Compte Rendu Projet AP2

Membres du binôme :

Ce projet a été fait par 2 étudiants :

Mirghane Mohamed Kamardine et Raffin Matteo

Mode D'emploi :

Pour lancer le programme, on lance le fichier « launcher.ml » et l'on exécute la fonction « launch », la fonction renverra automatiquement les données en analysant les fichiers .txt se trouvant dans le sous répertoire « Data ». Cette fonction ne prend rien en paramètre. Nous avons aussi une fonction « launch _2 » qui elle prend en paramètre un fichier sous la forme de string qui consiste en un fichier de mot de passe en clair.

Réponses aux Questions :

Q1 : est-il raisonnable de tenter une attaque en force brute, c'est à dire de tester tous les mots de passe possibles ?

R : Non ce n'est pas raisonnable, on a 126-33=93 caractères possibles pour chaque caractère possible dans le mot de passe qui peut varier de 6 à 128 caractères ce qui veut dire qu'on a entre 93^6=6e+11 et 93^128=9e+251 possibilités, ce qui prendrait un temps inimaginable à faire sachant que l'on doit traiter plusieurs fichiers d'environ 20000 lignes.

Q2 : Sinon, peut-on tenter des attaques en force brute partielles ? c'est à dire tester un sous-ensemble de mots générés automatiquement, et si oui quel sous-ensemble ?

R : On peut alors tenter une attaque en brute force partielle. On a notre disposition le fichier french_passwords_top20000.txt qui contient les mots de passe les plus couramment utilisé, ce qui veut dire que si on utilise une brute force partielles grâce à ce fichier, on pourra possiblement récupérer plusieurs mots de passe voir tous dans le meilleur des cas.

Q3 : Existe-t-il d'autres pistes pour casser tout ou partie des mots de passe avec les données dont il dispose ?

R : On a aussi des fichiers de données en clair, on pourrait aussi s'en servir afin de récupérer d'autres mot de passe possibles pour les utiliser dans un décryptage par brute force partielle, ce qui aura 2 avantages :

- pouvoir décrypter des mots de passe.
- réduire les cas possibles pour une attaque en force brute.

Ressources :

Dans cette section vous trouverez les liens des sites et aides qui nous ont en parties aider pour la conception de ce programme :

Site 1 : https://itsocial.fr/contenus/articles-decideurs/la-robustesse-des-mots-de-passe-est-toujours-autant-negligee/, pour répondre au question du hacker.

Site 2 : https://ocaml.org/docs/file-manipulation pour chercher comment manipuler des fichiers (il s'est avéré inutile...)

Site 3 : https://ocaml.org/manual/5.2/api/Sys.html , pour comprendre Sys.readdir qui permet de lire un répertoire.

Site 4 : https://ocaml.org/manual/5.2/api/List.html

Site 5 : ChatGPT, qui nous a permis de trouver les noms pour certaines fonctions dont nous n'avions pas d'idée de nom.

Présentation du répertoire :

Dans le répertoire principal : **kmirghan_mraffi04**, nous avons :

Sous répertoire « src » :

Le sous répertoire « src » contient les fichiers comportant le code source ocaml de notre projet, les fonctions sont reparties dans différentes fichiers.

Fichier « main.ml » :

Ce fichier ocaml comporte les fonctions principales ainsi que leurs code source qui nous permettrons de créer le programme de hack.

Fichier « Utils.ml » :

Ce fichier ocaml comporte des fonctions « utiles » qui permettent de faire des sous opérations (par exemple, vérifier si un élément est présent dans une liste, transformer liste en tableau, obtenir la liste des applications piratées, etc...) qui sont utiles pour coder les fonctions du fichier « main.ml ».

Fichier « tri.ml » :

Dedans nous avons implémenter en code source ocaml l'algorithme du tri par insertion, ainsi que toute fonction qui seront utiles à son bon fonctionnement.

Fichier « tools.ml » :

Ce fichier contient une fonction qui nous permet de lire un fichier .txt, ainsi que d'une autre fonction qui permet de hasher une chaine de caractère.

Fichier « liste.ml » :

Dans ce fichier vous retrouverez l'implémentation en code ocaml , des fonctions qui permettent de manipuler des listes (renverser une liste, concaténer,...).

Sous répertoire « Data » :

C'est dans ce répertoire que nous trouvons les fichiers .txt comportant les données des fuites des applications.

Sous répertoire « miscelleanous » :

Dans ce répertoire, nous aurons le sujet.

Code source Ocaml

Code du fichier « tools.ml »

Dans ce fichier (fournit initialement) nous avons le code des fonctions permettant de lire un fichier .txt, et une fonction permettant de hasher un mot de passe.

Code du fichier « main.ml »

C'est dans ce fichier que nous retrouvons les fonctions principales, d'où le nom 'main' pour le fichier.

Fonction « insert_data_in_var »

Comme son nom l'indique, cette fonction insère une donnée {login ; mot-de-passe} dans une variable, cette fonction prend en paramètre un nom de fichier, et une variable (vide ou non). La fonction possède une fonction auxiliaire récursive pour permettre d'insérer toutes les données de fuites d'une application dans une variable. La fonction auxiliaire de la fonction, s'aide de la fonction « is_login_in_lists » qui lui permettra de vérifier si un login est déjà dans la liste, pour pas la mettre et éviter ainsi les doublons.

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace si jamais la liste en entrée contient 1 seul élément sinon $O(n^2)$. Pour la complexité en pire, nous avons O (n^2) en temps et $O(n^2)$ en espace.

Fonction « fusion_files »

Cette fonction permet de fusionner les données des fichiers qui sont données en paramètre, dont celui-ci une liste contenant le nom des fichiers (exemple : ["slogram01.txt " ; "slogram02.txt"]).

La fonction possède une fonction auxiliaire récursive pour permettre d'insérer les fichiers successivement, elle se sert de la fonction « insert_data_in_var » pour insérer les données d'un fichier dans la variable finale.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « fusion_files_of_app »

Comme son l'indique, la fonction fusionne les données des fichiers d'une application dans une seul et même variable. La fonction prend en paramètre le nom de l'application (exemple : "slogram") sans le numéro de la fuite ni l'extension .txt.

La fonction se sert de la fonction fusion_files qui est appelée avec une fonction " transform_tab_to_list " qui transforme un tableau en liste le tableau contenant les fichiers les fichiers .txt correspondants aux fuites d'une application, récupéré par la fonction "insert_n_file_in_tab" qui insères dans un tableau le nom des fichiers de l'application dont le nom est donné en paramètre dans la fonction "fusion_files_of_app".

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (n) en espace.

Fonction « merge_app »

Fonction qui fusionne toutes les données de toutes les applications dans une seule et même variable. La fonction prend une liste de noms d'application et une variable dans laquelle on mettra les informations à l'intérieur.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (n) en espace.

Fonction « check_login_across_breachs »

Cette Fonction regarde si un login est présent au moins une fois dans plusieurs bases de données et détermine si les mots de passe sont identiques.

La fonction s'aide de sa fonction auxiliaire récursive check_login_across_breachs_aux qui prend en paramètre le login, la variable qui contiendra le résultat final, et une liste d'application. La fonction va checker récursivement pour chaque fuite dans une application qui est présente dans la liste d'application donnée en paramètre si le login si le présent dans une application et va renvoyer son couple (application, mot_de_passe).

La fonction vérifie si le login est bien dans la variable merge_app_file qui est constitué des login et mot de passe de cette application, si c'est le cas alors elle récupère les positions des endroits où elle a trouvé ces logins pour ensuite ajouter leurs mots de passe au résultat, ainsi que le nom de l'application dans lequel elle regarde afin de savoir d'où vient ce mot de passe.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire si un fichier possède le même mot de passe pour toutes les données, nous avons O (n^3) en temps et O (n)en espace.

Fonction « launch_clab »

Fonction qui permet de lancer la fonction « check_login_across_breachs » mais en prenant en paramètre que seulement une chaine de caractère qui est le login. Cela permet notamment lors de test de pouvoir essayer sans avoir à décrire tous les paramètres, elle ne sera d'ailleurs pas beaucoup utilisée car nous aurons besoin de l'autre fonction dans la suite.

Ses complexités sont pareilles que celle de « check_login_across_breachs ».

Fonction « check_pwd_accross_breachs »

Cette fonction est pareille « check_login_across_breachs » mais analyse le mot de passe donné en paramètre et renverra le couple (application, login).

La fonction s'aide de sa fonction auxiliaire récursive check_pwd_accross_breachs qui prend en paramètre, le mot de passe, la variable, la liste d'application, et une variable contenant les données.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n^ 3) en temps et O (n) en espace (pareille pour « check_login_across_breachs »).

Fonction « launch_cpab »

Fonction qui permet de lancer la fonction « check_pwd_across_breachs » mais en prenant en paramètre que seulement une chaine de caractère qui est le mot de passe. Cela permet notamment lors de test de pouvoir essayer sans avoir à décrire tous les paramètres, elle ne sera d'ailleurs pas beaucoup utilisée car nous aurons besoin de l'autre fonction dans la suite.

Sa complexité est la même que celle de « check_pwd_across_breachs ».

Fonction « matching_pwd_with_app_log »

Comme son nom l'indique la fonction donne un triplet contenant le mot de passe, l'application et le login.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (n) en espace.

Fonction « get_couple_app_log »

Fonction qui me retourne le couple (application,login).

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « launch_mpwal »

Fonction qui permet de lancer la fonction « matching_pwd_with_app_log » mais en prenant en paramètre que seulement une chaine de caractère qui est le mot de passe. Cela permet notamment lors de test de pouvoir essayer sans avoir à décrire tous les paramètres, elle ne sera d'ailleurs pas beaucoup utilisée car nous aurons besoin de l'autre fonction dans la suite.

Sa complexité est la même que celle de « matching_pwd_with_app_log ».

Code du fichier « Utils.ml »

Dans ce fichier nous commençons par la déclaration du type pour représenter une donnée d'utilisateur représenté par le login qui est une chaine de caractère et le mot de passe qui est lui aussi une chaine de caractère.

Fonction « is_txt_file »

Comme son nom l'indique, cette fonction regarde si le fichier donnée en paramètre par son nom complet (càd *nomfichiernumérofuite.extention*) sous forme de chaine de caractère est un fichier txt ou non.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (1) en temps et O (1) en espace.

Fonction « is_data_in_list »

Cette fonction regarde si une donnée d'utilisateur est présente dans une liste de donnée d'utilisateur. La fonction prend en paramètre la liste, ainsi que la donnée (couple (login,motdepasse)).
Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n) et O (n) en espace.

Fonction « is_in_list »

Cette fonction est pareille que la précédente sauf qu'elle est un peu plus général, c'est surtout pour un élément qui n'est pas une donnée d'utilisateur.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « number_of_file »

La fonction permet de connaitre le nombre de fichiers d'une application qui a fuité (donc aussi le nombre de fois qu'il a fuité c'est obvious).
La fonction prend juste en paramètre le nom d'une application.

Comment fonctionne – t -elle ? Nous avons une variable locale « dir » qui stocke un tableau trié contenant tous les fichiers du répertoire « Data » (contenant tous les fuites). « Sys.readdir("Répertoire ") » est une fonction du module "Sys" d'Ocaml, Son problème est qu'il donne un tableau de fichiers présent dans le répertoire ciblé mais d'un ordre aléatoire, donc nous les trions avec un « tri_insertion » pour les avoir dans l'ordre alphabétique.

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (n) en espace.

Fonction « insert_n_file_in_tab »

La fonction insère les fichiers d'une application dans un tableau, en connaissant le nombre de fuites (on peut l'avoir en utilisant la fonction précédente).

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (n) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « transform_tab_to_list »

Comme son nom l'indique la fonction permet de transformer un tableau en liste de longueur n.

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « recup_all_txt »

Une fonction qui récupère tous les fichiers avec l'extension .txt dans le répertoire « Data ». La fonction s'aide de sa fonction auxiliaire récursive recup_all_txt_aux qui elle insères dans une variable donnée en paramètre un fichier si celui-ci est un fichier .txt.

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « is_login_in_lists »

Une fonction qui regarde si un login donné est présent dans une liste de données d'utilisateur, alors elle ressemble à « is_in_list » et à « is_data_in_list » mais l'avantage c'est qu'elle check que le login alors que « is_data_in_list » check le couple(login,motdepasse). La fonction s'aide de sa version auxiliaire récursive qui elle prend en paramètre le login, la liste de donné et un compteur, et si à la fin dépasse 1 alors le login est présent.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (1) en espace.

Fonction « pwd_of »

La fonction prend en paramètre un login et renvoie le **dernier** mot de passe trouvé associé à ce login dans une liste de base de données d'utilisateur, elle aussi donnée en paramètre. Si le mot de passe est introuvable pour un login donné, alors on renvoie « pwd not found ».

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (1) en espace.

Fonction « get_pos_log »

Permet de récupérer la position de la première occurrence trouvée du login tout en renvoyant celle-ci et la liste des éléments non parcouru pour pouvoir ensuite rappeler cette fonction sur cette fois si les éléments non parcourus.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « correct_list »

Permet de renvoyer une liste avec les positions correctes des occurrences du login (entrée : [-1 ; 4 ; 4; 6], vrai position : [6; 11; 16])

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n^2) en espace.

Fonction « get_pos_log_list »

Renvoie la liste des positions des occurrences du login en ayant vérifié que ces positions soit cohérente. Etant donné que get_pos_log() renvoie la liste des éléments non parcourue et qu'on se sert de celle-ci pour pouvoir atteindre la fin de la liste, les positions sont donc réinitialiser à chaque fois, c'est pourquoi correct_list() les corrige afin que ces positions soient réellement les occurrences du login

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « is_pwd_in_lists »

Même fonctionnement que « is_login_is_lists » sauf quelle celle-ci prend en paramètre un mot de passe. La fonction possède aussi une fonction auxiliaire récursive qui « is_pwd_in_lists_aux » qui fonctionne de la même manière comme « is_login_in_lists_aux ».

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (1) en espace.

Fonction « login_of »

Pareillement pour cette fonction, elle possède le même système comme « pwd_of » mais prend en entré un mot de passe haché ou pas, ainsi qu'une base de données d'utilisateur et renvoie le **dernier** login associé trouvé. Si le login est introuvable pour un mot de passe donnée, alors on renvoie « login not found ».

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (1) en espace.

Fonction « get_pos_pwd »

Permet de récupérer la position de la première occurrence trouvée du mot de passe tout en renvoyant celle-ci et la liste des éléments non parcouru pour pouvoir ensuite rappeler cette fonction sur cette fois si les éléments non parcourus.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « get_pos_pwd_list »

Renvoie la liste des positions des occurences du mot de passe en ayant vérifié que ces positions soit cohérente. Etant donné que get_pos_log() renvoie la liste des éléments non parcourue et qu'on se sert de celle-ci pour pouvoir atteindre la fin de la liste, les positions sont donc réinitialiser à chaque fois, c'est pourquoi correct_list() les corrige afin que ces positions soient réellement les occurrences du login.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « verif_doublon »

Une fonction qui permet de vérifier les doublons et les enlever à conditions que la liste donnée en entrée soit triée. Attention c'est fonction a été faite dans le « tas », pour être exécuté avec des listes avec moins d'éléments car elle est très couteuse en temps.

Fonction « add_char »

Fonction qui permet de parcourir un nom pour retirer le nombre de fuites et les .txt. Elle s'aide de son auxiliaire récursive « add_char_aux » qui utilise « Char.escaped » pour transformer un caractère en chaine de caractère.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (1) en temps et O (1) en espace.

Fonction « get_name_list »

Une fonction qui permet d'avoir une liste contenant les noms des applications qui ont fuités, elle ne prend rien en paramètre. Elle s'aide de sa fonction auxiliaire récursive « aux_get_name » qui prend en paramètre une liste d'application (au départ vide mais l'insère de nom d'application au fur et à mesure de son exécution et de ses appels).

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n) en temps et O (n) en espace.

Fonction « is_hashed »

Fonction qui prend en paramètre un mot de passe et détermine si celui est haché ou non.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (1) en temps et O (1) en espace.

Fonction « read_pwd_file »

Permet de parcourir un fichier de mot de passe et de renvoyer une liste contenant ces mots de passe, adaptation de la fonction read_data_from_file() qui nous as été fournis.

Fonction « init_list_clear_pwd »

Prends en paramètre un fichier et renvoie la liste de son contenu.

Utilise « read_pwd_file » pour le faire.

Code du fichier « tri.ml »

Fonction « tri_arr »

Fonction qui effectue un tri insertion sur un tableau donnée en entrée tout en ayant la possibilité de garder le tableau non triée grâce à une variable auxiliaire qui permet de ne pas réécrire sur le tableau d'entrée (contrairement à la fonction « tri_insertion » qui modifie le tableau en entrée). Cette fonction fait appel à « tri_insertion ».

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (n) en espace avec n la longueur du tableau.

Fonction « swap »

Fonction prenant en paramètre un tableau, un indice i et j, et permute les éléments à la i-eme et j-ieme place.

Pour la complexité en mieux de cette fonction nous avons O (1) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (1) en temps et O (1) en espace.

Fonction « tri_insertion »

Fonction qui tri par tri insertion un tableau.

Pour la complexité en mieux de cette fonction nous avons O (n) en temps et O (1) en espace. Pour la complexité en pire, nous avons O (n^2) en temps et O (1) en espace.

Répartition du travail

Pour la conception des fonctions , Kamardine a fait la partie 1 complète avec toutes les fonctions qui sont appellé dans « Utils.ml ». Ensuite pour la partie 2 et 3, les fonctions du fichier Main.ml ont été faites ensemble. Celles du fichier Utils.ml ont été faite par Kamardine sauf les get_pos et correct list qui ont été faite par Mattéo. La partie 4 et la partie finale ont été faite par Mattéo.