freeRADIUS

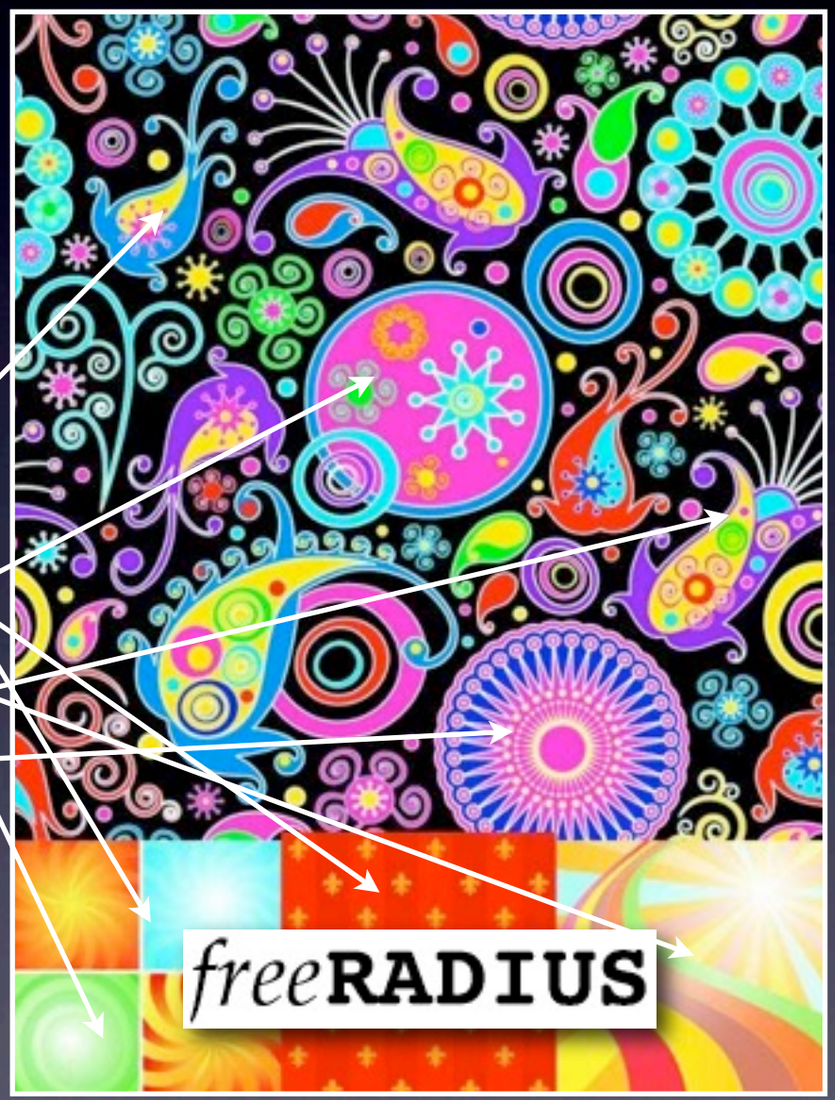
Serveur RADIUS libre, performant et modulaire

mais pas vraiment simple

Aurélien Geron, Wifirst, 7 janvier 2011

Plan

- Plusieurs protocoles : RADIUS, EAP...
- Un serveur sous GPLv2
- Un système de configuration puissant
- Une multitude de modules
- Comment créer un module ?



Source image: http://crshare.com/abstract-backgrounds-vector-clipart/

Module python

- Editer `/etc/freeradius/modules/python` et indiquer le nom du module et de la fonction *python* à appeler pour chaque phase du traitement des requêtes RADIUS :

```
python mon_module_python {
     mod_instantiate = mon_package.mon_module
     func_instantiate = instantiate
     mod_authenticate = mon_package.mon_module
     func_authenticate = authenticate
     mod_authorize = mon_package.mon_module
     func_authorize = authorize
     mod_checksimul = mon_package.mon_module
     func_checksimul = checksimul
     mod_pre_acct = mon_package.mon_module
     func_pre_acct = pre_acct
     mod_accounting = mon_package.mon_module
     func_accounting = accounting
     mod_pre_proxy = mon_package.mon_module
     func_pre_proxy = pre_proxy
     mod_post_proxy = mon_package.mon_module
     func_post_proxy = post_proxy
     mod_post_auth = mon_package.mon_module
     func_post_auth = post_auth
     mod_recv_coa = mon_package.mon_module
     func_recv_coa = recv_coa
     mod_send_coa = mon_package.mon_module
     func_send_coa = send_coa
     mod_detach = mon_package.mon_module
     func_detach = detach
}
```

Module python

- On peut maintenant écrire le module python très simplement :

```
import radiusd

def get_radius_attribute(attribute, attributes):
    for attr, val in attributes:
        if attr==attribute:
            return val

def instantiate(p):
    radiusd.radlog(radiusd.L_DBG, '*** python: instantiate ***')
    return radiusd.RLM_MODULE_OK

def authorize(request_attributes):
    radiusd.radlog(radiusd.L_DBG, '*** python: authorize ***')
    username=get_radius_attribute('User-Name', request_attributes)
    if username!='"alain"':
        return radiusd.RLM_MODULE_NOTFOUND
    return (radiusd.RLM_MODULE_UPDATED,
            (('Session-Timeout', str(1000)),('Filter-ID', 'web-seul')),
            (('Auth-Type', 'python'),))

def authenticate(request_attributes):
    radiusd.radlog(radiusd.L_DBG, '*** python: authenticate ***')
    password=get_radius_attribute('User-Password', request_attributes)
    if password!='"POfFoFjnv3$"':
        radiusd.radlog(radiusd.L_INFO, '*** python: mauvais mdp ***')
        return radiusd.RLM_MODULE_REJECT
    return radiusd.RLM_MODULE_OK
```

Le module **radiusd** contient la fonction `radlog` et les constantes des codes retour

`p` ne sert à rien pour l'instant

Chaque fonction reçoit la liste des attributs de la liste **request** sous la forme d'un *tuple* `((attr1, val1), (attr2, val2),...)`. On n'a malheureusement pas accès aux listes **control** ou **reply** (qui se dévoue pour améliorer `rlm_python`) ?

Attention : les valeurs des attributs de type `string` sont entourées par des guillemets doubles

On peut renvoyer juste le code retour ou bien un *tuple* contenant le code retour, une liste d'attributs à rajouter à la liste **reply**, et une autre liste d'attributs à rajouter à la liste **control**

`/usr/lib/python2.5/site-packages/mon_package/mon_module.py`

➡ Il faut s'assurer que le module soit placé dans l'un des répertoires où *python* va chercher (cela dépend des systèmes). Ne pas oublier non plus de créer un fichier `__init__.py` dans le répertoire `mon_package`.

Librairies dynamiques

- ***Malheureusement, sur certaines plateformes (essentiellement Debian), pour les versions < 2.1.9 de FreeRADIUS, le code python échoue dès qu'un module python dynamique est importé (c'est à dire tous les modules standard ou presque :*** `/usr/lib/python2.5/lib-dynload/*.so`***).***
- ***Sur les plateformes concernées par ce bug, on ne peut donc, par défaut, qu'écrire des modules python extrêmement simples. On ne peut même pas faire*** `import math` ***par exemple !***

Détails

Si un programme importe une librairie sans l'option `RTDL_GLOBAL`***, alors les autres librairies importées ne pourront pas accéder aux symboles de cette librairie. Or, FreeRADIUS importe dynamiquement*** `rlm_python`***, qui importe*** `libpython`***, qui importe par exemple*** `math.so`***. Sous Debian, l'option*** `RTDL_GLOBAL` ***est désactivée par défaut (entre autres par mesure de sécurité, c'est une des seules plateformes configurées ainsi), donc*** `math.so` ***n'a pas accès aux symboles de*** `libpython`***, alors qu'il en a besoin. L'erreur suivante s'affiche :***

```
/usr/lib/python2.5/lib-dynload/math.so: undefined symbol: PyExc_ValueError
```

Il y a le même problème, exactement, avec perl. Pour plus de détails (en anglais) :
http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=416266

Librairies dynamiques

- ***Heureusement, on peut contourner le problème en forçant FreeRadius à précharger la librairie dynamique de python. Pour cela, il suffit de rajouter la ligne suivante au début de*** `/etc/init.d/freeradius` **:**
  `export LD_PRELOAD=/usr/lib/libpython2.5.so.1`
  - ➡ ***Il faut penser à mettre cette ligne à jour si la version change !***
- ***Par ailleurs, FreeRADIUS 2.1.9 résout proprement ce problème pourvu qu'une option de compilation*** `HAVE_HAVE_LT_DLADVISE_INIT` ***soit définie à la compilation. Dans FreeRADIUS 2.1.11, l'option a été renommée*** `HAVE_LT_DLADVISE_INIT` ***(il y avait un copier/coller de trop).***
- ***Les premiers packages Debian pour la version 2.1.9 et 2.1.10 n'ont pas utilisé l'option (suite au nom erroné), donc sont mauvais pour python. Il faut prendre des packages récents (après mi-janvier 2011).***

Détails

Une fonction `lt_dlopenadvise` ***du package*** `libltdl` ***permet de charger une librairie dynamique avec l'option*** `RTDL_GLOBAL`***, alors que la fonction par défaut*** `lt_dlopenext` ***ne le permet pas sous Debian.***
FreeRADIUS 2.1.9 et ultérieur utilisent la fonction `lt_dlopenadvise` ***quand l'option de compilation est précisée. Ainsi, les symboles de la librairie python sont accessibles aux modules python chargés dynamiquement.***

Bonus Django !

- On peut facilement écrire un module *python* avec un accès à une base de données, le tout avec le framework Web *Django*
- Pour installer le framework sous Debian :

```
#aptitude update
#aptitude install python-django
```

- Ensuite on crée le projet `ma_societe` contenant une application `mon_appli`, et on enlève les fichiers utiles uniquement dans un contexte Web :

```
#cd /usr/lib/python2.5/site-packages
#django-admin startproject ma_societe
#cd ma_societe
#django-admin startapp mon_appli
#find
.
./manage.py
./__init__.py
./urls.py
./mon_appli
./mon_appli/__init__.py
./mon_appli/views.py
./mon_appli/models.py
./settings.py
#rm urls.py mon_appli/views.py
```

Bonus Django !

- On peut configurer l'accès à la base de données dans le fichier `settings.py` (c'est tout ce qu'il a besoin de contenir, d'ailleurs) :

```
# -*- coding: utf-8 -*-
DEBUG = True
DATABASE_ENGINE = 'mysql'
DATABASE_NAME = 'radius'
DATABASE_USER = 'radius'
DATABASE_PASSWORD = 'FlG,4Vl$DnG'
DATABASE_HOST = 'mysql.ma-societe.fr'
DATABASE_PORT = '3306'
```

ma_societe/settings.py

- Ensuite on peut créer les modèles *Django*, tout à fait normalement :

```
from django.db import models

class Utilisateur(models.Model):
    identifiant  = models.CharField
    mot_de_passe = models.CharField

    def __unicode__(self):
        return self.identifiant
```

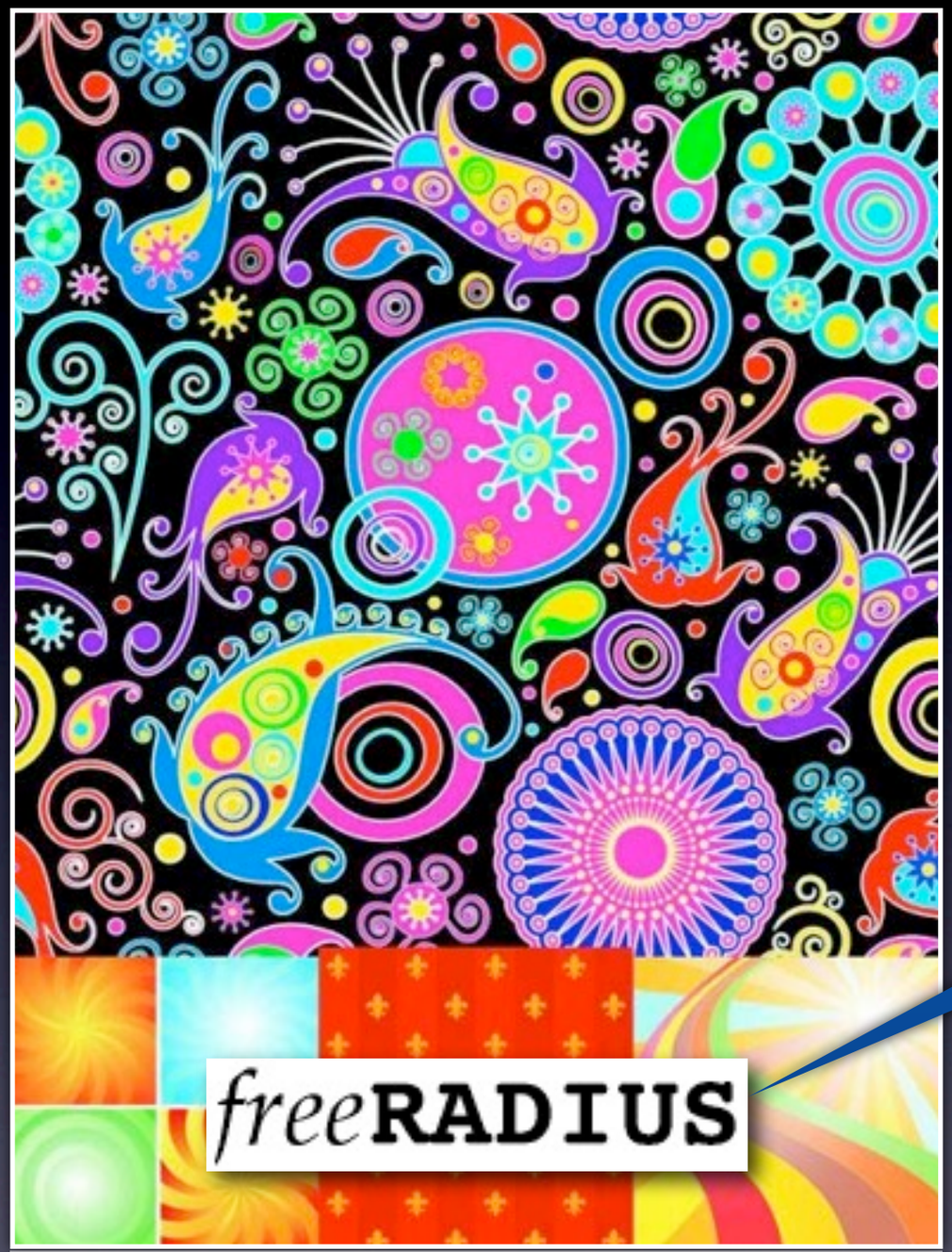
ma_societe/mon_appli/models.py

Bonus Django !

- Il faut fixer la variable d'environnement `DJANGO_SETTINGS_MODULE` pour pointer vers le module qui contient nos réglages de base de données
- Par exemple dans `/etc/init.d/freeradius`, rajouter :
  `export DJANGO_SETTINGS_MODULE=ma_societe.settings`
- On peut maintenant écrire le module proprement dit, en utilisant les modèles Django

Ecrire un module en C

- Le plus simple est de s'inspirer du module `rlm_example`, qui fait partie du code source de freeRADIUS
- Le principe est le même que pour python, avec en plus la possibilité d'accéder à la configuration.
- L'API n'est pas documentée (à ma connaissance) : il faut lire les fichiers `.h`



Questions ?