` and `email` to response data
- When changing phone number, user is flagged as inactive only after the phone token is created and sent successfully
- All API endpoints related to phone token and SMS sending are now disabled (return 403 HTTP response) if SMS verification not enabled at organization level

19.4.3 Bugfixes

- Removed `static()` call from media assets
- Fixed password reset for inactive users
- Fixed default password reset URL value and added docs
- Documentation: fixed several broken internal links

19.5 Version 0.1.0 [2020-09-10]

- administration web interface
- support for freeradius 3.0
- multi-tenancy
- REST API
- integration with rlm_rest module of freeradius
- possibility of registering new users via API
- social login support

- mobile phone verification via SMS tokens
- possibility to import users from CSV files
- possibility to generate users for events
- management commands and/or celery tasks to perform clean up operations and periodic tasks
- possibility to extend the base classes and swap models to add custom functionality without changing the core code