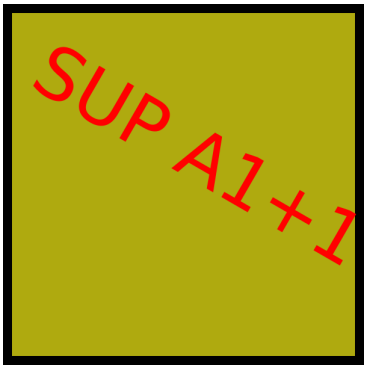

JeanTheRipper

Projet de S4

Cahier des Charges

Auteurs:
Simon "BarbeVerte" Guiot
Louis Guo
Marile "Low_battery" Lin
Kenny "Purple" Lorin

Vendredi 11 janvier 2019

Sommaire

1 Introduction

Ce cahier des charges est un rapport détaillant toutes les caractéristiques essentielles au bon développement de notre projet. Il sera notre support et un cahier de bord que l'on complétera au fur et à mesure de notre avancement.

Dans le cadre du projet de S4, nous allons vous proposer un projet qui s'orienterait plus sur la réalisation d'un logiciel qui, à la manière de JohnTheRipper ou de Hashcat, soit en rapport avec le cassage de mots de passe.

Ce document vous présentera dans un premier temps la partie descriptive du programme que nous souhaitons concrètement réaliser. C'est à dire les différentes méthodes de cassages de mots de passe, ainsi qu'une explication plus détaillée du hachage et du parallélisme.

Tandis que dans un second temps, on vous présentera les améliorations éventuelles que nous pourrions apporter, avec une possible réalisation technique que nous expliquerons, ainsi que ce que nous attendions personnellement de ce projet.

Plus loin, est ajouté un tableau regroupant les noms des membres du groupe chargés de toutes les tâches à réaliser et distribuées au sein de l'équipe. Cette partie comportera également un autre tableau qui exprimera l'avancement estimé, en pourcentage, des objectifs que nous aimerions atteindre durant les périodes entre deux soutenances.

2 État de l'art

2.1 Cassage de mots de passe

Pour commencer, nous pouvons citer le logiciel John The Ripper ainsi que Hashcat pour leurs fonctions permettant de craquer un mot de passe.

John The Ripper aussi surnommé JTR possède trois modes pour le cassage : le mode simple, l'attaque par dictionnaire et l'attaque par incrémentation.

En plus de JTR, Hashcat permet d'utiliser les techniques suivantes : l'attaque par masque, l'attaque hybride, attaque par règles et attaque par permutation. Certaines de ces techniques sont décrites plus bas.

Nous allons donc à présent définir les différentes méthodes de cassages. Il y a plusieurs méthodes de cassages de mot de passes et chacun possède ses avantages et inconvénients, nous allons nous concentrer essentiellement sur ces méthodes-ci :

- **Brute Force :**

La méthode par brute force, est la plus simple. Elle permet de générer toutes les possibilités combinatoires et de les tester une par une. Elle a donc un taux de réussite de 100%. Toutefois, elle est très coûteuse en termes de temps selon la rapidité de la machine utilisée, cela peut prendre jusqu'à des années pour craquer un mot de passe.

- **Dictionnaire :**

La méthode du dictionnaire permet de prendre en argument un fichier texte contenant une liste de mots afin de tous les tester un par un. Cette méthode ne permet donc pas de trouver un mot de passe de manière certaine mais peut, en revanche, être rapide à exécuter comparée à celle de brute force.

- **Masque :**

La méthode par masque, est une méthode qui permet de réduire considérablement les combinaisons possibles et donc le temps de cassage par rapport au brute force, en ajoutant des conditions. Néanmoins, elle nécessite de connaître des informations au préalable sur le mot de passe comme la taille du mot de passe, la position des majuscules etc.

- **Combinaison :**

La méthode par combinaison est une méthode plutôt efficace mais nécessite de connaître en amont des informations sur l'utilisateur comme sa date de naissance, son nom, son prénom, hobbies etc, qui sont mis dans un dictionnaire et combinées pour trouver le mot de passe, exemple : « louis29021999villejuifepitafootball ».

• **Rainbow table :**

Cette méthode consiste à hacher ou récupérer le hachage du plus de mots possibles et de les associer au clair dans une base de données. Lorsque l'attaquant voudrait trouver un mot de passe, il suffira de chercher le hash dans la base de données et récupérer le texte clair, s'il s'y trouve. Cette technique est donc extrêmement efficace pour les mots de passes simples ou très utilisés, car il n'y a pas de calcul, il suffit de faire la recherche.
L'inconvénient se situe dans le stockage de la base de données, qui fait souvent plusieurs dizaines de Giga Octets.
A cause de l'inconvénient de stockage nous ne comptons donc pas implémenter cette méthode.

Pour gagner en efficacité, le mélange de plusieurs de ces méthodes est envisageable.

Méthode	**Réussite**	**Temps**	**Conditions Requises**
Brute Force	100%	Très Long	-
Dictionnaire	Variable	Rapide	Dictionnaire de mots de passe
Masque	100%	Long	Données sur la structure du mot de passe
Combinaison	Variable	Moyen	Informations sur la cible
Rainbow Table	Variable	Rapide	Base de données importante

2.2 Hachage

Dans cette section, il s'agira de définir le hachage.

Une fonction de hachage transforme une donnée en une autre sans possibilité de retrouver la donnée de départ. Il s'agit dont de crypter les données.

Le principe d'une fonction de hachage est que la transformation de la donnée est rapide, unique pour chaque entrée. En réalité, créer une fonction de hachage parfaite est impossible : la plupart du temps, les fonctions de hachage peuvent engendrer des collisions.

Les fonctions de hachage modernes utilisent des opérations telles que des additions, des rotations, ou des opérations booléennes telles que XOR, OR, ou NOT. Ces opérations sont réalisées sur les bits de la donnée à crypter, et donc faciles et rapides à appliquer.

Les fonctions de hachage les plus populaires sont les suivantes :

• **md5**
• **sha-1**
• **sha-256**

Parlons par exemple de la fonction md5 :

Elle fonctionne selon un schéma assez simple, tout d'abord, on ajoute au début de la donnée à crypter afin d'avoir une longueur multiple d'un certain nombre (512bits), puis on applique le même processus à chaque bloc de 512bits de longueur, qu'on découpe en sous-blocs de 32 octets. On applique des fonctions à chacun de ces blocs, on les ajoute, les concatène, et voilà notre donnée de sortie.

Bien que cette fonction soit rapide à utiliser, il est très compliqué d'obtenir la donnée d'entrée en utilisant la force brute ; c'est le principe de Kerckhoffs. Cependant, cette fonction ayant été créée il y a un certain temps, il existe de nombreux sites permettant de trouver les possibles textes ayant servi à générer un hash. Il s'agit d'une des limitations de cette méthode. Elle est donc aujourd'hui de moins en moins utilisée.

La fonction sha-1 est considérée comme la deuxième plus utilisée, mais elle a récemment été déconseillée à cause de sa faible robustesse face à de grosses ressources.

La fonction sha-256 est-elle considérée comme robuste et efficace et est probablement la préférée aujourd'hui.

Nous comptons donc utiliser ces trois fonctions car ce sont les plus utilisées aujourd'hui dans les bases de données de mots de passe. Il en existe des centaines d'autres mais qui ne sont utilisées que dans des cas spécifiques (gestionnaires de mots de passe, mots de passe windows/linux etc) que nous ne comptons pas supporter en priorité.

2.3 Parallélisme

Avec le parallélisme, notamment le multithreading, nous pouvons distribuer la charge de travail sur plusieurs cœurs ou processeurs. Cela permet de grandement diviser le temps d'exécution de certains programmes : si l'on parvient à utiliser par exemple quatre processeurs en même temps au lieu d'un seul, on peut pratiquement diviser le temps d'exécution par quatre. Ceci est très efficace pour les calculs à répétition pendant de longues durées, et donc au cassage de mots de passe (qualifié également de "Embarrassingly parallel problem").

Néanmoins, le multithreading demande un agencement du code particulier dès le début.

Il existe plusieurs bibliothèques pour le multithreading. Nous nous concentrerons sur OpenMP et les threads POSIX. OpenMP est plutôt simple d'utilisation, puisqu'elle permet d'utiliser facilement les threads POSIX. Nous l'utiliserons donc au début pour avoir un code fonctionnel, puis nous essaierons de n'utiliser que les threads POSIX afin d'en apprendre un peu plus sur le multithreading. Il existe probablement d'autres bibliothèques permettant de faire du parallélisme, mais de cette façon nous pouvons utiliser les bibliothèques déjà présentes. La simplicité d'OpenMP permet en quelques lignes pour quatre coeurs de faire un quart des itérations d'une boucle for par coeur, alors que sans OpenMP il faut, par exemple, d'abord chercher

le nombre de processeurs afin de créer le nombre de threads adéquat, puis créer manuellement les threads et manipuler les données plus difficilement.

2.4 Potentielles améliorations

Nous avons plusieurs idées d'améliorations que nous pourrons réaliser après le programme principal. Tout au long du projet, nous comptons chercher activement à rendre les fonctions de chacun les plus rapides possible à l'exécution. Nous aurons également des dizaines de fonctions de hachage à étudier et potentiellement à implémenter en plus de celles que nous aurions déjà, ainsi que des fonctions permettant de manipuler les dictionnaires (suppression des doublons par exemple). Nous pourrions essayer de détecter automatiquement le type de hachage ou encore permettre l'ajout de sel. Nous ne comptons pas réaliser toutes ces idées, mais nous essaierons en fonction du temps qu'il nous restera.

2.5 La chaîne de Markov

L'utilisation de la chaîne de Markov a été implémentée à la fois dans Hashcat et dans John The Ripper. En utilisant cette option, on peut analyser un dictionnaire afin de déterminer les probabilités qu'une lettre soit suivie par une autre (ou dans Hashcat uniquement d'obtenir la probabilité qu'une lettre soit par exemple à la sixième position). Ceci permet de rendre l'attaque par brute-force bien plus efficace sur un grand nombre de mots de passe. Dans Hashcat et John The Ripper, on fait pratiquement deux fois moins d'essais avec cette option tout en cassant dix fois plus de mots de passe. Nous pensons donc que ceci peut être très intéressant à étudier mais également très compliqué. Ce ne sera donc pas une de nos priorités, mais cela ne nous empêchera d'essayer de la mettre en place.

2.6 Résultat attendu

Pour ce projet, nous sommes plus que déterminés à le travailler et à le terminer ensemble, tout en sachant que nous devrions faire face à certaines difficultés tout au long de sa réalisation.

Nous espérons obtenir à la fin un programme permettant de craquer un mot de passe avec, comme vous aurez pu le lire dans les parties précédentes, différentes manières de cassages. Nous pensons que l'utilisateur aimera avoir différentes manières en différents temps de craquer un mot de passe, pour mieux prendre connaissance lui aussi des diverses techniques utilisées pour le faire.

De plus, nous essaierons d'implémenter le plus de méthodes possible (principalement celles déjà décrites précédemment) permettant ainsi à notre projet d'être en constante évolution et qui peut donc toujours comporter des améliorations futures.

Comme demandé, nous aurons également un site web complet, sur lequel nous pourrons expliquer plus en détail les méthodes utilisées ainsi que nos éventuelles difficultés rencontrées lors de ces implémentations, avec également un moyen de tester notre programme.

3 Avancement et répartition du projet

3.1 Répartition des tâches

Voici la répartition des tâches. Nous avons essayé de les assigner aux membres du groupe de la façon la plus efficace possible en fonction des préférences et surtout des connaissances de chacun. D'autres tâches pourraient éventuellement être ajoutées au fur et à mesure du projet :

Tâche	**Simon**	**Louis**	**Marile**	**Kenny**
Parallélisme	⊕			+
Brute-force et masque	+	⊕		
Dictionnaire et combinaison	⊕		+	
Hachage				⊕
Améliorations	+	+	+	⊕
Site web			⊕	

Légende:

⊕ : Activité principale
\+ : Activité secondaire

3.2 Tableau d'avancement

Le tableau ci-dessous représente les tâches que nous espérons voir finies ou sur lesquelles nous souhaitons avancer pendant chaque semaine des prochains mois, jusqu'à la dernière soutenance.

Tâches \ Soutenances	Avant 1ère soutenance	Avant 2ème soutenance	Avant 3ème soutenance
Site Web	30%	60%	100%
Brute-force	100%	100%	100%
Attaque par masque	30%	100%	100%
Attaque par dictionnaire	100%	100%	100%
Attaque par combinaisons	80%	100%	100%
Parallélisme	15%	50%	80%
Hachage	60%	100%	100%
Améliorations	0%	15%	40%

4 Conclusion

On a tenté dans ce cahier des charges de vous présenter au mieux notre projet, c'est à dire tous ses aspects techniques et méthodologiques, ce que cela pourrait nous apporter professionnellement ou personnellement.

Il est évident que travailler sur ce projet sera pour nous une expérience très enrichissante. Ce travail nous permettra de compléter nos connaissances dans différents domaines (autant en programmation, en algorithmique ou encore en connaissances personnelles). Il nous plonge également dans les conditions de travail auxquelles nous seront confrontés à la fin de nos études, comme le travail en groupe, le respect des délais des dates limites de réalisations (soutenances) etc.

Et enfin, on essaiera lors de ce projet d'optimiser au mieux notre temps en partageant les tâches et en se concertant sur toutes les décisions importantes. On espère également que ce projet vous plaira autant qu'à nous !