Bien commencer avec Hashcat

Hashcat est un outil de décryptage de mots de passe utilisant la technique dite de force brute. C'est à dire consistant à essayer toutes les combinaisons possibles jusqu'à trouver la bonne.

Voici un document explicatif de comment l'utiliser :

Préparation

Avant de commencer, il est nécessaire de connaître avec quel algorithme est haché le mot de passe en votre possession (c'est à dire pouvoir déterminer si on est en présence d'un hachage MD5, SHA-256, etc.)

Si celui-ci vous est inconnu, vous pouvez utiliser un outil comme Hash-Identifier (inclus dans Kali Linux) pour vous aider.

Dans la suite du document, nous admettrons que votre mot de passe haché est stocké bien au chaud dans le fichier `hash.txt`

L'attaque par dictionnaire

Quand on parle de casser des mots de passe, il nous vient souvent à l'esprit d'essayer les combinaisons les plus populaires comme `12345`, `azerty` ou encore `choucrouteLPB`.

Ce type d'attaque se nomme attaque par dictionnaire, et vous demande de ce fait de vous munir d'un fichier *dictionnaire*.

Les plus populaires sont rockyou.txt ou encore SecLists. Sous Kali Linux, vous en avez quelques uns à disposition dans le dossier `/usr/share/wordlists`.

Pour lancer une attaque par dictionnaire avec Hashcat, utilisez la commande ci-dessous complétée avec le hash-code correspondant à votre situation :

```
$ hashcat -m 100 -a 0 hash.txt dic.txt
```

- `-m 100` : Algorithme de hachage — Voir le tableau plus bas
- `-a 0` : Type d'attaque — 0 = Attaque par dictionnaire standard
- `hash.txt` : Fichier contenant le hash
- `dic.txt` : Dictionnaire à utiliser

Tableau des codes correspondant aux algorithmes de hachage populaires :

Algorithme de hachage du mot de passe	Code à utiliser après `-m`
MD5	`0`
SHA-1	`100`
SHA-256	`1400`
SHA-512	`1700`
GPG (AES-128/AES-256 × SHA-1)	`17010`
7-Zip	`11600`
PDF 1.7 Level 8 (Acrobat 10 – 11)	`10700`

Note : Vous pouvez toujours afficher la liste complète des algorithmes supportés et leur code avec la commande `hashcat -h`, section [Hash modes]

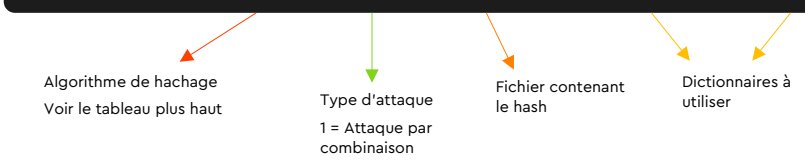
L'attaque par combinaison ⓘ

Dans la continuité de la première, cette seconde attaque va consister à non pas essayer chaque mot du dictionnaire un par un, mais les combiner deux par deux, puis trois par trois, etc.

Nous aurons donc besoin de fournir à l'outil non pas un, mais deux dictionnaires. Notez qu'il est parfaitement possible de lui fournir deux fois le même.

Pour lancer une attaque par combinaison avec Hashcat, utilisez la commande ci-dessous complétée avec le hash-code correspondant à votre situation (référez-vous au tableau vu plus haut ou avec l'aide de hashcat) :

```
$ hashcat -m 100 -a 1 hash.txt dic1.txt dic2.txt
```



L'attaque par masque ⓘ

Beaucoup d'utilisateurs construisent leurs mots de passe en suivant une structure assez récurrente, et c'est sur cette faiblesse humaine que nous allons nous attarder avec l'attaque par masque.

De nombreux paramètres sont disponibles pour ce type d'attaque, permettant de réduire le nombre de tentatives avant la découverte dudit mot de passe. Voici les principaux :

Argument	Signification
`-m`	Code de l'algorithme de hachage (comme vu précédemment)
`-a 3`	Type d'attaque : 3 = Attaque par masque
`-i`	Augmenter progressivement la longueur des mots de passe essayés
`-increment-min`	Longueur minimale du mot de passe
`-increment-max`	Longueur maximale du mot de passe
`-1 (avec le chiffre 1)`	Jeu de caractères personnalisé n°1 (possible d'en fournir un 2e avec -2, etc.)

Les masques de caractères disponibles sont les suivants :

Masque	Jeu de caractères correspondant
`?l`	abcdefghijklmnopqrstuvwxyz
`?u`	ABCDEFGHIJKLMNOPQRSTUVWXYZ
`?d`	0123456789
`?h`	0123456789abcdef
`?H`	0123456789ABCDEF
`?s`	␣!"#$%&'()*+,-./:;<=>?@[\]^_{|}~ (␣ représente l'espace)
`?a`	Équivalent à `?l` + `?u` + `?d` + `?s`
`?b`	L'ensemble des caractères Unicode de `0x00` à `0xff`
`azerty`	azerty (composez vos propres jeux de caractères de cette manière, sans le « ? » au début !)

Une attaque par masque se lance à l'aide d'une commande comme celle-ci :

```
$ hashcat -m 100 -a 3 -i --increment-min 1 --increment-max 10 -1 ?l?d hash.txt
```

- Algorithme de hachage — Voir le tableau plus haut
- Type d'attaque — 3 = Attaque par masque
- Voir tableau des arguments plus haut
- Jeu de caractères n°1 (et celui utilisé) : Lettres minuscules et chiffres

Il est aussi possible d'offrir à Hashcat des indices sur la construction du mot de passe, par exemple cette commande indique :

```
$ hashcat -m 100 -a 3 -1 0134?u -2 ?l?d hash.txt ?1?1?2?2?2?2?2?s
```

1. Que le code est haché en SHA-1
2. Que l'on souhaite faire une attaque par masque
3. Que le jeu de caractères n°1 est `01234` ainsi que tout l'alphabet majuscule
4. Que le jeu de caractères n°2 est tout l'alphabet minuscule et les chiffres
5. Et enfin que le code à trouver est composé :
   - du jeu 1 sur les deux premiers caractères
   - du jeu 2 pour les 5 suivants
   - du jeu « ?s » (caractères spéciaux) pour le dernier

Remarques :

- Si l'on offre un masque de composition (dernier argument ci-dessus) à Hashcat, alors `--increment-max` est inutile : il suffit de construire un masque de la longueur maximale à fournir
- Vous pouvez inclure des caractères fixes (ceux que vous connaîtrez) dans le masque de composition : par exemple, « `?1?1?2hello?2?2?2?2?s` » va forcer le mot de passe à travers ***hello*****.
- L'incrémentation d'un masque de composition se fera par lecture linéaire de gauche à droite (d'après l'exemple ci-dessus, tout d'abord « `?1` », puis « `?1?1` », puis « `?1?1?2` », etc.)
- Dans certains cas, les « ? » utilisés dans les commandes sont interprétés par le shell lui-même. Pensez donc à les isoler, soit entre guillemets (`"?l?l?l?l?l"`), soit en les échappant (`\?l\?l\?l\?l`)

L'attaque hybride (dictionnaire + masque)

Une autre faiblesse des mots de passe humains est le fait qu'ils sont souvent composés de mots d'un côté, et de nombres de l'autre. L'attaque hybride cherche tout simplement à tirer parti de deux des attaques citées précédemment.

L'attaque hybride est composée ainsi :

```
$ hashcat -m 100 -a 6 hash.txt dict.txt ?d?d?d?d
```

- Algorithme de hachage — Voir le tableau plus haut
- Type d'attaque — 6 = Attaque hybride dict + masque; 7 = Attaque hybride masque + dict
- Dictionnaire
- Masque de composition

Dans ce cas, les combinaisons testées iront de `Motdebut0000` À `Motfin9999`.

Dans l'autre sens, pour tester des combinaisons *masque + dictionnaire*, utilisez la syntaxe suivante :

```
$ hashcat -m 100 -a 7 hash.txt ?d?d?d?d dict.txt
```

Dans ce cas, les combinaisons testées iront de `0000Motdebut` À `9999Motfin`.

L'attaque par incrémentation (force vraiment brute)

Si aucune des méthodes mentionnées ci-dessus n'a réussi à fonctionner, il y a alors l'ultime recours, la force vraiment brute, l'attaque par incrémentation.

Pour ce type d'attaque, armez-vous de beaucoup de patience et éventuellement de processeurs et cartes graphiques performantes, car ici, il n'est plus question d'essayer les combinaisons les plus probables, mais de les essayer TOUTES !

Ce type d'attaque est tout simplement une variante de l'attaque par masque :

```
$ hashcat -m 100 -a 3 hash.txt ?a?a?a?a?a?a?a?a
```

Masque composé de « ?a », indiquant « n'importe quel caractère ».
Leur quantité définira la longueur max du mot de passe à forcer

Comment lire le résultat de Hashcat ?

Voici une liste des informations les plus utiles à repérer :

```
9526ac6017e3fa93e0eda744a982bb56:Helloo!

Session........: hashcat
Status.........: Cracked
Hash.Type......: MD5
Hash.Target....: hash.txt
Time.Started...: Sun Jan 1 12:15:33 2023 (87 secs)
Time.Estimated.: Sun Jan 1 12:17:00 2023 (0 secs)
Guess.Base.....: File (dict.txt)
Guess.Mask.....: ?u?l?l?l?l?l?s
Guess.Queue....: 6/10 (62.83%)
Speed.#1.......: 4 MH/s (2.95ms) @ Accel:4 Loops:2 Thr:8 Vec:8
Recovered......: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.......: 716358653913/16783586539239 (27.68%)
Rejected.......: 0/16783586539239 (0.00%)
Restore.Point..: 423135865393/16783586539239 (12.54%)
Restore.Sub.#1.: Salt:0 Amplifier:19200-19264 Iteration:0-64
Candidates.#1..: Tapjcqs& -> Xfgjxfs_
```

1re ligne : Résultat trouvé (au format `hash_initial:résultat`)

Status : `Exhausted` => En cours | `Cracked` => Succès

Hash target : fichier source des hashs

Guess base : fichier de dictionnaire (si disponible)

Guess mask : masque de composition (si disponible)

Speed : vitesse de forçage en Hashs par seconde (H/s)

Recovered : nombre de mots de passe trouvés / total demandé

Rejected : Nombre de candidats rejetés à cause de limitations matérielles ou logicielles.

Progress : Nombre de candidats essayés / total des possibilités

Restore point : progression du dernier point de restauration (utile pour reprendre en cas de coupure)

Candidates : fourchette de candidats pour la vue actuelle (par rapport à la dernière actualisation)

Plus d'informations disponibles sur le site officiel du projet.