

 **niveau1.md**

On entre dans le vif du sujet !

## Contenu Niveau 1

Dans le niveau 1 nous verrons:

[ Niveau 1 - Ligne de commande, boucles, lecture de fichier ]

- Installer un programme/un package avec pip install - NIVEAU 1
- Utiliser brutegen pour générer des combinaisons - NIVEAU 1
- Utiliser crackrar pour cracker un mot de passe en ligne de commande - NIVEAU1
- Estimer le temps nécessaire pour cracker un mot de passe - NIVEAU 1
- Réfléchir à des alternatives à l'attaque par force brute- NIVEAU 1
- Tester un mot de passe .rar en python - NIVEAU 1
- Tester tous les nombres possibles - NIVEAU 1
- Tester tous les mots de passe stockés dans un fichier (attaque par dictionnaire) - NIVEAU 1

Nous allons installer un programme: **crackrar** et son jumeau **brutegen**

**brutegen** est un programme d'illustration permettant de générer des combinaisons de mot de passe. **crackrar** est un programme qui repose sur les mêmes principes, mais appliqués à l'ouverture de fichiers .rar cryptés

## Installer un programme/un package avec pip install

**python** est un **programme** qui permet de **lire le code** qu'on écrit des fichiers (des "**scripts**") ou au clavier ("**l'interpréteur**") pour ensuite **traduire** nos instructions en langage compréhensible par la machine puis **exécuter les actions demandées**.

Il est possible d'y ajouter des **extensions** avec tout un tas de choses déjà codées (souvent en python, mais certaines extensions peuvent aussi être codées en C, en Java, etc).

Ces extensions sont appelées des **modules** si ce sont des **scripts** sauvegardés sur l'ordinateur avec une extension **".py"** ou sinon des **packages** si ils sont **installables**.

Pour **installer un package** python on utilise un programme qui s'appelle **pip**, et qui est livré par défaut avec python.

**pip** est un installateur d'extensions pour python

pip, comme de nombreux programmes, s'utilise en **ligne de commande**. C'est-à-dire que pour l'utiliser vous devez utiliser sous Linux **CTRL+ALT+T**", et ensuite **taper son nom** et appuyer sur **entrée**

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
delevoye@gmcp04:/export/home1/users/gmc/delevoye$ pip

Usage:
  pip <command> [options]

Commands:
  install          Install packages.
  download         Download packages.
  uninstall        Uninstall packages.
  freeze           Output installed packages in requirements format.
  list             List installed packages.
  show             Show information about installed packages.
  check            Verify installed packages have compatible dependencies.
  config           Manage local and global configuration.
  search           Search PyPI for packages.
  wheel            Build wheels from your requirements.
  hash             Compute hashes of package archives.
  completion       A helper command used for command completion.
  debug            Show information useful for debugging.
  help             Show help for commands.

General Options:
  -h, --help          Show help.
  --isolated           Run pip in an isolated mode, ignoring
                      environment variables and user configuration.
  -v, --verbose       Give more output. Option is additive, and can be
                      used up to 3 times.
  -V, --version       Show version and exit.
  -q, --quiet         Give less output. Option is additive, and can be
                      used up to 3 times (corresponding to WARNING,
                      ERROR, and CRITICAL logging levels).
  --log <path>       Path to a verbose appending log.
  --proxy <proxy>     Specify a proxy in the form
                      [user:passwd@]proxy.server:port.
  --retries <retries> Maximum number of retries each connection should
                      attempt (default 5 times).
  --timeout <sec>     Set the socket timeout (default 15 seconds).
  --exists-action <action> Default action when a path already exists:
                      (s)witch, (i)gnore, (w)ipe, (b)ackup, (a)bort.
  --trusted-host <hostname> Mark this host or host:port pair as trusted,
                      even though it does not have valid or any HTTPS.
  --cert <path>       Path to alternate CA bundle.
  --client-cert <path> Path to SSL client certificate, a single file
                      containing the private key and the certificate
                      in PEM format.
  --cache-dir <dir>   Store the cache data in <dir>.
  --no-cache-dir      Disable the cache.
  --disable-pip-version-check Don't periodically check PyPI to determine
                      whether a new version of pip is available for
                      download. Implied with --no-index.
  --no-color          Suppress colored output
delevoye@gmcp04:/export/home1/users/gmc/delevoye$ 

```

Dans la plupart des cas, un programme en **ligne de commande** (= Utilisable par la console) affichera une erreur ou une aide si on le tape seul.

La ligne ***pip [options]***

...est ici celle qui nous intéresse le plus ici

Ce qui s'affiche ici à l'écran est une **aide**, un **manuel** pour utiliser le programme. Comme on sait que pip est un programme pour **installer** des packages (ce que nous voulons faire ici)

## Façons d'installer un package

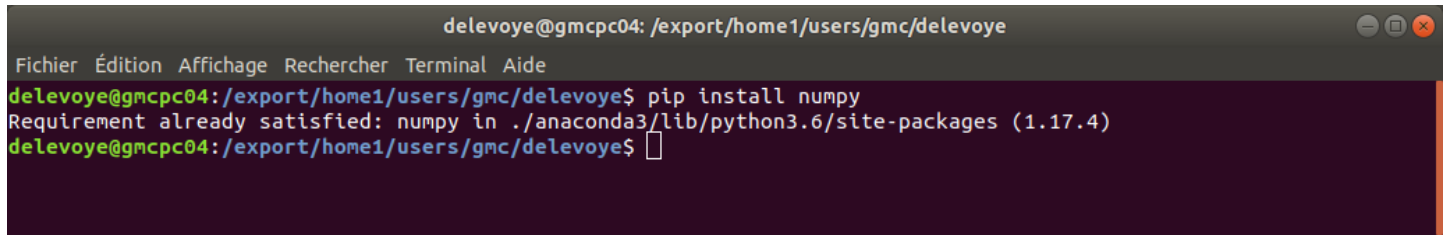
IL y a plusieurs façons d'installer un package

**Si le package est disponible sur PyPI (pas pour nous aujourd'hui)**

Certains programmes sont stockés en ligne sur le site [ <http://www.PyPI.org> ]. Lorsque c'est le cas on peut installer le programme juste en utilisant

```
user@computer$:pip install NOM_DU_PACKAGE
```

C'est le cas par exemple du package "**numpy**":

A screenshot of a terminal window titled 'delevoye@gmcp04: /export/home1/users/gmc/delevoye'. The terminal shows the command 'pip install numpy' being executed. The output indicates that the requirement is already satisfied, showing 'numpy in ./anaconda3/lib/python3.6/site-packages (1.17.4)'. The prompt returns to 'delevoye@gmcp04: /export/home1/users/gmc/delevoye\$'.

Ici, le package numpy est déjà installé chez moi

## Avec une "release"

Lorsque le package n'est pas disponible sur PyPI il faut alors le télécharger d'abord, et l'installer en local.

Le package qui va nous intéresser aujourd'hui est **crackrar**

Vous pouvez l'installer en allant à l'URL du package [ <https://github.com/GDelevoye/codingsisters-crackrar> ], puis en cliquant sur **release**. Vous pouvez alors choisir la version qui vous intéresse (généralement la dernière). Télécharger le fichier zip (ou .tar.gz). Le décompresser.

Une fois que vous avez décompressé le fichier, ouvrez un terminal, et utilisez pip pour installer le package

```
pip install -e ./[CHEMIN_VERS_FICHIER_DECOMPRESSE]/[NOM_DU_DOSSIER_DECOMPRESSE]
```

Ici, "**-e**" est une **option**, un **argument** du script pour préciser que le package est "éditable", c'est à dire que vous pourrez modifier le programme sans avoir à le réinstaller à chaque fois

### ATTENTION:

Dans le cas où on installe un package en local il **faut** indiquer le chemin avec **"./"**[CHEMIN]. Sinon, **pip chercherait le package sur PyPI, et risquerait de ne pas le trouver** ou pire, d'installer **une autre version du package** voir un autre package qui porterait le même nom sur PyPI.org

## Depuis gitHub avec git (pour nous !)

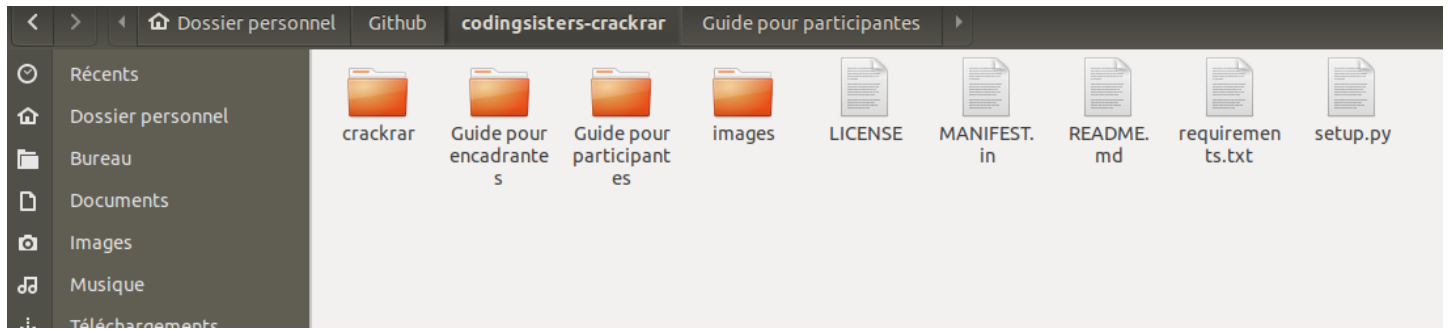
Quand le package est disponible sur le site GitHub on peut l'installer en clonant le répertoire, puis en demandant à pip de l'installer

```
git clone https://github.com/GDelevoye/codingsisters-crackrar.git
mv ./codingsisters-crackrar crackrar
pip install ./crackrar
```

Et voilà !

## Utiliser brutegen pour générer des combinaisons

Dans le dossier "crackrar", vous trouvez notre programme !



Maintenant que le programme est installé, on peut le lancer depuis n'importe quel terminal, et voir comment il s'utilise

Le package contient deux programmes : **crackrar** et **brutegen**. Nous allons commencer avec brutegen

1. **brutegen** permet de générer des combinaisons en "force brute" (essayer toutes les combinaisons) et les afficher à l'écran.
2. **crackrar** permet de faire la même chose, mais sur des fichiers .RAR pour les crackner

Regardons comment **brutegen** s'utilise !

```
delevoeye@gmcp04:/export/home1/users/gmc/delevoeye$ brutegen
usage: brutegen [-h] --combination_length COMBINATION_LENGTH
               [--charsets {lowercase,uppercase,digits,special_characters} [{lowercase,uppercase,digits,special_characters}]]
               [--dictionary DICTIONNARY]
brutegen: error: the following arguments are required: --combination_length/-l
```

- Le premier argument **-h** permet d'afficher le message d'aide
- Le deuxième argument **--combination\_length** doit être suivi d'un nombre entier: La longueur de la combinaison que vous voulez générer

Attention: Si vous en mettez une très longue, ce sera très long !

- Le troisième argument **--charset** nous permet d'utiliser des majuscules, des minuscules, des chiffres, des caractères spéciaux. A nous de choisir !
- Le quatrième argument **--dictionnaire** est un chemin vers un fichier qui comporte un mot par ligne. brutegen va alors afficher tous ces mots, et ne pas se soucier des autres arguments

Ici, un seul argument est obligatoire: **"--combination-length"** , qui peut être abrégé **"-l"**

## A vous de jouer !

Vous pouvez utiliser brutegen pour afficher à l'écran ce que vous voulez. Par exemple:

- Toutes les combinaisons de 3 caractères
- Toutes les combinaisons de 4 caractères spéciaux
- Tous les mots en minuscules de 5 caractères

Vous pouvez vérifier que tous les mots de n caractères sont bien présents ! Par exemple, avec tous les mots en minuscule de 5 caractères, on devrait avoir le mot "chien", "chine" et "niche". Vous pouvez le vérifier avec la commande **"grep"**, qui ne va afficher que les mots que vous recherchez

Une fois que vous l'avez, passez à la suite !

## Comment crackner un fichier .rar crypté avec crackrar

Voici comment fonctionne crackrar

```
delevoeye@gmcp04:/export/home1/users/gmc/delevoeye$ crackrar
usage: crackrar [-h] [--combination_length COMBINATION_LENGTH]
```

```
--charsets {lowercase,uppercase,digits,special_characters} [{lowercase,uppercase,digits,special_characters}]
--dictionary DICTIONNARY
--verbosity {DEBUG,INFO,WARNING,ERROR, CRITICAL,SHOW_PSW}]
rarFile
```

```
crackrar: error: the following arguments are required: rarFile
```

Vous avez compris le principe ?

C'est quasiment le même principe que brutegen ! IL suffit juste d'ajouter un chemin vers un fichier crypté .rar.

Comme nous avons de la chance: Il existe justement un fichier .rar mal protégé que vous pouvez attaquer facilement !

Allez d'abord dans `./[REPERTOIRE_DU_PACKAGE]/crackrar/test_package_testdata`

```
delevoye@gmcp04:~/Github/codingsisters-crackrar/crackrar/test_package/testdata$ ls
dicttest1.txt dicttest2.txt lang-french-full.txt notes.rar
```

Le fichier qui nous intéresse est **notes.rar**

Quel est ce mot de passe ? Quelle longueur fait-il ? :)

Indice: La personne qui l'a crypté a utilisé le code PIN de son téléphone, parce qu'elle avait peur de l'oublier !

Une fois que tu sais répondre à la question, félicitation: Tu as cracké un premier mot de passe. Facile non ?

Je t'encourage à t'amuser à essayer des mots de passe de la longueur et de la composition que tu veux pour estimer le temps qu'il faut pour les cracker sur ton ordinateur, et à essayer de **comparer la différence de vitesse** entre brutegen et crackrar. Pour cela, n'hésite pas à demander aux tutrices comment utiliser la commande **grep** avec un **pipe**. C'est un très bon exercice de ligne de commande.

## Estimer le temps nécessaire pour cracker un mot de passe

Alors ça y est, c'est fini ?

Non. Pas du tout ! La preuve : As-tu regardé combien de temps il faut pour générer les différentes combinaisons avec brutegen ? Combien de temps il faut pour les essayer avec crackrar sur un vrai fichier ? Qu'est-ce qu'on peut en déduire ?

Combien de temps pense-tu qu'il faut pour essayer tous les mots de passes de 6 caractères sur lesquels on ne sait pas s'ils contiennent des lettres majuscules ou minuscules, des chiffres, des caractères spéciaux ? Et si on ne connaît pas la taille du mot de passe qu'on essaye d'attaquer ?

Tu peux mesurer la vitesse que brutegen et crackrar mettent pour travailler avec le programme pythonb **tqdm** et la commande **wc -l** qui permet de compter les lignes affichées à l'écran

```
user@computer:~/crackrar/crackrar/test_package/testdata$ brutegen -l 4 | python -m tqdm | wc -l
112550881it [01:42, 1095235.63it/s]
112550881
```

C'est fou, non ?

Ici, nous étions beaucoup aidés: Nous savions la longueur du mot de passe (un code PIN), et nous savons que ce genre de code ne possède que 4 chiffres parmi les 10 chiffres : [0,1,2,3,4,5,6,7,8,9]

Combien de combinaisons devait-t-on essayer ?

$10^4 = 10000$  combinaisons allant de 0000 à 9999 !

Pour un ordinateur, **c'est très peu** ! Mais si on cherche **toutes les combinaisons de majuscules, minuscules et chiffres de 10 caractère, combien de temps aurait-il fallu** ?

Essaye de répondre approximativement à cette question pour te rendre compte par toi-même !

Comment faire mieux ?

# Réfléchir à des alternatives à l'attaque par force brute

**brutegen** et **crackrar** sont des programmes très simples. Ils permettent de cracker sans se poser de questions tous les mots de passe jusqu'à une taille de 5 ou 6 caractères sans trop de problèmes, pourvu qu'on ait un peu de temps devant soi.

Néanmoins les gens ne choisissent pas leurs mots de passe au hasard. Très souvent ils utilisent:

- Des mots
- Des dates d'anniversaire
- Des codes PIN
- Des dates historiques
- Des noms d'animaux, ou de personnes proches
- Ils alternent les majuscules ou les minuscules
- Souvent quand on met des chiffres dans un mot de passe, on les met en début ou en fin de mot de passe. Très rarement en plein milieu !
- Pareil pour les caractères spéciaux
- Quand un site demande "au moins" 1 majuscule 1 minuscule 1 chiffre 1 caractère spécial et une longueur de 8, alors **énormément de personnes** vont utiliser **précisément** un mot de passe de la plus petite sécurité possible avec seulement une majuscule, seulement 1 chiffre, seulement un caractère spécial, et une longueur de seulement 8 !
- Certains mots de passe sont extrêmement utilisés dans le monde: "motdepasse", "password", "test", "admin", "admin", "azerty", "qwerty" etc
- Parfois certaines combinaisons sont évidentes entre ces mots de passe évidents: "**123Admin!**" par exemple, est un superbe exemple de mot de passe qui **remplit les critères de sécurité de beaucoup de sites internet** tout en étant **extrêmement utilisé et facile à trouver** !
- Parfois les utilisateurs utilisent des variations d'un mot de passe que vous connaissez déjà. Exemple: **Lechocolatcestbon24!** ou **Lecafécestbon28!**. C'est également très fréquent.

D'ailleurs, **le genre, l'âge, l'origine ethnique ou la culture de l'utilisateur influence également sur la nature du mot de passe**. Plusieurs études ont montré que les femmes avaient beaucoup plus tendance que les hommes à utiliser des mots de passe faisant référence à la **nourriture**. Les mots de passe faisant référence aux **BD comics** ou des personnages de **pop culture** sont extrêmement fréquents dans les milieux geeks ou fortement masculins.

Est-ce qu'il y a des mots de passe simples que brutegen et crackrar ne pourront pas craquer selon vous ? :) Que peut-on en conclure quant à nos deux outils en ligne de commande ?

**Conclusion n°1:** crackrar et brutegen sont des outils **trop simples** pour attaquer des longs mots de passe. Pourtant, certains mots de passe longs ou sécurisés seraient **faciles à trouver** si on réfléchissait intelligemment ! Nous pourrions pourtant attaquer ces mots de passe !\*\*\*

**Conclusion n°2:** Nous sommes des personnes intelligentes. Nous allons programmer en python pour coder des attaques sur ces mots de passe !

## Tester le mot de passe d'un fichier .rar en python

D'ordinaire on peut tout faire facilement en python. Pour tester un mot de passe sur un fichier .rar il suffit d'utiliser la fonction essayer\_mdp de crackrar

IL suffit d'ouvrir un interpréteur

```
delevoye@gmcp04:/export/home1/users/gmc/delevoye$ python
Python 3.6.7 | packaged by conda-forge | (default, Feb 28 2019, 09:07:38)
[GCC 7.3.0] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Les trois chevrons >>> signifie que vous pouvez taper des commandes en python

Puis ensuite, on va importer la fonction essayer\_mdp qui fait partie du package crackrar, et afficher son aide:

```
>>> from crackrar import essayer_mdp
>>> help(essayer_mdp)
```

Ce qui nous donnera:

```
Help on function essayer_mdp in module crackrar.rar_ouverture:

essayer_mdp(chemin_vers_fichier_RAR, mdp, nom_fichier=None)
    Permet de tester le mot de passe d'un fichier RAR
    Renvoie True si le mot de passe est bon, et renvoie False si on n'a pas
    pas réussi à ouvrir le fichier RARself.

    Si le nom du fichier (nom_fichier) n'est pas donné en détail, alors par
    défaut la fonction essaye d'ouvrir le premier fichier de la listeself.

    ATTENTION: Si la fonction renvoie False, cela peut également être dû
    à une mauvaise variable nom_fichier
```

Facile non ?

## Un premier exemple: Essayez tous les nombres possibles

- Imaginons que le mot de passe soit simplement un nombre entre 1 et 10.000. Comment peut-t-on le tester en python ?
- Imaginons que le mot de passe soit juste un nombre, mais qu'on veut pouvoir tester n'importe quel nombre, peu importe sa taille (y compris des nombres très grands) ?
- Et si on veut tester **uniquement** des nombres très grands ? (par exemple, supérieurs à 1.000.000)

## Tester tous les mots d'un dictionnaire d'attaque

L'une des alternatives à l'attaque par force brute est d'utiliser un **dictionnaire d'attaque**.

Concrètement on voudra:

- Lire toutes les lignes d'un fichier où sont écrits tous les mots de passe à tester
- Essayer toutes les combinaisons

Saurais-tu le faire ?

Rq: Dans [CHEMIN\_VER\_PACKAGE]/data/ tu peux trouver tout un tas de dictionnaires différents. Tu peux déjà commencer par "most\_common\_psw\_ever.txt" par exemple !

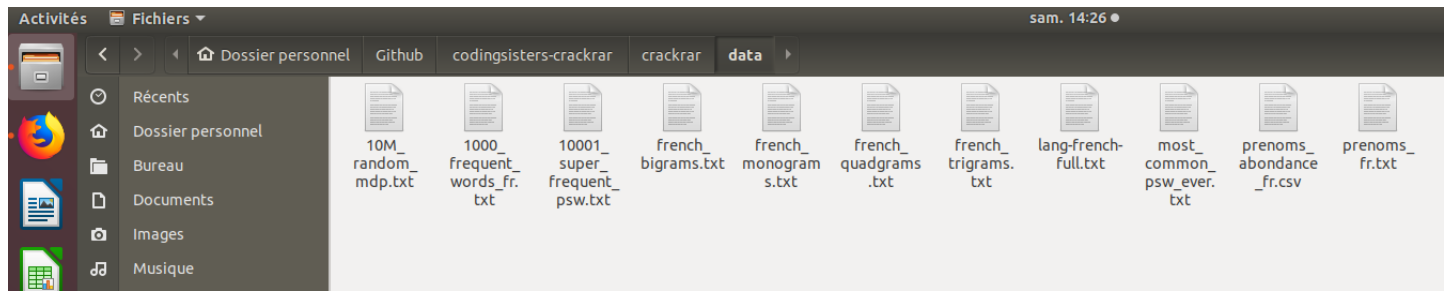
## Autre: Des exemples de dictionnaires

Dans [CHEMIN\_VERS\_PACKAGE]/crackrar/data/ on trouvera plusieurs dictionnaires au format texte (une ligne par mot):

- 10 millions de mots de passe aléatoires
- 1000 mots parmi les plus fréquents en français
- Une liste de 10.000 mots de passe parmi les plus utilisés au monde
- Un dictionnaire de 330.000 mots français

- Les 75 mots de passe les plus fréquents de tous les temps (pas nécessairement dans l'ordre)
- Une liste de prénoms français (en majuscules)
- Un .csv qui reprend ces prénoms mais permet aussi de les classer par abondance

Cela permet de réaliser des attaques par dictionnaire



Il y a de quoi faire !

On trouve aussi: **Les mono, bi, tri et quadgrams de la langue française avec leur abondance**

Cela permet de générer des mots qui pourront **"sonner français"** sans pour autant être des mots de français

## Voilà !

C'est fini pour la partie 1. J'espère que tu as appris des choses, et que tu n'hésiteras pas à demander aux tutrices si tu veux de l'aide, si tu n'as pas bien compris quelque chose, ou même si tu as des idées ou des projets que tu voudrais partager à partir de ce que tu as appris !

Quand tu penseras avoir fait le tour, je t'invite à passer à la partie 2 :)