

Projet Crypto :
Sujet 2

Sommaire :

Table des matières

Introduction :	3
L'histoire du chiffrement :	4
Stockage de mots de passe :	4
Fonctions de Hachage cryptographique définitions et limites :	4
Présentation aux attaques par dictionnaire :	7
Un type bien particulier : les Rainbow tables :	7
Les dictionnaires de hash :	7
Démonstration d'attaques par dictionnaire :	8
Solutions :	15
Conclusion :	17
Sources :	18

Introduction :

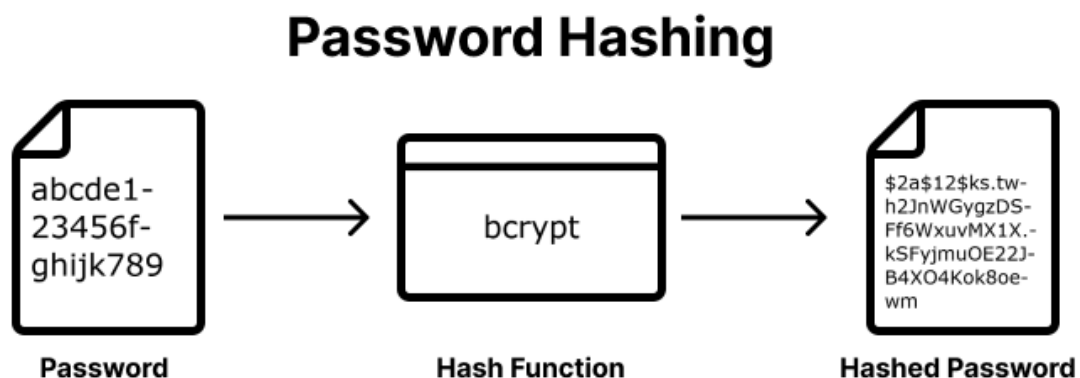
De nos jours les fuites de base de données sont courantes au sein de grandes entreprises ainsi il a fallu revoir la vision des entreprises sur le stockage des mots de passes qui sont une pratique courante dans les systèmes informatiques pour protéger des informations confidentielles. En effet les mots de passes sont les principaux mécanismes d'authentification et aussi de protection face à des individus mal intentionnés. Dans le contexte actuel de cyberguerre et de menace constante, les entreprises ont dû revoir leurs méthodes de stockage de mots de passes et leurs mécanismes de défense face aux attaques des pirates. Alors à travers ce projet nous allons voir les différents moyens pour stocker ses mots de passe, les failles de certaines méthodes, des exemples d'attaques. Alors on peut se demander comment bien stocker nos mots de passes et les protéger des attaques potentielles ?

L'histoire du chiffrement :

Le premier système de chiffrement a été inventé par Blaise de Vigenère en 1586 et ce chiffrement utilise une méthode de substitution poly alphabétique. C'est une méthode qui est restée incassable pendant 2 siècles. À l'aide d'une cryptanalyse Charles Babbage et Kasiski réussirent à trouver la clé de chiffrement en ayant juste le texte chiffré et d'une analyse statistique des lettres. Le but est de trouver des répétitions dans le texte chiffré et de deviner la clé. Depuis, de nombreuses méthodes de chiffrement ont vu le jour et nous allons particulièrement nous pencher sur les méthodes utilisées pour chiffrer des mots de passes en clair. En effet, lors de l'introduction les nombreuses fuites de données courantes ont été évoquées. De ce fait stocker des mots de passe en clair ne doit plus se faire et nous devons utiliser le chiffrement pour essayer de protéger ceux-ci même si une fuite de données a eu lieu. Ainsi de nombreuses méthodes sont utilisées pour chiffrer des mots de passe dans les bases de données et nous allons voir en détail celles-ci.

Stockage de mots de passe :

Premièrement pour le stockage de mots de passes on préfère utiliser des méthodes de hashage cryptographique. En effet si on chiffre le mot de passe il y'a forcément une clé pour le déchiffrer. La méthode de chiffrement ne possède pas une sécurité forte car si une base de données a été compromise il y'a de grandes chances que la clé de chiffrement l'a été aussi. De nos jours l'utilisation des fonctions de hashage cryptographiques est primordiale afin d'assurer une sécurité convenable.



Fonctions de Hachage cryptographique définitions et limites :

Les mots de passe sont passés dans une fonction de hash et sont cryptés par la suite. Nous allons voir ce processus en détail.

Une fonction de hachage cryptographique est une fonction qui prend en paramètre un message de taille quelconque et produit un résultat de taille fixe appelé un hash. Elle donne un résultat imprévisible car la taille de l'entrée peut être quelconque et la sortie a une taille aléatoire de n bits. De plus les valeurs que l'on obtient en sortie ne dépendent pas des valeurs en entrée ce qui rend celles-ci équiprobables.

Une fonction de hachage cryptographique est une fonction qui possède 3 caractéristiques principales. Premièrement celle-ci doit être résistante à la première préimage c'est-à-dire que si $y=h(M)$ il doit être très difficile pour une personne extérieure de trouver un message M' tel que $y=h(M')$. Elle doit être aussi résistante à la deuxième préimage c'est-à-dire pour l'attaquant il doit être

impossible de trouver une deuxième entrée qui donne la même valeur de hachage que la première valeur ($h(x)=h(x')$). Enfin celle-ci doit être résistante à la collision et est irréversible ce qui implique que l'on ne peut pas revenir en arrière c'est pour cette raison que le CRC n'est pas considéré comme une fonction de hachage cryptographique car le résultat est réversible.

On doit alors stocker la plupart de nos mots de passes sous forme de hash irréversibles à l'aide de fonctions de hachage irréversibles mais sont-elles fiables et incassables ?

Il existe encore de nombreux sites qui utilisent des fonctions de hachage cryptographiques qui se sont révélées obsolètes avec le temps tels que MD5 et SHA1. MD5 est devenu obsolète car on a découvert des collisions et le programme John The Ripper permet de casser les MD5 facile par brute force. De plus les Rainbow tables permettent de cracker en quelques secondes les hashes. Ainsi MD5 n'est plus considéré comme une fonction de hachage cryptographique car celle-ci n'est pas résistante aux collisions. Pour SHA 1 la cause est à peu près similaire car une attaque de brute force basée sur le birthday paradox permet de trouver une collision sur une clé SHA 1 avec 2^{33} opérations. Ainsi l'utilisation de SHA 1 est interdite.

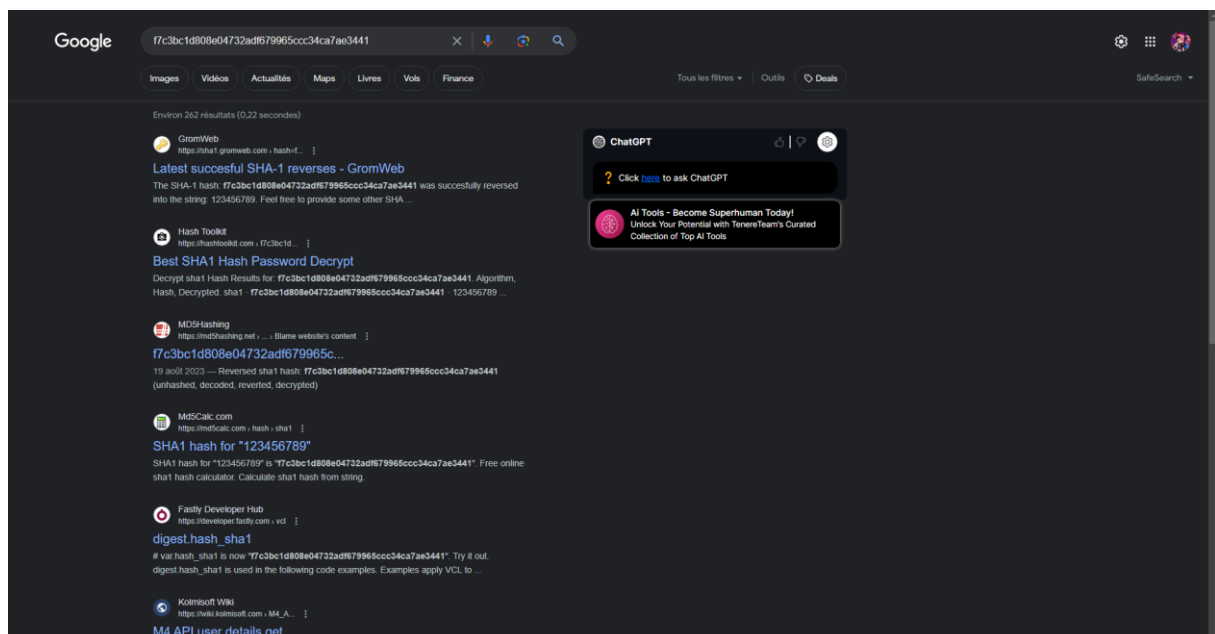
[Home Page](#) | [SHA1 in JAVA](#) | [Secure password generator](#) | [Linux](#) | [Privacy Policy](#)

SHA1 and other hash functions online generator

123456789 hash

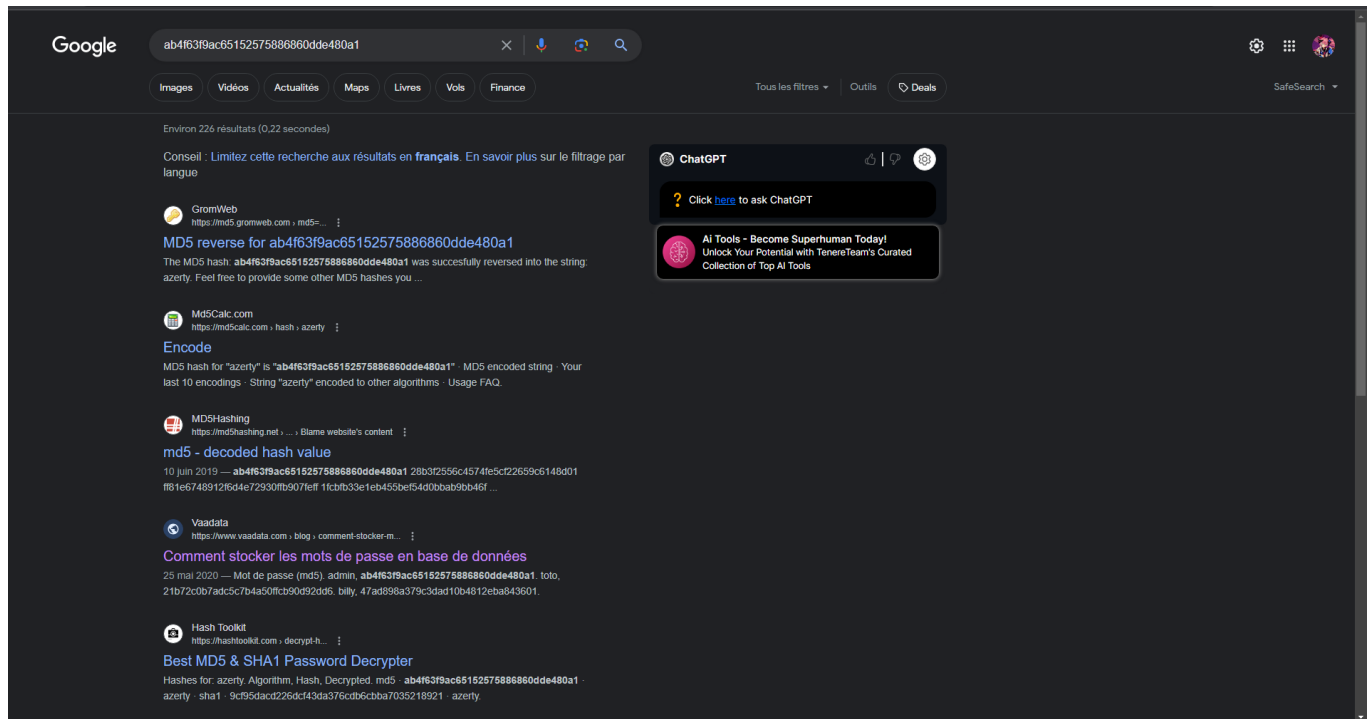
sha-1

Result for sha1: f7c3bc1d808e04732adf679965ccc34ca7ae3441



On a pu retrouver un mot de passe en 5 secondes.

Par exemple si un des mots de passe dans la base de données est « azerty », l'attaquant a juste à taper le hash donné par md5 « ab4f63f9ac65152575886860dde480a1 » pour ce mot de passe et il retrouvera azerty :



Cependant si la recherche internet n'a pas été fructueuse il peut se tourner vers d'autres méthodes : le brute force et l'attaque par dictionnaire.

Brute force :

L'attaquant essaye toutes les clés possibles jusqu'à ce qu'une traduction intelligible en texte clair soit obtenue. En moyenne, la moitié de toutes les clés possibles doivent être essayées pour réussir.

Nous allons voir les attaques par dictionnaire en détail plus tard.

Pour résumer l'utilisation de fonctions de hachage cryptographiques est plus que fortement recommandé car les anciennes sont obsolètes dangereuses et interdites.

Présentation aux attaques par dictionnaire :

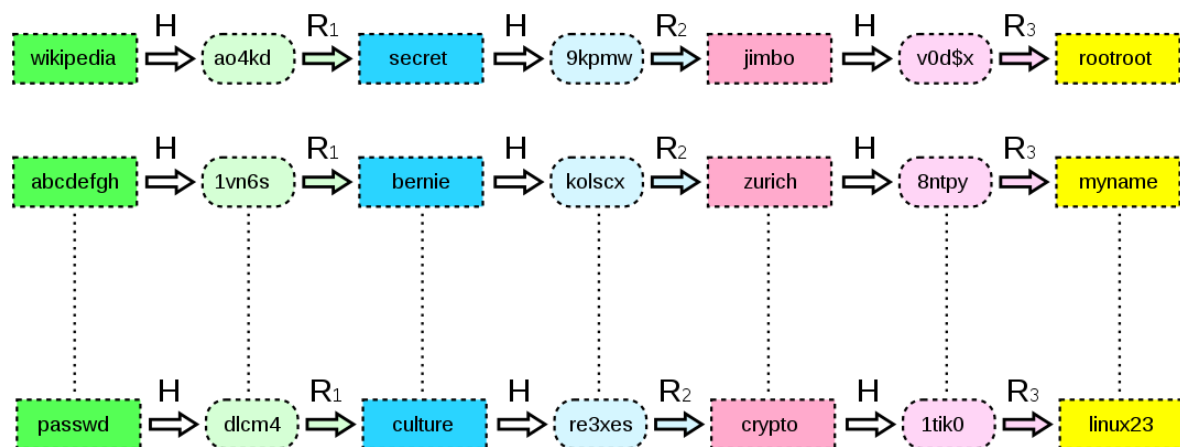
Une attaque par dictionnaire est une méthode de cryptanalyse qui trouve un mot de passe ou une clé grâce à de nombreux tests réalisés sur une série de mots de passe potentiels. A l'aide d'un dictionnaire on peut trouver des mots de passes très simple en très peu de temps car ceux-ci sont stockés dans celui-ci. Un dictionnaire stocke généralement que des mots de passes simple tels que des prénoms, des couleurs ou des animaux. Cette attaque repose alors sur l'utilisation de mots de passes simples et courts. Cette attaque est une variante de la Bruteforce mais à la place elle utilise une liste de mots de passes courants très utilisés et facile à trouver.

Il existe plusieurs types tels que les listes en fichier txt (rockyou), les Rainbow tables et les dictionnaires de hash.

Un type bien particulier : les Rainbow tables :

Une Rainbow table en cryptanalyse est une structure de données créée en 2003 par Philippe Oeschlin et sert à trouver un mot de passe à partir de son hash. Les Rainbow tables sont des outils qui permettent à des criminels à gagner de l'argent comme à des experts qui les utilisent pour vérifier si les mots de passes sont bien sécurisés. Elles permettent même sous certaines conditions de trouver des mots de passes en quelques secondes (des mots de passes simples).

Alors les Rainbow tables permettent de casser le hachage de mot de passe en cryptanalyse. Elle contient de nombreuses paires de mots de passe qui sont obtenues en partant d'un mot de passe et en calculant son hash. Ensuite une fonction de réduction permet ensuite la création un nouveau mot de passe à partir de ce hash. On continue ce processus et après un nombre fixe d'opérations on obtient le deuxième mot de passe de la paire.



Elles ont été créées pour améliorer les compromis temps-mémoire. En effet cette méthode d'attaque est très rapide car hacher un mot de passe et le réduire sont des actions rapides et efficaces ce qui en fait une méthode redoutable. Alors on prend des mots de passes communs, puis on les hash et on les réduit pour trouver un nouveau mot jusqu'au résultat attendu ici dans notre exemple nous allons hacher jusqu'à obtenir rootroot qui est le mot de passe qui sera stocké à la fin. Donc on va transformer le hash en un autre mot de passe grâce à la réduction.

Les dictionnaires de hash :

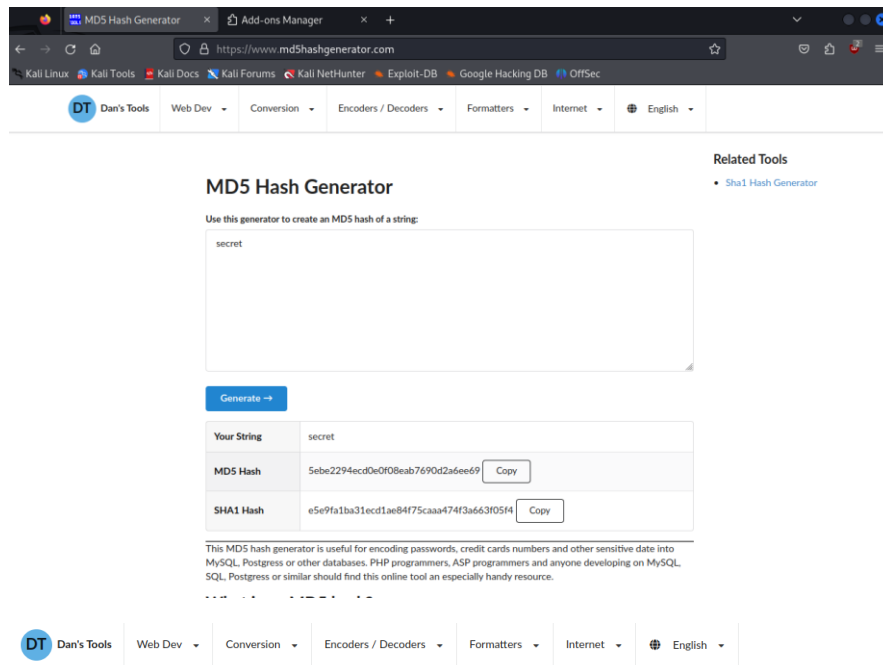
Ce dictionnaire contient des mots de passes extrêmement utilisés et courants tels que azerty etc.... Et les hashes correspondants aux mots de passes en clairs. Les calculs de hash ne sont pas nécessaires car ils sont déjà calculés ce qui fait gagner du temps mais elles prennent beaucoup de place dans le

stockage. Ils sont certes plus rapide que les rainbow tables les dictionnaires de hashes coûtent beaucoup d'espaces.

Démonstration d'attaques par dictionnaire :

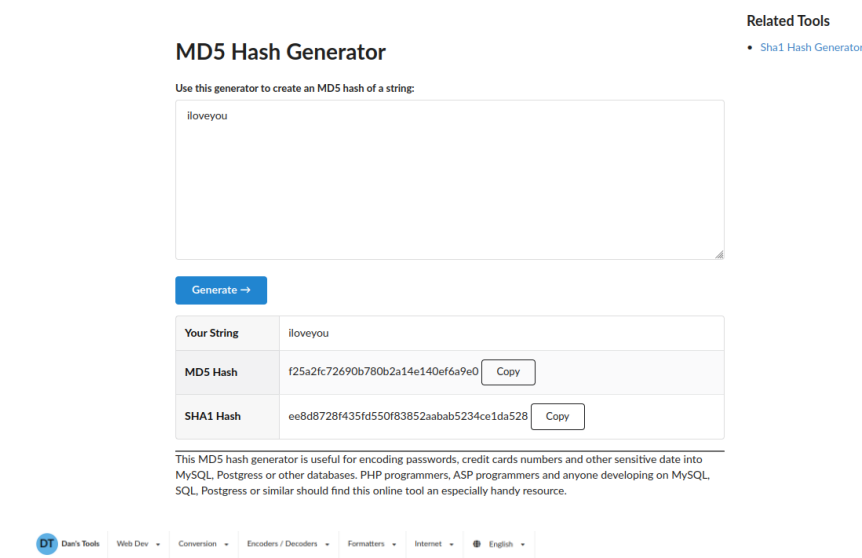
Liste RockYou :

On génère un hash md5 puis on le stocke dans un fichier txt



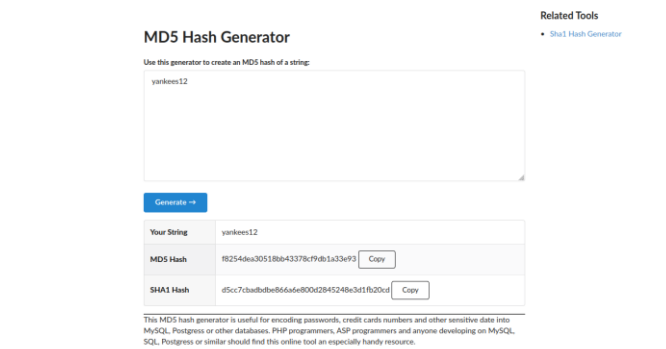
The screenshot shows the MD5 Hash Generator website. The input field contains the string "secret". The "Generate" button is clicked. The results table shows the MD5 Hash as "5ebe2294ecd0e08eab7690d2a6ee69" and the SHA1 Hash as "e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4".

Your String	secret
MD5 Hash	5ebe2294ecd0e08eab7690d2a6ee69 Copy
SHA1 Hash	e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4 Copy



The screenshot shows the MD5 Hash Generator website. The input field contains the string "iloveyou". The "Generate" button is clicked. The results table shows the MD5 Hash as "f25a2fc72690b780b2a14e140ef6a9e0" and the SHA1 Hash as "ee8d8726f435fd550f83852aabab5234ce1da528".

Your String	iloveyou
MD5 Hash	f25a2fc72690b780b2a14e140ef6a9e0 Copy
SHA1 Hash	ee8d8726f435fd550f83852aabab5234ce1da528 Copy



The screenshot shows the MD5 Hash Generator website. The input field contains the string "yankees12". The "Generate" button is clicked. The results table shows the MD5 Hash as "f82546a30518b643378f99b5a33a93" and the SHA1 Hash as "d5c7baad8be86a6e80042945248e3d1fb20c".

Your String	yankees12
MD5 Hash	f82546a30518b643378f99b5a33a93 Copy
SHA1 Hash	d5c7baad8be86a6e80042945248e3d1fb20c Copy


```
(kali@kali)~[/usr/share/wordlists]
$ sudo hashcat -m 0 -a 0 hashes.txt /usr/share/wordlists/rockyou.txt -o mdp.txt
[sudo] password for kali:
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian Linux, None+Asserts, RELOC, SPIR, LLVM 15.0.6, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

* Device #1: pthread-penryn-13th Gen Intel(R) Core(TM) i5-13600KF, 4949/9963 MB (2048 MB allocatable), 9MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashfile 'hashes.txt' on line 6 (bb04b7a0ef48ac1e6e2eecd0d4b984f5): Token length exception

* Token length exception: 1/6 hashes
  Passwords or other datasets PHP programmers ASP programmers and anyone developing on
  This error happens if the wrong hash type is specified, if the hashes are
  malformed, or if input is otherwise not as expected (for example, if the
  --username option is used but no username is present)

Hashes: 5 digests; 5 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Initializing backend runtime for device #1. Please be patient ... ^X@s^X@s^

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 2 MB

Dictionary cache built:
* Filename: /usr/share/wordlists/rockyou.txt
* Passwords: 14344392
* Bytes: 139921548
* Keyspace: 14344385
* Runtime: 0 secs

Cracking performance lower than expected?

* Append -O to the commandline.
  This lowers the maximum supported password/salt length (usually down to 32).

* Append -w 3 to the commandline.
  This can cause your screen to lag.

* Append -S to the commandline.
  This has a drastic speed impact but can be better for specific attacks.
  Typical scenarios are a small wordlist but a large ruleset.

* Update your backend API runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 0 (MD5)
Hash.Target.....: hashes.txt
Time-Started.....: Mon Oct 2 15:06:40 2023 (5 secs)
Time-Elapsed...: Mon Oct 2 15:06:45 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 3182.0 MH/s (0.45ms) @ Accel:1024 Loops:1 Thr:1 Vec:4
Recovered.....: 3/5 (60.00%) Digests (total), 3/5 (60.00%) Digests (new)
Progress.....: 14344385/14344385 (100.00%)
Rejected.....: 0/14344385 (0.00%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:8-1 Iteration:0-1
Candidates.#1...: $HEX[216361726f6c796e] -> $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1...: Util: 22%

Started: Mon Oct 2 15:06:25 2023
Stopped: Mon Oct 2 15:06:40 2023

(kali@kali)~[/usr/share/wordlists]
$ ls
mass_dib dirbuster fasttrack.txt fern-wifi hashes.txt john.lst legion mdp.txt metasploit mmap.lst rockyou.txt rockyou.txt.gz sqlmap.txt wfuzz wifite.txt
(kali@kali)~[/usr/share/wordlists]
```

On a cracker les 3 mots de passe en 5 secondes et on l'a réalisé sur une machine virtuelle ce qui est mal optimisé pour casser des passwords.

En 15 secondes il a calculé tous les hashes dans la liste rockyou.txt-

```
*usr/share/wordlists/hashe.txt - Mousepad
File Edit Search View Document Help
Warning: you are using the root account. You may harm your system.
1 5ebe2294ecd0e0f08eab7690d2a6ee69
2 f8254dea30518bb43378cf9db1a33e93
3 f25a2fc72690b780b2a14e140ef6a9e0
4 $
```

```
/home/kali/Downloads/mdp.txt - Mousepad
File Edit Search View Document Help
Warning: you are using the root account. You may harm your system.
1 f25a2fc72690b780b2a14e140ef6a9e0:iloveyou
2 5ebe2294ecd0e0f08eab7690d2a6ee69:secret
3 f8254dea30518bb43378cf9db1a33e93:yankees12
4
```

Maintenant on essaye le hashcat et la liste rockyou sur du sha256 :

```
(kali@kali)-[/usr/share/rainbowcrack]
--$ sudo hashcat -m 1400 sha256.txt /usr/share/wordlists/rockyou.txt -o mdp.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian Linux, None+Asserts, RELOC, SPIR, LLVM 15.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

* Device #1: pthread-penryn-13th Gen Intel(R) Core(TM) i5-13600KF, 4949/9963 MB (2048 MB allocatable), 9MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 2 MB

Starting self-test. Please be patient... █
```

```
/usr/share/rainbowcrack/mdp.txt - Mousepad
File Edit Search View Document Help
[Icons]
Warning: you are using the root account. You may harm your system.
1 e4ad93ca07acb8d908a3aa41e920ea4f4ef4f26e7f86cf8291c5db289780a5ae:iloveyou
2 |
```

```
kali@kali: /usr/share/rainbowcrack
File Actions Edit View Help
See the above message to find out about the exact limits.
Watchdog: Temperature abort trigger set to 90c
Host memory required for this attack: 2 MB
Dictionary cache hit:
* Filename...: /usr/share/wordlists/rockyou.txt
* Passwords..: 14344385
* Bytes.....: 139921507
* Keyspace...: 14344385

Session.....: hashcat
Status.....: Cracked
Hash.Mode.....: 1400 (SHA2-256)
Hash.Target....: e4ad93ca07acb8d908a3aa41e920ea4f4ef4f26e7f86cf8291c ... 80a5ae
Time.Started...: Mon Oct 2 16:29:54 2023 (0 secs)
Time.Estimated...: Mon Oct 2 16:29:54 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 3374.8 kH/s (0.39ms) @ Accel:1024 Loops:1 Thr:1 Vec:4
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 9216/14344385 (0.06%)
Rejected.....: 0/9216 (0.00%)
Restore.Point...: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> sassy123
Hardware.Mon.#1..: Util: 3%

Started: Mon Oct 2 16:29:44 2023
Stopped: Mon Oct 2 16:29:55 2023
```

On a cracké le mot de passe en 11 sec avec hashcat

Rainbow Tables :

On installe premièrement le package rainbow crack qui va nous générer une rainbow table :

```
(kali㉿kali)-[~]
└─$ sudo apt-get install rainbowcrack
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  rainbowcrack
0 upgraded, 1 newly installed, 0 to remove and 689 not upgraded.
Need to get 130 kB of archives.
After this operation, 506 kB of additional disk space will be used.
Get:1 http://archive-4.kali.org/kali kali-rolling/main amd64 rainbowcrack amd64 1.8-0kali1 [130 kB]
Fetched 130 kB in 1s (226 kB/s)
Selecting previously unselected package rainbowcrack.
(Reading database ... 398533 files and directories currently installed.)
Preparing to unpack .../rainbowcrack_1.8-0kali1_amd64.deb ...
Unpacking rainbowcrack (1.8-0kali1) ...
Setting up rainbowcrack (1.8-0kali1) ...
Processing triggers for kali-menu (2023.4.3) ...
```

```
(kali㉿kali)-[/usr/share/rainbowcrack]
└─$ sudo rtgen sha256 loweralpha-numeric 1 7 0 19000 19000 0
rainbow table sha256_loweralpha-numeric#1-7_0_19000x19000_0.rt parameters
hash algorithm:      sha256
hash length:         32
charset name:        loweralpha-numeric
charset data:        abcdefghijklmnopqrstuvwxyz0123456789
charset data in hex: 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 30 31 32 33 34 35 36 37 38 39
charset length:      36
plaintext length range: 1 - 7
reduce offset:       0x00000000
plaintext total:      80603140212

sequential starting point begin from 0 (0x0000000000000000)
generating ...

^[[C19000 of 19000 rainbow chains generated (0 m 8.0 s)
```

```
(kali㉿kali)-[/usr/share/rainbowcrack]
└─$ sudo rtgen md5 loweralpha-numeric 1 4 0 3000 3000 0
rainbow table md5_loweralpha-numeric#1-4_0_3000x3000_0.rt parameters
hash algorithm:      md5
hash length:         16
charset name:        loweralpha-numeric
charset data:        abcdefghijklmnopqrstuvwxyz0123456789
charset data in hex: 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a 30 31 32 33 34 35 36 37 38 39
charset length:      36
plaintext length range: 1 - 4
reduce offset:       0x00000000
plaintext total:      1727604

sequential starting point begin from 0 (0x0000000000000000)
generating ...
3000 of 3000 rainbow chains generated (0 m 0.1 s)
```

On génère une rainbow table avec des hash de format sha256 et MD5 pour un plaintext de 1 à 4 et de 1 à 7 caractères avec des caractères en minuscules de a à z, les tables auront comme taille 1000 et va créer 3000 chaînes (19000 et 19000) et on stockera tout dans un fichier .rt.

On trie les tables pour accélérer les accès et les recherches dans les tables.

```
(kali㉿kali)-[/usr/share/rainbowcrack]
$ sudo rtsort $(pwd)
[sudo] password for kali:
/usr/share/rainbowcrack/sha256_loweralpha-numeric#1-10_0_1000x4000_0.rt:
10339647488 bytes memory available
loading data ...
sorting data ...
writing sorted data ...
/usr/share/rainbowcrack/md5_loweralpha-numeric#1-10_0_1000x4000_0.rt:
10339647488 bytes memory available
loading data ...
sorting data ...
writing sorted data ...
```

On va voir le temps que met rainbowcrack à cracker un hash md5 puis on fera le test sur sha256.

Source:

rain|

Result:

23678db5efde9ab46bce8c23a6d91b50

Copy

Submit

```
(root㉿kali)-[/usr/share/rainbowcrack]
# sudo rcrack . -h 23678db5efde9ab46bce8c23a6d91b50
1 rainbow tables found
memory available: 8070964838 bytes
memory for rainbow chain traverse: 48000 bytes per hash, 48000 bytes for 1
memory for rainbow table buffer: 2 x 48016 bytes
disk: ./md5_loweralpha-numeric#1-4_0_3000x3000_0.rt: 48000 bytes read
disk: finished reading all files
plaintext of 23678db5efde9ab46bce8c23a6d91b50 is rain

statistics
-----
plaintext found:                1 of 1
total time:                    0.11 s
time of chain traverse:         0.11 s
time of alarm check:           0.00 s
time of disk read:             0.00 s
hash & reduce calculation of chain traverse: 4497000
hash & reduce calculation of alarm check: 11146
number of alarm:                292
performance of chain traverse:  42.03 million/s
performance of alarm check:     2.79 million/s

result
-----
23678db5efde9ab46bce8c23a6d91b50  rain  hex:7261696e
```

Maintenant on va tester des hash de sha 256 avec rainbowcrack :

This SHA256 online tool helps you calculate hash from string or binary.

a

Input type Text ☐ Remember Input

Hash ☒ Auto Update

ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb

Copy

https://emn178.github.io/online-tools/sha256.html?input=a&input_type=text

Copy

```
(root@kali)-[/usr/share/rainbowcrack]
# sudo rcrack -h ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb
1 rainbow tables found
memory available: 8091202355 bytes
memory for rainbow chain traverse: 304000 bytes per hash, 304000 bytes for 1 hashes
memory for rainbow table buffer: 2 x 304016 bytes
disk: ./sha256_loweralpha-numeric#1-7_0_19000x19000_0.rt: 304000 bytes read
disk: finished reading all files
plaintext of ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb is a

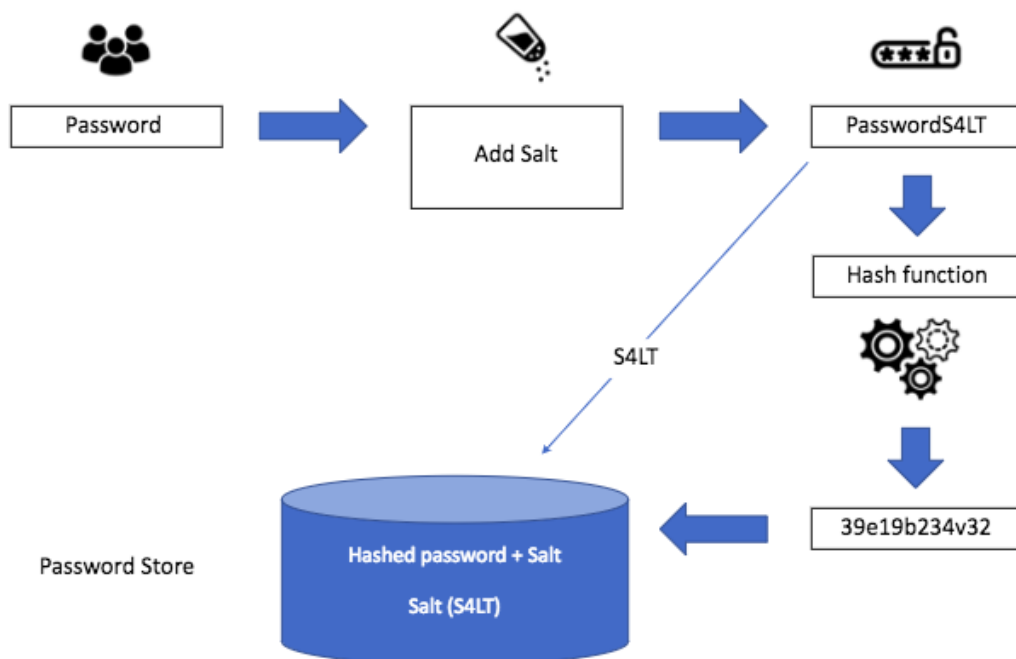
statistics
-----
plaintext found:          1 of 1
total time:              4.98 s
time of chain traverse:   4.96 s
time of alarm check:      0.01 s
time of disk read:        0.00 s
hash & reduce calculation of chain traverse: 180481000
hash & reduce calculation of alarm check:    288205
number of alarm:          43
performance of chain traverse: 36.34 million/s
performance of alarm check:  26.20 million/s

result
-----
ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb a hex:61
```

Pour rain en md5 on a mis 0.11 secondes et pour a en sha256 on a mis 5s. Ce qui montre que sha256 est plus robuste que md5

Solutions :

Donc après toutes ces démonstrations on se rend compte que des fonctions comme sha256 et les autres sha ne sont pas très efficaces pour stocker des mots de passes. En effet celles-ci servent à contrôler l'intégrité des fichiers, créer des signatures électroniques et l'optimisation de la recherche. En effet on a vu que hashcat a trouvé en 11 seconde le mot de passe chiffré en sha256. Le problème des fonctions de hash sont déterministes (Possibilité de retrouver le mot de passe en utilisant des attaques par dictionnaire tels que les Rainbow tables ou hashcat avec la liste rockyou). Cependant on peut optimiser ces fonctions pour qu'elles soient plus résistantes. Pour cela nous allons utiliser un sel sur ce hash. Le sel permet d'ajouter des données supplémentaires à un hash pour empêcher que l'on tombe sur des hash identiques pour un mot donc d'avoir des hashes différents pour un même mot. Ainsi nous avons pu grâce au sel rendre ces fonctions non déterministes.



An Example of Salting a Password

Password (Original Input):	h@ckPr00f
Salt (Unique Integer):	H13qY7YPCDJULY7I
Password + Salt Combo:	h@ckPr00fH13qY7YPCDJULY7I

Finalement on peut ajouter un poivre aux hashes à qui on a déjà appliqué un sel. Le poivre est une valeur secrète ajoutée à une entrée telle qu'un mot de passe lors du hachage avec une fonction de hachage cryptographique. Cette valeur diffère d'un sel en ce qu'elle n'est pas stockée avec un hachage de mot de passe, mais plutôt le poivre est conservé séparément dans un autre support, tel qu'un module de sécurité matérielle. Un poivre est similaire en concept à un sel ou une clé de chiffrement. C'est comme un sel dans la mesure où c'est une valeur aléatoire qui est ajoutée à un hachage de mot de passe, et c'est similaire à une clé de chiffrement dans la mesure où elle doit être gardée secrète.

Un poivre remplit un rôle comparable à celui d'un sel ou d'une clé de chiffrement, mais alors qu'un sel n'est pas secret (simplement unique) et peut être stocké avec la sortie hachée, un poivre est secret et ne doit pas être stocké avec la sortie. Le hachage et le sel sont généralement stockés dans une base de données, mais un poivre doit être stocké séparément pour éviter qu'il ne soit obtenu par l'attaquant en cas de violation de la base de données 2. Alors que le sel doit seulement être suffisamment long pour être unique par utilisateur [douteux - discuter], un poivre doit être suffisamment long pour rester secret contre les tentatives de force brute pour le découvrir (le NIST recommande au moins 112 bits).

Enfin l'utilisation d'un mot de passe robuste et unique est obligatoire pour se protéger face à ce type d'attaques qui marchent efficacement sur des mots de passes simples.

Conclusion :

Pour conclure nous avons pu à travers ce projet comprendre les mécanismes d'une fonction de hachage cryptographiques, ses limites, les failles de certaines et comprendre celles-ci, des types d'attaques et leurs applications pratiques et les différentes solutions pour pallier ce problème et stocker ses mots de passes de manière sécurisés.

Sources :

<https://www.ssi.gouv.fr/uploads/2021/10/anssi-guide-authentification-multifacteur-et-mots-de-passe.pdf>

<https://www.ssi.gouv.fr/uploads/2016/05/fiches-authentification-1.6.pdf>

<https://www.ssi.gouv.fr/uploads/2021/09/anssi-ii-901-inter-ministerial-directive-pertaining-to-the-protection-of-sensitive-information-systems.pdf>

<https://www.nextinpact.com/article/48417/phrases-passe-anssi-passe-en-mode-2-0>

<https://www.francenum.gouv.fr/guides-et-conseils/protection-contre-les-risques/cybersecurite/pourquoi-et-comment-utiliser-un>

<https://www.ionos.com/digitalguide/server/security/rainbow-tables/>