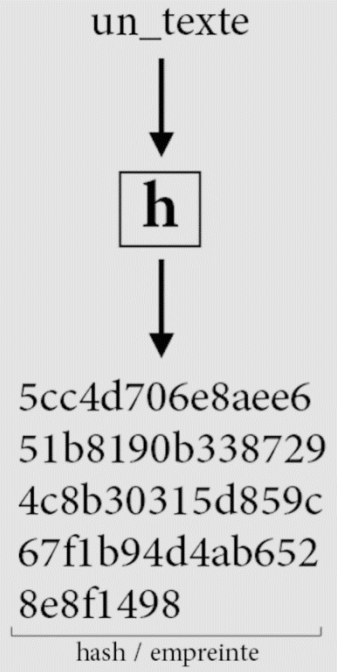
LABO1 – Hachage

1 LES FONCTION DE HACHAGE CRYPTOGRAPHIQUE



Les fonctions cryptographiques de hachage sont des primitives qui ont pour objectif de calculer une empreinte cryptographique à partir de données pouvant être de taille arbitraire. Cette empreinte a une taille fixe, par exemple 128, 160, 256 bits ou 512 bits, selon la fonction de hachage utilisée.[1] Remarquez qu'une fonction de hachage ne prend pas de clé cryptographique en paramètre !

Une fonction de hachage cryptographique a trois propriétés :

1. Il doit être impossible de recalculer le contenu d'un message qu'on a jamais vu à partir de son empreinte, c'est ce qu'on appelle **la résistance à la pré-image**.
2. À partir d'un message donné, de son empreinte et de son algorithme, il doit être impossible de générer un nouveau message qui donne la même empreinte, c'est ce qu'on appelle **la résistance à la seconde préimage**.
3. Enfin, il doit être impossible de trouver deux messages qui donnent la même empreinte, c'est ce qu'on appelle **la résistance aux collisions**.

Une fonction de hachage qui a ces trois propriétés est ce qu'on appelle **une fonction de hachage cryptographiquement sûre**.

Actuellement, les fonctions qui génèrent des empreintes de 128 et 160 bits sont à proscrire. Mais, elles sont encore largement déployées et utilisées.

Fonction	Longueur empreinte	Usage pour rétrocompatibilité	Usage pour applications futures
SHA-2	224-256-384-512	V	V
SHA-3	224-256-384-512	V	V
Whirlpool	512	V	V
RIPEMD-160	160	V	X
SHA-1	160	V	X
MD5	128	X	X
RIPEMD-128	128	X	X

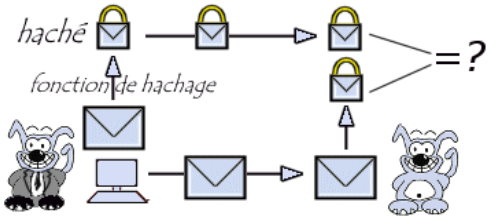
[1] Définition reprise de l'ouvrage "Sécurité Informatique (3e édition)" de Gildas Avoine – Pascal Junod – Philippe Oeschlin – Sylvain Pasini

1.1. Vérification d'intégrité et d'authenticité

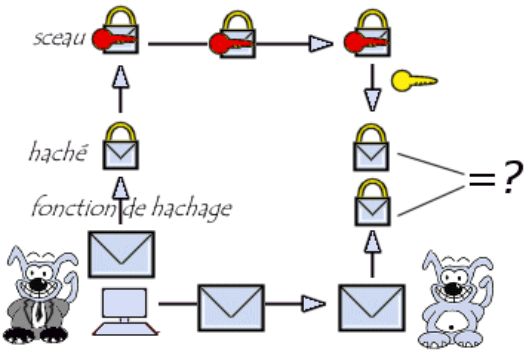
Les fonctions de hachage ont de nombreuses utilités en sécurité. Elles sont par exemple utilisées pour compresser de grandes quantités de données en une petite empreinte qui pourra être ensuite signée au moyen d'un algorithme de signature à clé publique. Un algorithme de signature étant plutôt lent, il est beaucoup plus efficace de signer l'empreinte que les données elles-mêmes.

L'intégrité :

Par exemple, visitez la page de téléchargement d'un système d'exploitation, vous constaterez qu'un checksum vous permet de vérifier l'empreinte numérique du fichier image (.iso) que vous téléchargez par une simple comparaison après son téléchargement.



L'authenticité :



Mais rien ne prouve que le fichier image et son empreinte n'ont pas été compromis. Ainsi, pour garantir l'authenticité du fichier, il suffit à l'émetteur de signer l'empreinte de hachage à l'aide de sa clé privée (l'empreinte signée est appelée **sceau**) et de rendre **sceau** disponible. Après téléchargement du fichier signé, il suffit de déchiffrer le sceau avec la clé publique de l'émetteur, puis de comparer l'empreinte obtenue avec la fonction de hachage du fichier téléchargé.

1.1.1 Exercice 1 – Vérifier l'intégrité et l'authenticité d'un fichiers

1. Sur votre VM Kali Linux, créez un dossier Security dans votre dossier personnel.
2. Dans ce nouveau dossier, téléchargez le fichier **SHA256SUMS** et le fichier **SHA256SUMS.gpg** sur la page de download de Kali Linux (http://cdimage.kali.org/kali-weekly/).
3. Déterminez le format de ces deux fichiers via la commande `file SHA256SUMS*`
4. Visualisé le contenu du fichier avec la commande `less SHA256SUMS`
5. Visualisé le contenu du fichier avec la commande `less SHA256SUMS.gpg`
6. La clé publique officielle de Kali peut maintenant être téléchargée comme suit[2] :

```
wget -q -O - https://www.kali.org/archive-key.asc | gpg --import
gpg --fingerprint 7D8D0BF6
pub rsa4096 2012-03-05 [SC] [expires: 2021-02-03]
44C6 513A 8E4F B3D3 0875 F758 ED44 4FF0 7D8D 0BF6
uid [ unknown] Kali Linux Repository <devel@kali.org>
sub rsa4096 2012-03-05 [E] [expires: 2021-02-03]
```

[2] Information reprise du site de téléchargement https://www.kali.org/downloads/

7. Vérifiez que le **fingerprint** affiché par cette dernière commande correspond à celui renseigné sur le site https://www.kali.org/downloads
8. Une fois les deux fichiers **SHA256SUMS** et **SHA256SUMS.gpg** téléchargés et la clé publique de Kali importée, vous pouvez vérifier la signature comme suit :
```
gpg --verify SHA256SUMS.gpg SHA256SUMS
gpg: Signature made Tue 13 Feb 2018 03:33:53 AM EST
gpg: using RSA key 44C6513A8E4FB3D30875F758ED444FF07D8D0BF6
gpg: Good signature from "Kali Linux Repository <devel@kali.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Primary key fingerprint: 44C6 513A 8E4F B3D3 0875 F758 ED44 4FF0 7D8D 0BF6
```
9. Si vous ne recevez pas le message "Good signature" ou si le **fingerprint** de la clé ne correspond pas, vous devez arrêter le processus et vérifier si vous avez téléchargé les fichiers à partir d'un miroir Kali légitime.
10. **OPTIONNEL** – Afin de ne pas surcharger la ligne internet inutilement, cette étape est juste présentée pour votre information. Placez un fichier .iso téléchargé sur le site Kali Linux dans le même dossier que vos deux fichiers **SHA256SUMS***.
Vérifiez que l'empreinte contenue dans le fichier **SHA256SUMS** et le hache de votre fichier iso correspondent via la commande :
```
sha256sum -c SHA256SUMS 2>&1 | grep OK
kali-linux-2020-W46-installer-netinst-amd64.iso: OK
```

Ces étapes vous permettent de garantir l'intégrité et l'authenticité d'un fichier téléchargé sur Internet. Cette procédure est généralement commune sur plusieurs distributions de logiciels ou système d'exploitation.

1.2. Les outils de hachage sur fichiers

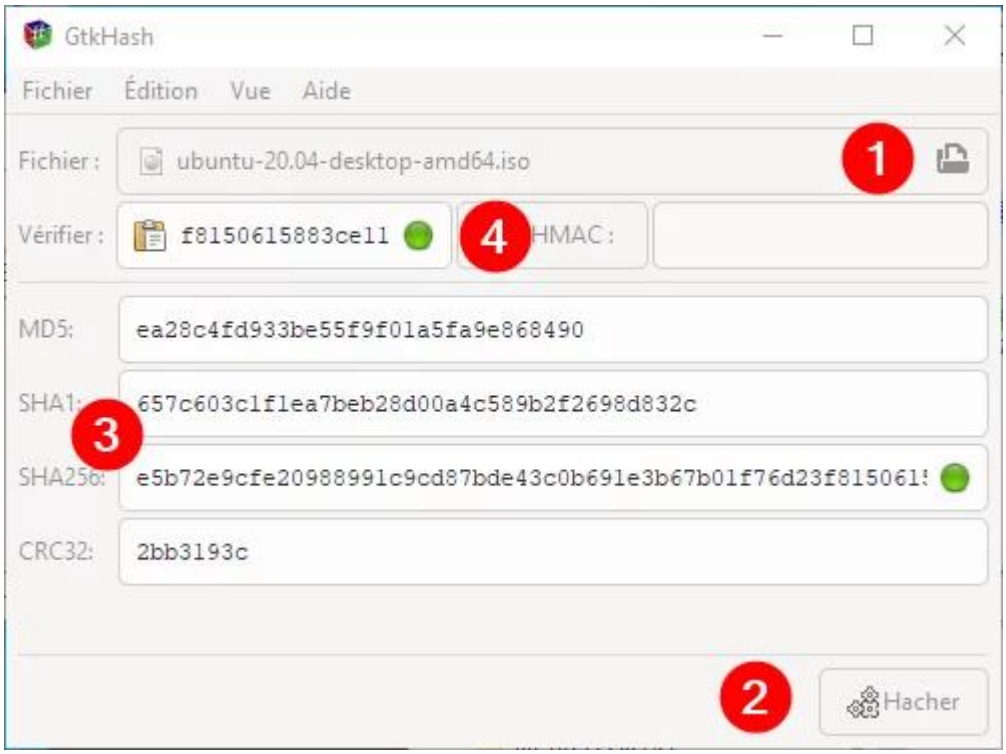
Différentes solutions existent pour obtenir l'empreintes numériques de fichiers informatiques.

1.2.1 Le logiciel Gtkhask

Gtkhask est un logiciel libre (Licence GPLv2) disponible sous Linux et Windows.

Il présente l'avantage de pouvoir comparer le résultat du calcul des empreintes d'un fichier avec une empreinte fournie. En fonction de votre architecture (32 bits ou 64 bits), vous pouvez télécharger un installeur ou un format portable depuis https://sourceforge.net/projects/gtkhash/.

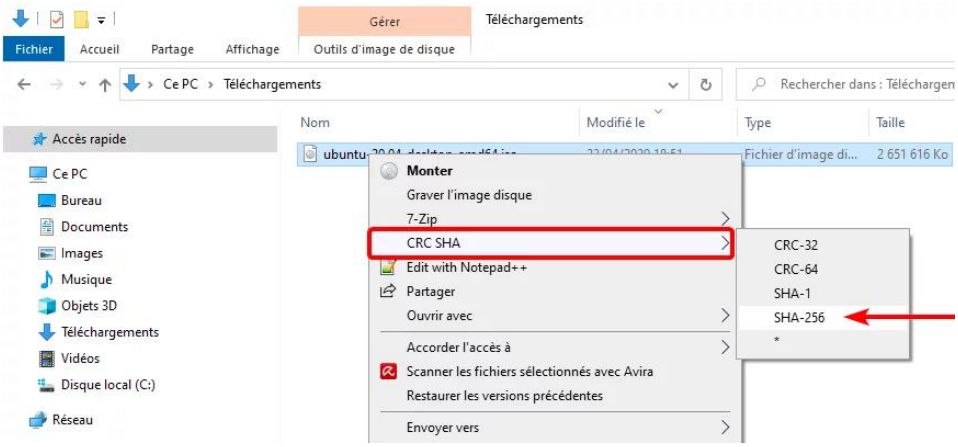
Dans son interface vous pourrez :



1. sélectionner un fichier
2. calculez les différentes empreintes d'un fichier
3. Voir le résultat du calcul
4. comparer ces résultats avec l'empreinte d'origine que vous aurez collé dans le champs. La pastille verte indique une comparaison réussie.

1.2.2 Le logiciel 7Zip

Si vous avez installé 7zip sur votre système, le logiciel intègre directement la fonctionnalité pour afficher les empreintes de hachage à partir de l'explorateur de fichier.



1.2.3 Nativement sous Windows

Nativement, il est possible d'obtenir l'information en ligne de commande :

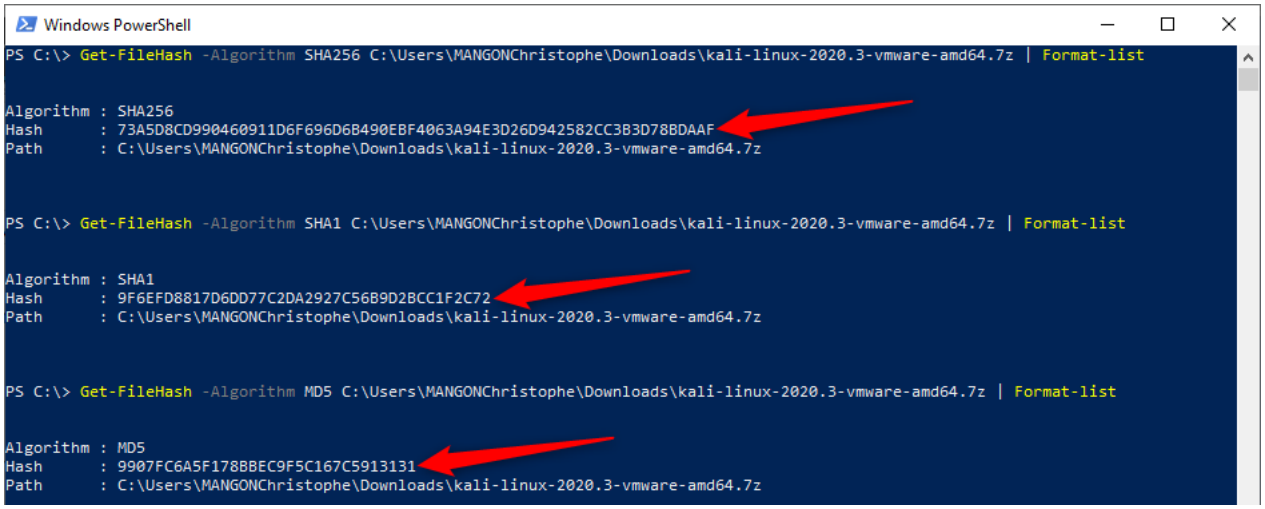
1. Ouvrez une invite PowerShell
2. puis faites :

```
Get-FileHash -Algorithm SHA256 chemin\vers\fichier_a_controler.ext | Format-List
```

chemin\vers\fichier_a_controler.ext est à remplacer par le chemin vers votre fichier. Exemple : `D:\Download\ubuntu-20.04-desktop-amd64.iso`

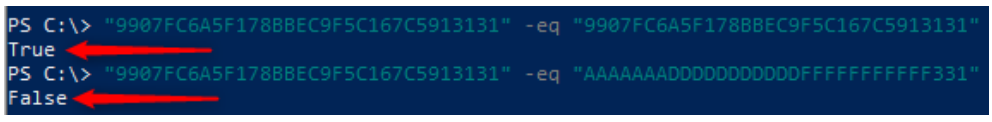
-Algotithm SHA256 indique l'algorithme utiliser pour le calcul de l'empreinte. Vous pouvez remplacer SHA256 par MD5, SHA1, SHA384, SHA512

| Format-List (facultatif) permet d'afficher les résultats sous forme de liste et non sur une ligne



Pour faciliter la comparaison de deux empreintes sous PowerShell, vous pouvez utiliser

```
"empreinte_calculee" -eq "empreinte_original"
```



1.2.4 Nativement sous Linux

Vous disposez des commandes qui vous permettent d'afficher le hash des fichiers passés en argument :

- **md5sum**
- **sha1sum**
- **sha224sum**
- **sha256sum**
- **sha384sum**
- **sha512sum**

Exemple :

```
root@kali:~# sha256sum /etc/passwd
593ad95a57a5352811e53a473beea2f793395d47b9f8707a0dee750acbfb7003  /etc/passwd
```

1.3. Utilisation des fonctions de hachage en Python

Python propose nativement un module **hashlib** permettant l'usage des fonctions de hachage. Visitez la page https://docs.python.org/fr/3/library/hashlib.html

Illustration permettant de vérifier l'empreinte MD5 d'un fichier ***hardcodé*** dans le script :

```
# Import hashlib library (md5 method is part of it)
import hashlib

# File to check
file_name = 'filename.exe'

# Correct original md5 goes here
original_md5 = '8d41402abc4b2a76b9719d911017c592'

# Open,close, read file and calculate MD5 on its contents
with open(file_name) as file_to_check:
    # read contents of the file
    data = file_to_check.read()
    # pipe contents of the file through
    md5_returned = hashlib.md5(data).hexdigest()

# Finally compare original MD5 with freshly calculated
if original_md5 == md5_returned:
    print "MD5 verified."
else:
    print "MD5 verification failed!."
```

1.3.1 Exercice 2 – hashsum.py

En python, développez un programme qui réponde aux spécifications suivantes :

SYNOPSIS

python hashsum.py [OPTION] filename

DESCRIPTION

Sans option, affiche les empreintes MD5, SHA1, SHA256 et SHA512 du fichier passé en argument.

-h
Affiche un résumé de l'usage du script

-a ou --algo {all,md5,sha1,sha256,sha512}
Précise la méthode de hachage à afficher ou à vérifier avec l'option -c

-c ou --check {hash}
Vérifie l'empreinte avec toutes les empreintes disponibles

Pour gérer le passage des arguments à votre script, utilisez le module **argparse** (https://docs.python.org/fr/3/library/argparse.html).

Illustration de l'usage pour l'option **-a ou --algo {all,md5,sha1,sha256,sha512}:**

```
>>> import argparse
>>> parser = argparse.ArgumentParser()
>>> parser.add_argument('-a', '--algo', default='all',
choices=['all','md5','sha1','sha256', 'sha512'])
_StoreAction(option_strings=['-a', '--algo'], dest='algo', nargs=None, const=None,
default='all', type=None, choices=['all', 'md5', 'sha1', 'sha256', 'sha512'],
help=None, metavar=None)
>>> parser.parse_args([])
Namespace(algo='all')
>>> parser.parse_args(['-a','sha1'])
Namespace(algo='sha1')
>>> parser.parse_args(['-a','md4'])
usage: [-h] [-a {all,md5,sha1,sha256,sha512}]
: error: argument -a/--algo: invalid choice: 'md4' (choose from 'all', 'md5', 'sha1',
'sha256')
```

1.4. Authentification par mot de passe

Les fonctions de hachage sont utilisées pour stocker les mots de passe informatiques.

Pour un site eCommerce, la gestion des comptes clients se fait par exemple via un couple login/mot de passe. Vous avez un compte, et votre mot de passe est stocké haché sur le serveur. Vous utilisez votre mot de passe habituel, **TrucMush2020**.

TrucMuch2020 → fonction de hachage SHA256 →
d15658de909d8a3dbe25244ff763928b60403aff6c0c878f056fcbe980a86aad
C'est le haché qui sera sauvegardé dans la base de données clients du site.

A chaque connexion sur ce site, le serveur va hacher le mot de passe que vous saisissez dans la fenêtre de connexion, et comparer le résultat à l'empreinte stockée lors de la création du compte. Si les deux sont identiques, le mot de passe saisi est validé.

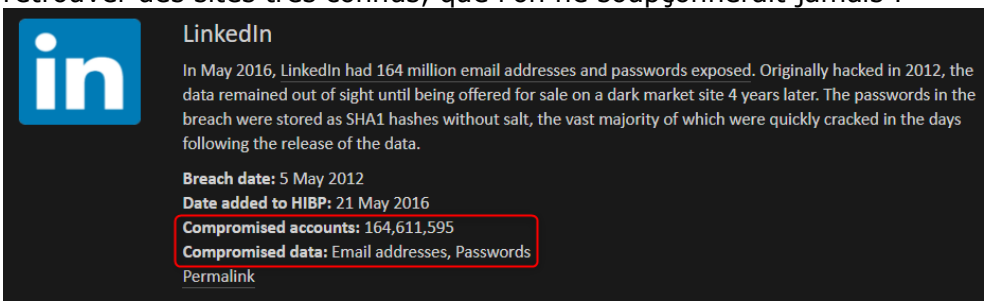
1.5. La mort des mots de passe, pourquoi ?

Extrait d'un des nombreux articles sur ce sujet encore d'actualité de nos jours : "*Les mots de passe traditionnels sont également vecteurs de problèmes, comme en attestent les nombreux vols d'ID, dont il apparaît qu'ils contribuent à près de la moitié de tous les cas de fraude recensés au 1er trimestre 2015.*" (https://www.strategies.fr/blogs-opinions/idees-tribunes/1048157W/la-mort-du-mot-de-passe.html - 03/10/2016 - par Patrick Salyer).

Pour illustrer ces propos, je vous invite à visiter deux sites qui référencent les fuites de bases de données :

- https://vigilante.pw/#breaches → Liste les bases de données, le nombre d'enregistrements concernés et la fonction de hachage de mots de passe contenus.

- https://haveibeenpwned.com/PwnedWebsites → Plus descriptif sur la fuite, ce site renseigne un historique et donne le détail du type de données fuitées. Vous pouvez y retrouver des sites très connus, que l'on ne soupçonnerait jamais !



Notez la possibilité de tester si vos mots de passe ont été exposés dans une des fuites référencées. MAIS, EN REGLE GENERALE, NE RENTREZ JAMAIS VOS CODES SECRETS SUR UN SITE DU GENRE ! Votre mot de passe et son empreinte risque d'être exposé.

Récupérer une base de données ne demande pas des connaissances très pointues ou de débourser des bitcoins sur le **Dark Web**. Une simple recherche sur Internet via un **Google Dorks**[3] peut nous donner accès à des milliers de bases de données. Essayez cette syntaxe sur le moteur de recherche **Google** afin de vous rendre compte des données directement disponibles sur Internet : **`intitle:index of "*db.tar.gz"`**[4]

1.6. Les attaques sur mot de passe hors-ligne (Offline Attacks)

Une fois la base de données récupérée, un pirate pourra retrouver votre mot de passe par différents moyens assez accessibles. Il lui faudra reconnaître la fonction de hachage utilisée. Mais tout le monde utilisent les mêmes donc ce n'est pas bien difficile. On peut facilement reconnaitre la fonction de hachage par l'empreinte.

1.6.1 Exercice 3 - Identifier le type de hache

Identifiez le type de hache pour les empreintes de mots de passe repris dans le tableau.

1. Par simple visualisation d'exemples d'empreintes associés à sa fonction de hache → https://hashcat.net/wiki/doku.php?id=example_hashes

2. Sur un système Kali Linux, utilisez le script **`hashid`** (**`hashid -h`** pour obtenir l'aide). Utilisez l'option **`-m`** pour obtenir l'identifiant du mode associé au hachage identifié (utile pour la suite de l'exercice).

[3] Voir annexe – QU'EST-CE QUE GOOGLE DORKS OU GOOGLE HACKING ?
[4] Toutes ne sont pas nécessairement exploitables. Cette information est à usage didactique !

Empreinte	Longueur	Hache
046e0196	**8**	
xoVnUO.uKmk	**11**	
3133d0480cef330cf55520b1c1566c6c		
PBZX1EG4rNTTC65x08EHRpHegxKSNOm.		
$1$$I9KdXSXfz2O4Oh4yoRDLK1		
e3df1dca78977b78c969856641c671d3f9b0774d		
d00862c7330b6dc9cbb5aafb3e2cb8ab6e71ae02b0dcfbccb8e66d33a5a9335407f1f0a584006740e6daef7a223af1f2		
e7bb750c6e11dd275866c4dad5dead5873221600f9856c3e2402e37d182552d8eac915556841ad167755573a9c0fa8323b31d4817fc1ea0a8eb23f7df277ba8a		

1.6.2 Exercice 4 – Casser du hache

Toutes les réponses seront dans la liste classique de mots de passe nativement installée sur Kali Linux **/usr/share/wordlists/rockyou.txt.gz** sous format compressé.

```
gunzip /usr/share/wordlists/rockyou.txt.gz
du -h /usr/share/wordlists/rockyou.txt
134M    /usr/share/wordlists/rockyou.txt
wc -l /usr/share/wordlists/rockyou.txt
14344392 /usr/share/wordlists/rockyou.txt
```

Soit un dictionnaire de 14.344.392 mots d'une taille de 134Mo.

Les premières questions sont assez facile à trouver à l'aide d'un outil de craquage en ligne tel que https://crackstation.net/

Indiquez le mot de passe pour les empreintes suivantes :

1. Hache: **48bb6e862e54f2a795ffc4e541caed4d**	
Mot de passe :	Type de hache :
2. Hache: **CBFDAC6008F9CAB4083784CBD1874F76618D2A97**	
Mot de passe :	Type de hache :
3. Hache: **1C8BFE8F801D79745C4631D09FFF36C82AA37FC4CCE4FC946683D7B336B63032**	
Mot de passe :	Type de hache :
4. Hache: **$2y$12$Dwt1BZj6pcyc3Dy1FWZ5ieeUznr71EeNkJkUlypTsgbX1H68wsRom**	
Mot de passe :	Type de hache :

Pour cette 4e empreinte, les choses se compliquent. Vous devez d'abord identifier le type d'empreinte via les procédés vus auparavant. Créez un fichier texte **hash.txt** sur votre machine Kali et copiez/collez le hachage entier dans ce fichier.

Utilisez l'outil **hashcat** pour briser le hache. (**hashcat -h** pour afficher l'aide).

Attention certaines fonctions de hachage prennent énormément de temps pour retourner l'empreinte. C'est une manière d'empêcher les attaques sur mot de passe.

- Précisez le temps estimé par **hashcat** en début d'exécution :
- Et le temps nécessaire pour votre système :
- Comparez votre durée pour trouver le mot de passe avec d'autres étudiants et expliquez :

5. Hache: **279412f945939ba78ce0758d3fd83daa**	
Mot de passe :	Type de hache :
6. Hache: **F09EDCB1FCEFC6DFB23DC3505A882655FF77375ED8AA2D1C13F640FCCC2D0C85**	
Mot de passe :	Type de hache :
7. Hache: **1DFECA0C002AE40B8619ECF94819CC1B**	
Mot de passe :	Type de hache :
8. Hache: **6aReallyHardSalt$6WKUTqzq.UQQmrm0p/T7MPpMbGNnzXPMAXi4bJMl9be.cfi3/qxIf.hsGpS41BqMhSrHVXgMpdjS6xeKZAs02.**	
Mot de passe :	Type de hache :

Pour cette 8e empreinte, vous devrez à nouveau identifier la fonction de hachage avant de pouvoir utiliser **hashcat**.
Notez que le hachage que nous avons collecté commence par $6, suivi d'un second $ et d'un troisième $. Il s'agit là d'un **salage**. Afin de compliquer la recherche de mot de passe, un sel est ajouté au mot de passe avant hachage. Reprenons l'exemple de tout à l'heure.

Génération de sel de manière aléatoire pour cet utilisateur : sel = **dac6595c04dda81**
TrucMuch2020+sel -> Fonction de hachage SHA256 ->
f1c829b4039db06ef077637b8c5c25544810c557b82d40c1e22c5f2cc2889b5e
Cette fois-ci le **haché** et le **sel** seront sauvegardés, et pas au même endroit si possible.

En quoi cela complique-t-il la tâche? Si le sel n'est pas connu, l'attaque par dictionnaire, brute-force devient impossible. Par contre, si les sels sont obtenus, retrouver les mots de passe se fera sans difficulté ! Voyez l'exercice sur la 8e empreinte.

9. Hache: **e5d8870e5bdd26602cab8dbe07a942c8669e56d6:tryhackme**	
Mot de passe :	Type de hache :

Parfois, l'identification du type de hachage n'est pas toujours précise, même avec les outils spécialisés. Il faut donc multiplier les sources de recherches. Essayez cette solution d'analyse en ligne https://www.tunnelsup.com/hash-analyzer/
Le hache est **HMAC-SHA1**, ce qui signifie que le mot **tryhackme** n'est pas réellement d'un **sel** mais d'une **clé**.
L'utilisation d'une **clé** est similaire à un **sel** sauf que la même **clé** sera utilisée pour tous les mots de passe et qu'elle sera conservée au secret, sans être sauvegardée avec l'empreinte (sauf pour cet exercice). Ca limite le risque de trouver la **clé** et donc retrouver tous les mots de passe. Si deux utilisateurs ont le même mot de passe, le hache sera identique et on pourra en déduire l'usage d'une **clé**. C'est pour cela que l'on préfère, quand c'est possible, mélanger de ces deux techniques.
Si nous n'aviez pas trouvé le type de hache, vous auriez pu le trouver par essais-erreurs, en utilisant quelques modes SHA-1 différents avec l'outils **hashcat**.

Ces exercices de craquages de mots de passe sont repris d'internet, voir page https://tryhackme.com/room/crackthehash.

1.6.3 Exercice 5 - Génération de fichier de mots de passe hachés

Pour crypter nos mots de passe, nous allons utiliser la commande **mkpasswd**
La syntaxe est la suivante : **mkpasswd [OPTION] MotDePasse**
L'option la plus intéressante dans notre cas est le choix de la méthode de chiffrement (**-m methode**). Pour connaitre les méthodes possibles, tapez **mkpasswd -m help**

Pour commencer, nous allons simplement tester en chiffrant le mot de passe **PASSWORD** en DES. **mkpasswd -m des PASSWORD** Le résultat affiché est le mot de passe chiffré.

REMARQUE : Si vous saisissez plusieurs fois la commande, vous constaterez que le résultat obtenu est différent à chaque fois. Pourtant, nous chiffrons toujours le même mot de passe (PASSWORD). L'explication est simple, le système utilise un "sel" ou "alea" pour obtenir un résultat "aléatoire" à chaque appel de la fonction.

Essayez avec : **mkpasswd -m des -S 10 PASSWORD**
Le résultat sera : **10uLxLRqvALKg** , l'option **-S** permet de forcer un **sel**.

Maintenant que nous savons générer rapidement des empreintes pour nos mots de passe, passons à la construction des fichiers à fournir pour les prochains exercices.
Rien de compliqué, il suffit de rediriger le résultat de la commande **mkpasswd** dans des fichiers.

```
mkpasswd -m des -S 10 PASS123 > passDES.txt
```

Pour ajouter d'autres mots de passe dans le fichier, utilisez la double-redirection :

```
mkpasswd -m des -S 10 ABCD208 >> passDES.txt
```

Le contenu de votre fichier **passDES.txt** doit ressembler à ceci :

```
105PoP0HFTShU
105vVeotb7U9A
```

Créez trois nouveaux fichiers nommés respectivement **passMD5.txt**, **pass256.txt** et **pass512.txt**.

Utilisez bien sûr les mêmes mots de passe à chaque fois (PASSWORD et 123456), mais avec un **sel** pour MD5 et SHA d'au moins 8 caractères. Prenez **12345678**.

A la fin de cet exercice, vous devez disposer de 4 fichiers contenant chacun le hache des deux mots de passe.

1.6.4 Exercice 6 – Construction d'un dictionnaire brute-force

L'attaque par force brute, ou **brute-force** consiste à hacher toutes les combinaisons possibles de lettres/chiffres/symboles : a, b, …, aaa, aab, aac, …, aba, abb, … baa… C'est très long mais comme la plupart des fonctions de hachage sont très rapides, c'est efficace sur des mots de passe courts (moins d'une minute avec un ordinateur de jeu pour calculer les hachés avec SHA1 de tous les mots de passes comprenant jusqu'à 6 caractères).

Pour cet exercice sous Kali Linux, vous devez construire les combinaisons de mots de passe avec une taille de **5 caractères** et avec le motif **[A-Z][a-z]{2}[0-9][!"#$%&'()*+,-./:;<=>?@[\]^_`{\|}~]**

Utilisez la commande **crunch** pour construire vos combinaisons dans le fichier **password.lst**.

Précisez la commande utilisée :

1.6.5 Exercice 7 – Attaque brute-force avec hashcat

Réalisez une attaque brute-force pour retrouver les deux mots de passe codés dans vos 4 fichiers créés à l'exercice précédent : **passDES.txt**, **passMD5.txt**, **pass256.txt** et **pass512.txt**. Utilisez la commande **hashcat** pour réaliser vos attaques.

Pour chaque fichiers, précisez l'instruction, ses paramètres et le temps nécessaire pour craquer le mot de passe. Attention, plusieurs manières existent pour réaliser cet exercice, certaines plus efficaces que d'autres. Pour plus d'efficacité, vous devez au minimum utiliser les masques en rapport à la composition des mots de passe à l'exercice 5. Cherchez la meilleure solution en vous aidant de la documentation (voir https://hashcat.net/wiki/)

La commande **hashcat** garde un historique des mots de passe dévoilés. Pour visualiser ces solutions en temps réel durant cet exercice, vous pouvez ouvrir un second terminal et lancer la commande **tail -f ~/.hashcat/hashcat.potfile**.

Instruction :
Durée :

Instruction :
Durée :

Instruction :
Durée :

Instruction :
Durée :

- Que pouvez-vous constater quant à la différence des durées d'attaque en relation au type de hachage à casser ?

1.6.6 Exercice 8 – L'attaque par dictionnaire et John the Ripper

Vous l'avez déjà utilisé précédemment et avez compris son principe. **L'attaque par dictionnaire** consiste à hacher pleins de "mots" d'un "dictionnaire" de mots couramment utilisé, des mots de la langue ciblée, des noms propres, etc. Les 500 mots de passe les plus courants sont utilisés par plus de 75% des utilisateurs.

L'on trouve de nombreux dictionnaires en tous genres sur la toile. Pour cet exercice, téléchargez le dictionnaire *Richelieu.*

```
wget https://raw.githubusercontent.com/tarraschk/richelieu/master/french_passwords_top20000.txt
```

Il s'agit d'une liste des mots de passe français les plus courants. Elle est basée sur des fuites de données filtrées afin de conserver les mots de passe liés à des adresses mail ".fr". L'objectif du projet *Richelieu* est de fournir un dictionnaire pour évaluer la sécurité de logins francophones par des administrateurs systèmes.

Vous pouvez parcourir ce fichier avec la commande less pour vous faire une idée des mots de passe francophones les plus couramment utilisés. J'espère que vos mots de passe n'y sont pas répertoriés (commande `grep "votre mot de passe" french_passwords_top20000.txt` (mais attention, vous allez afficher vos mots de passe en clair à l'écran et ils seront repris dans votre historique de commandes `~/.bash_history` 😊))

John the Ripper (ou JTR, ou John) est un logiciel utilisé notamment pour tester la sécurité d'un mot de passe qui fonctionne sur de multiples systèmes d'exploitation.

Comme pour l'exercice 5, générez à l'aide de la commande `mkpasswd` un fichier `johnHACHE.txt` contenant le hache `sha-512` avec le sel **12345678** pour le mot **astalavista**, **poussinette**.

John nécessite de lier les haches à un identifiant. Pour associer les mots de passe à des utilisateurs, il suffit de rajouter leurs nom en début de ligne (en utilisant nano) :

```
user1:$6$12345678$.....
user2:$6$12345678$.....
```

Mode simple :

Dans ce mode John enchaine les modes **`single`**, **`wordlist`** et **`incremental`**. Le fichier de mots utilisés par défaut est `/usr/share/john/password.lst`.

Il ne reste plus qu'à tester notre fichier avec `john` :

```
sudo john johnHACHE.txt
```

Avec des mots de passe simple, ce mode est très efficace. Pour autant que l'on ait un dictionnaire correspondant à la langue des utilisateurs. Le mode **`incremental`** est un mode

brute force. C'est le mode de craquage le plus puissant. Cependant, il est supposé qu'il ne se terminera jamais, le nombre de combinaisons étant trop grand ! Vous pouvez définir une limite de longueur de mot de passe faible ou préciser un petit jeu de caractères.

Vous pouvez stopper l'attaque par un CTRL+C. Sans quoi vous risquez d'y passer des jours.

Faite un `man john` pour visualiser l'aide. Recherchez comment reprendre l'exécution là où l'attaque a été interrompue et visualisez le fichier contenant le statut au moment de l'arrêt.

Exécutez l'attaque avec le dictionnaire des mots français Richelieu et précisez le temps nécessaire pour briser les mots de passe.

Comme pour la commande `hashcat`, la commande `john` garde une trace des mots de passe qu'elle a cassés dans un fichier `~/.john/john.pot`. Si vous ouvrez ce fichier vous devriez y voir les 2 mots de passe cassés précédemment.

Testez les mots de passe sur votre système Linux. Mais avant, vous devez consolider les deux fichiers `/etc/passwd` et `/etc/shadow`. Une fois l'exercice terminé, effacez le fichier de travail.

```
sudo unshadow /etc/passwd /etc/shadow > mypasswd.txt
sudo chmod 000 mypasswd.txt
```

2 ANNEXE

2.1. Qu'est-ce que Google Dorks ou Google Hacking ?

Google Dorks est l'ensemble des possibilités de recherches avec la syntaxe propre à Google, et qui permet de faire du hacking. Grâce à Google Dorks on utilise donc la puissance du moteur de recherche de Google comme un outil de hack, on parle donc de Google Hacking ou de Google Dorking.

Il est alors possible de :

- Trouver des informations sensibles (des emails stockés en ligne, des sauvegardes, des bases de données, des mots de passe, etc.).
- Trouver des pages d'authentification d'application web.
- Trouver du matériel connecté à Internet (caméras vidéo, routeurs internet, imprimantes, mais aussi du matériel industriel, etc.).
- Trouver des serveurs web mal configurés, ou installés par défaut.
- Trouver les technologies ou des outils sur le serveur web (phpmyadmin, phpinfo(), zones d'administration du site, etc.).
- Trouver des vulnérabilités qui permettent de pirater la cible.
- Et bien d'autres choses…