DOCUMENTATION
Ignis-Promethei

Par
Grégory SCHAAF

SOMMAIRE

Introduction

Fonctionnement

Ce programme permet de casser des noms d'utilisateurs et des mots de passe par force brute en réorganisant un ensemble d'éléments de façon arrangée, permutée, combinée avec répétition et combinée sans répétition.

En permettant de choisir l'odre de génération des résultats ce programme permet à l'utilisateur une créativité et une personnalisation sans limite. Chaque numéro d'ordre est converti en son équivalent sous formes de mot final via la liste d'éléments déclarée.

Il est également possible de déclarer de façon parallèle différentes configurations de génération pour générer plusieurs mots et ainsi les assembler pour former un résultat final qui pourra être testé comme nom d'utilisateur ou mot de passe (concaténation de résultats de générations).

La dernière possibilité de personnalisation est la réorganisation des éléments composant un mot généré.

Plusieurs autres paramètres de génération peuvent être utilisés pour définir un temps entre chaque génération ou choisir le type de rendu visuel.

Introduction à Promethei

Fonctionnement

Promethei permet de créer la stucture permettant la création de la génération et sa personnalisation. Tous les paramètres indispensables sont contenus dans le répértoire de configuration créé par ce programme.

Les paramètres “utilisateur”, “mot de passe” et “authentification” doivent obligatoirement être déclarés.

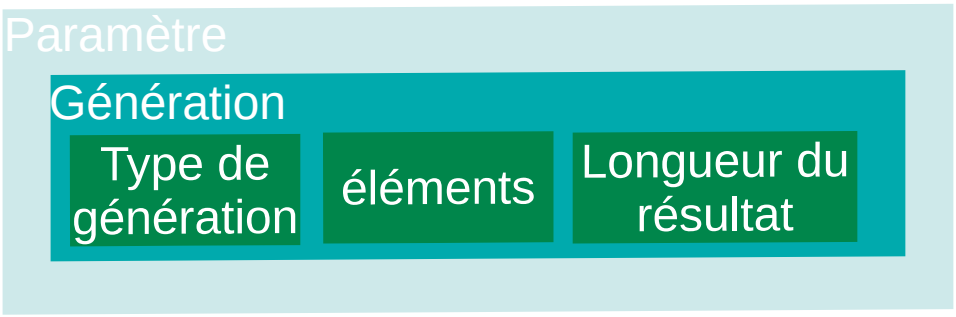
1.1

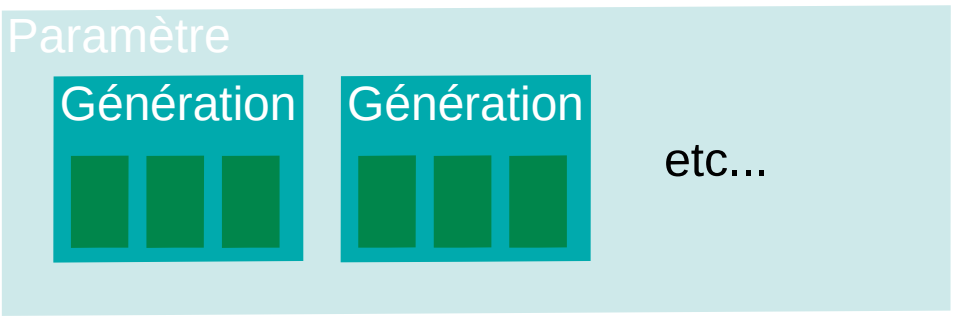
Utilisateur -u et mot de passe -p

Fonctionnement

Ces deux paramètres sont identiques dans la manière de les utiliser.
Pour déclarer une génération, un type de génération, des éléments et une taille sont obligatoires.

Structure





1.1

Utilisateur -u et mot de passe -p

Types de génération

- a= (arrangement)
- p= (permutation)
- C= (combinaison avec répétition)
- c= (combinaison sans répétition)
- f= (fixe)

Pour le type fixe, il n'est possible de déclarer qu'un seul élément.

Eléments

Deux types d'éléments peuvent être déclarés: les mots et les dictionnaires.

Un mot peut avoir une longueur minimum de 1 caractère et peut contenir tout caractère compris par le terminal.

Exemples:
- 1
- a
- Mot1
- testD/
- %32Jd
- etc...

Un dictionnaire contient le chemin du fichier contenant une liste de mots ainsi que les lignes début et de fin qui doivent être considérés.

Exemples:
- Repertoire/fichier.txt[:]
- Repertoire/fichier.txt[:10]
- Repertoire/fichier.txt[100:]
- Repertoire/fichier.txt[10:1]

1.1

Utilisateur -u et mot de passe -p

Déclaration finale

```
python3 promethei.py -u f=admin,1 -p a=dictionnaire.txt[:],mot1,mot2,mot3,mot4,5
```

```
python3 promethei.py -u f=admin,1 -p f=_test_,1+a=dictionnaire.txt[:],mot1,1
```

```
python3 promethei.py -u f=admin,1 -p f=_test_,1+a=test,dictionnaire.txt[:],?,/,teSt4,7,/1/2/4/3
```

1.2

Authentification -a

Introduction

Un type d'authentification doit être déclaré. Il permet de tenter de casser des hash ou de générer des dictionnaires de mots.

Types d'authentification

md5, sha1, sha224, sha256, sha384, sha512, blake2b, blake2s, sha3_224, sha3_256, sha3_384, sha3_512 et dico.

Concernant les hash le nom d'utilisateur correspond au mot crypté et le mot de passe à son potentiel équivalent. Le résultat final généré en tant que mot de passe est chiffré pour être comparé au mot crypté associé au nom d'utilisateur

Pour *dico* le nom d'utilisateur est le fichier qui sera créé pour contenir chaque résultat final de génération en tant que mot de passe.

Déclaration finale

```
python3 promethei.py -u f=098f6bcd4621d373cade4e832627b4f6,1
 -p a=a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,4 -a md5
```

```
python3 promethei.py -u f=dico.txt[1],1 -p a=dictionnaire_mdp.txt[:],test.txt[2:],10
 -a dico
```

1.3

Répertoire de configuration -c

Fonctionnement

Le nom du répertoire de configuration peut être choisit. Il ne doit contenir aucun “/”.

Exemple

```
python3 ignis.py -c rep_config
```

II

Introduction à Ignis

Fonctionnement

Ignis permet d'utiliser les paramètres inscrits dans le répértoire de configuration pour tenter de casser un hash ou de générer un dictionnaire.

Le paramètre permettant de déclarer le répertoire de configuration est obligatoire.

2.1

Répertoire de configuration -c

Fonctionnement

Le chemin du répértoire de configuration doit être déclaré.

Exemple

```
python3 ignis.py -c rep_config
```

2.1

Répertoire de configuration -c

Introduction

S'il n'y a aucune erreur le répertoire est créé. Il devra être utilisé par le programme **Ignis**.

Fichiers principaux

- config contient les paramètres déclarés comme -u, -p et -a.
- result.txt contient le jour, le mois, l'année, l'heure, les minutes et les secondes où une authentification a réussie ainsi que le nom d'utilisateur, le mot de passe et le type d'authentification.

Répertoire u et p

Un répertoire est créé pour chaque paramètre -u et -p contenant un fichier .g pour chaque paramétrage de génération qui contiennent les numéros d'ordres de chaque résultat de génération. C'est dans ces fichiers qu'il est possible de choisir les résultats générés en fonction de leur ordre logique de génération.

Par défaut, chaque fichier contient toutes les générations possibles sous forme d'intervalle ou de simple numéro d'ordre si une génération fixe est concernée.

Il est possible d'inscrire sur une même ligne des intervalles croissant, des intervalles décroissant et des nombres fixes séparés de virgules.

Exemple:

rep_config/p/1.g → 1:10000,100:3,10,30:35
1,10
103:1004

Il y a un total de 10106 générations à la première ligne, 2 générations à la deuxième ligne et 902 générations à la troisième ligne.

Regrouper le plus de générations sur une seule ligne permet de réduire le nombre de lecture du fichier et gagner du temps pour en contrepartie prendre de l'espace dans la mémoire volatile. Disperser les générations sur plusieurs lignes permet d'économiser de l'espace dans la mémoire volatile pour en contrepartie prendre du temps à lire plusieurs fois le fichier.

2.2

Rendu visuel -r

Fonctionnement

Le rendu visuel pemet d'afficher le suivi des tentatives de cassage. Il existe 2 types de rendu visuel : sous forme de chargement (%) et ligne par ligne (/). Une fois le type de rendu visuel déclaré un nombre de rendu peut être déclaré. Le nombre de tentatives sera visuellement mis à jour au nombre déclaré. S'il y a 100000 possibilités existantes pour 10000 affichages alors l'affichage se mettra à jour toutes les 10 tentatives.

Exemple

```
python3 ignis.py -c rep_config -r /1000
```

2.3

Le temps -t

Fonctionnement

Déclarer un temps permet d'imposer une pause entre chaque tentative. Déclarer 0.5 impose une pause d'une demi seconde entre chaque tentative.

Exemple

```
python3 ignis.py -c rep_config -r %1001 -t 0.89
```

Commandes à tester

Exemples

```
python3 promethei.py -u f=test.txt,1 -p p=0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,4,/1/2/3/4+f=__fin_.,1 -a dico -c rep_config
```

```
python3 ignis.py -c rep_config -r /
```

```
python3 ignis.py -c rep_config -r %
```

```
python3 ignis.py -c rep_config -r /15
```

```
python3 ignis.py -c rep_config -r %15 -t 0.1
```

```
python3 ignis.py -c rep_config -r %f
```