



Forensic : Password Crack

Rapport de projet MSI P9 2020



Dhia LAYADI
Lotfi DERRI
14/12/2020

Veracrypt

Description

VeraCrypt est un logiciel utilitaire sous licence libre utilisé pour le chiffrement à la volée (OTFE). Il est développé par la société française IDRIX et permet de créer un disque virtuel chiffré dans un fichier ou une partition. L'ensemble du dispositif de stockage demande une authentification avant de monter le disque virtuel.

En France, le logiciel est intégré à la liste des logiciels libres préconisés par l'État dans le cadre de la modernisation globale de ses systèmes d'informations (S.I.).

VeraCrypt est un fork qui a été initialement publié le 22 juin 2013 pour faire suite au projet TrueCrypt interrompu subitement quelques mois plus tard. Selon ses développeurs, des suspicions sur l'origine de plusieurs failles de sécurité ont été soulevées à la suite d'un audit du code source de TrueCrypt. Elles ont donné lieu à des améliorations de sécurité mises en œuvre dès la version 1.17 de VeraCrypt. La version 1.24 (update 2) a été publiée le 16 décembre 2019.

Mise en place

Installation

Il y a plusieurs méthodes d'installation :

- La première consiste à télécharger Veracrypt sur le site officiel et à le décompresser dans un dossier. Certaines dépendances, selon les versions Veracrypt, sont requises. L'installation est parfois plus complexe pour une machine tournant sous une architecture ARM. Auquel cas, l'installation par PPA peut être plus facile.
- La seconde consiste à ajouter un PPA (Personal Package Archives). Cette solution est efficace lorsqu'il y a un problème de compatibilité (notamment avec les architectures ARM), et facilite la mise à jour.

Avant l'installation d'un logiciel tel que veracrypt on doit d'abord vérifier que les fichier téléchargés sont les bons. Pour ce faire, on peut vérifier la signature du produit téléchargé .

Pour commencer, on doit créer une clé public via **gpg**

```
> gpg --gen-key
gpg (GnuPG) 2.2.20; Copyright (C) 2020 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Remarque : Utilisez « gpg --full-generate-key » pour une fenêtre de dialogue de génération de clef complète.
GnuPG doit construire une identité pour identifier la clef.

Nom réel : Layadi
Adresse électronique : dhialayadi@gmail.com
Vous avez sélectionné cette identité :
« Layadi <dhialayadi@gmail.com> »
```

Par la suite, nous allons sur le site officiel et on télécharge le clé publique du logiciel à vérifier

I released files are signed with a PGP key available on the following link : https://www.idrix.fr/VeraCrypt/VeraCrypt.PGP_public_key.asc. It's also available on major key servers with ID=0x680D16DE. Please check that its fingerprint is 5069A233D55A0EEB174A5FC3821ACD02680D16DE.

On importe la clé dans gpg puis on la signe avec notre clé publique pour pouvoir l'utiliser lors de vérification.

```
> gpg --import Veracrypte.key
gpg: key 821ACD02680D16DE: 1 signature not checked due to a missing key
gpg: clef 821ACD02680D16DE : clef publique « VeraCrypt Team (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr> » importée
gpg: Quantité totale traitée : 1
gpg: importées : 1
gpg: marginales needed: 3 complètes needed: 1 trust model: pgp
gpg: profondeur : 0 valables : 1 signées : 0
gpg: confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
gpg: la prochaine vérification de la base de confiance aura lieu le 2022-12-15
```

```
> gpg --sign-key 0x680D16DE

pub  rsa4096/821ACD02680D16DE
     créé : 2018-09-11 expire : jamais utilisation : SC
     confiance : inconnu validité : inconnu
sub  rsa4096/200B5A9D26878A32
     créé : 2018-09-11 expire : jamais utilisation : E
sub  rsa4096/0F5AACD65483D029
     créé : 2018-09-11 expire : jamais utilisation : A
[ inconnue ] (1). VeraCrypt Team (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr>

pub  rsa4096/821ACD02680D16DE
     créé : 2018-09-11 expire : jamais utilisation : SC
     confiance : inconnu validité : inconnu
Empreinte clef princip. : 5069 A233 D55A 0EEB 174A 5FC3 821A CD02 680D 16DE
VeraCrypt Team (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr>

Voulez-vous vraiment signer cette clef avec votre
clef « Layadi <dhialayadi@gmail.com> » (CE9E4210A635B435)
Voulez-vous vraiment signer ? (o/N) o
```

Pour finir on va télécharger notre logiciel avec sa **pgp key** .

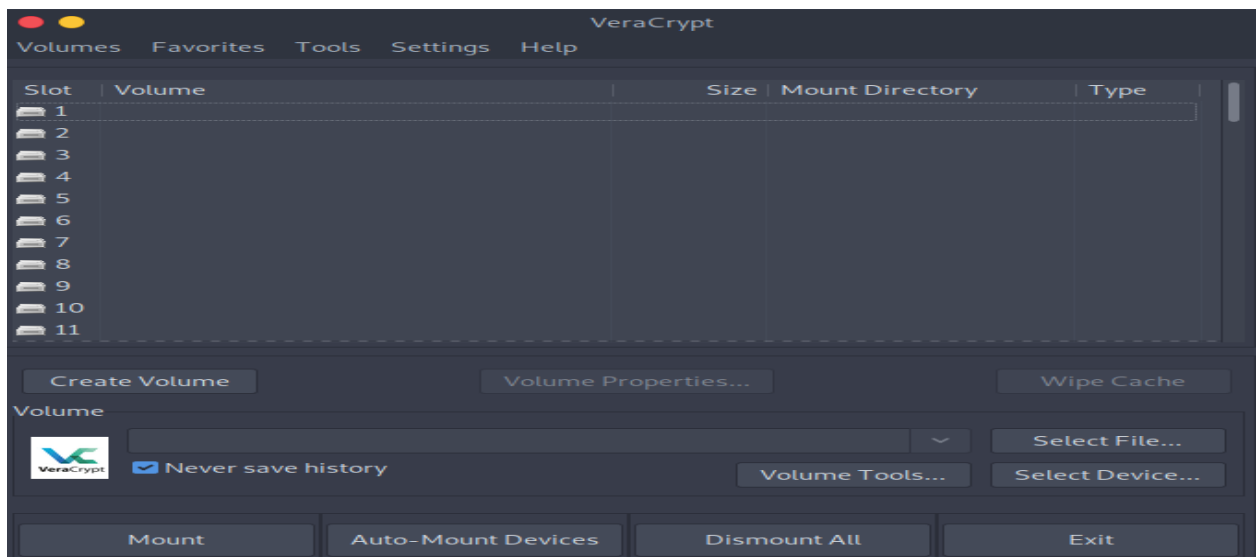
- Debian 10:
 - GUI: [veracrypt-1.24-Update7-Debian-10-amd64.deb](#) (PGP Signature)
 - Console: [veracrypt-console-1.24-Update7-Debian-10-amd64.deb](#) (PGP Signature)

```

> gpg --verify veracrypt-1.24-Update7-Debian-10-amd64.deb.sig
gpg: les données signées sont supposées être dans « veracrypt-1.24-Update7-Debian-10-amd64.deb »
gpg: Signature faite le sam. 08 août 2020 20:21:40 CEST
gpg:         avec la clef RSA 5069A233D55A0EEB174A5FC3821ACD02680D16DE
gpg: vérification de la base de confiance
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: profondeur : 0  valables : 1  signées : 1
gpg:   confiance : 0 i., 0 n.d., 0 j., 0 m., 0 t., 1 u.
gpg: profondeur : 1  valables : 1  signées : 0
gpg:   confiance : 1 i., 0 n.d., 0 j., 0 m., 0 t., 0 u.
gpg: la prochaine vérification de la base de confiance aura lieu le 2022-12-15
gpg: Bonne signature de « VeraCrypt Team (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr> » [totale]

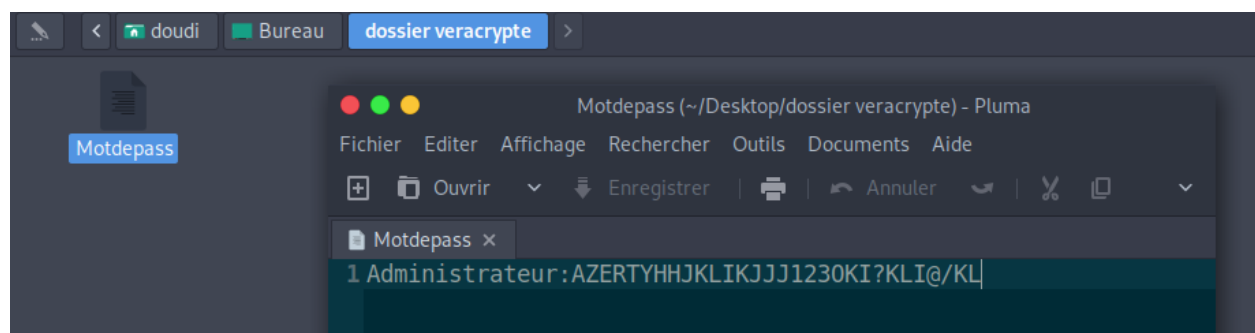
```

Nous avons confirmation que c'est le bon logiciel on peut se préparer à son installation.

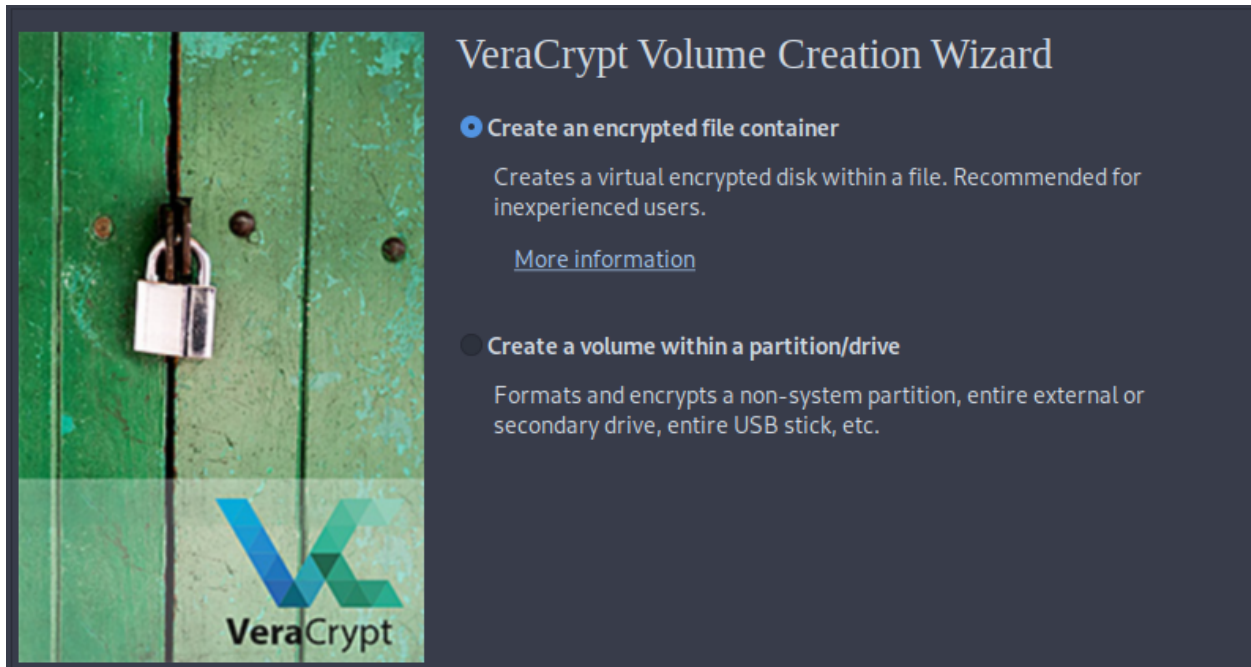


Utilisation

Nous allons utiliser veracrypt pour crypter un dossier contenant le mot de passe d'un compte administrateur :



Pour commencer, nous allons créer un espace de stockage pour contenir notre dossier à l'aide de l'interface de veracrypt.

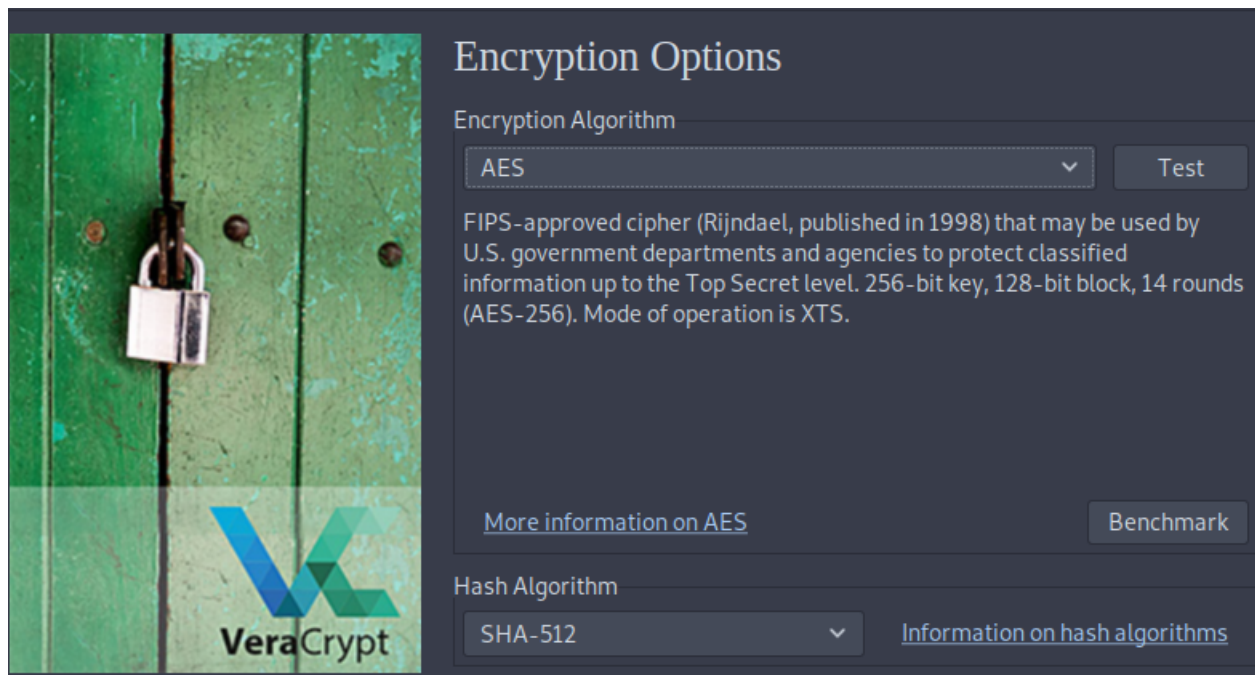


On peut utiliser deux modes : le mode *"normal"* et le mode *"hidden"* pour le nouveau volume créé.

Dans notre cas, nous avons gardé le mode normal puis nous avons choisi l'emplacement du nouveau disque.



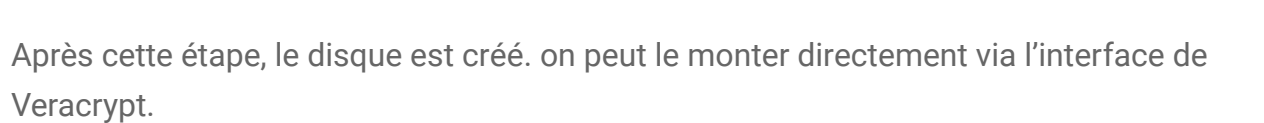
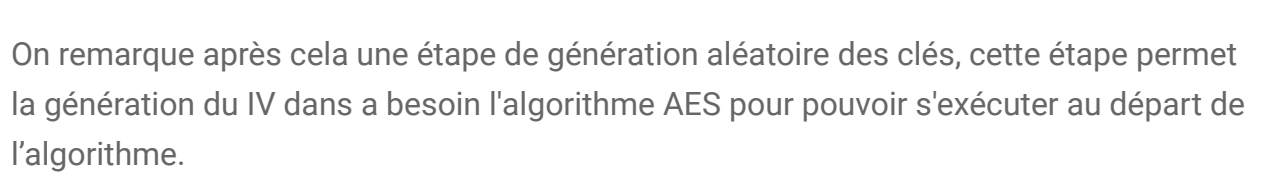
On peut aussi choisir les options de chiffrement et de hachage. Par défaut, Veracrypt utilise AES-256 pour chiffrer le disque et SHA512 pour hacher le mot de passe de chiffrement..

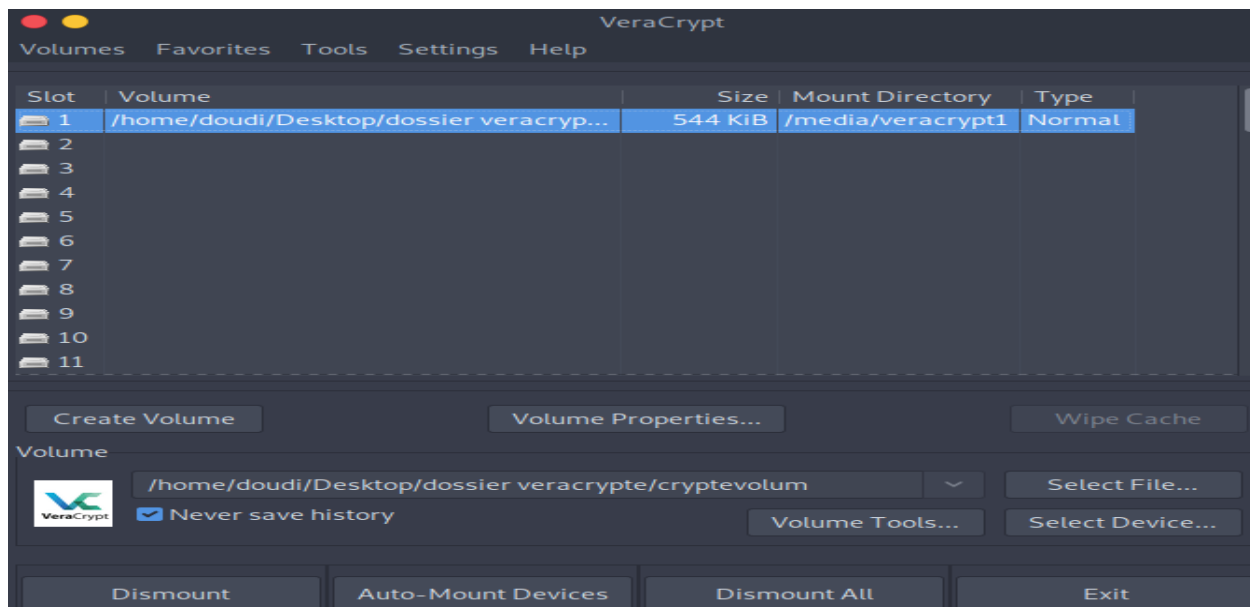


Choix du password

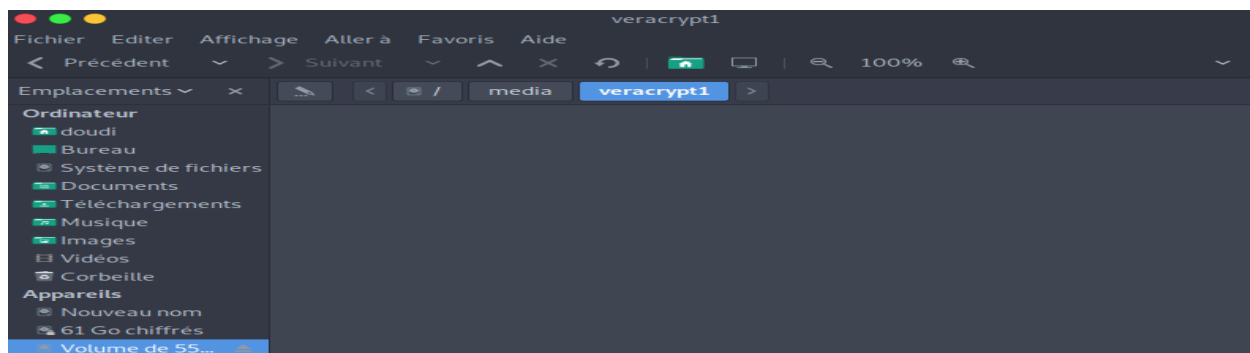
Après cette étape, nous pouvons passer au choix du mot de passe.

Cette étape est primordiale car en vu de la taille du mot de passe et les caractères contenus dans celui-ci on peut savoir si l'algorithme est vulnérable aux attaques bruteforce sur le hash. Dans notre exemple on a choisi le mot de passe "**Password**" qui est un mot de passe faible surtout selon les normes imposées par veracrypt qui suggère d'utiliser un mot de passe avec un longueurs de 20 caractères et avec des caractères spéciaux .





On remarque directement après celle-ci l'apparition de la partition dans le navigateur de disque.



On copie notre fichier dans le volume veracrypt

Extraction du hash

Pour l'extraction du hash on va utiliser l'outil DD présent dans Linux ,

L'opération d'extraction du hash est la même que celle de TrueCrypt, si VeraCrypt était utilisé sur le disque de démarrage, on se doit d'échapper les 64 premiers Kilo Octet , mais vu que dans notre cas, le disque est un disque de stockage qui n'est pas utilisé sur le disque de démarrage, nous allons directement extraire les 512 premiers bits et cela grâce à la commande suivante :


```
> dd if=cryptevolum of=cryptevolum.tc bs=1 count=512
512+0 enregistrements lus
512+0 enregistrements écrits
512 octets copiés, 0,00123799 s, 414 kB/s
```

Dans cette commande on va extraire du disque **cryptevolum** un bit en 512 tours pour les mettre dans le fichier cryptevolum.tc .

Attaque par dictionnaire

Cette attaque consiste à chercher dans des dictionnaires connus si le mot de passe utilisé est référencé, ou fait partie des mots de passe les plus utilisés.

Pour réaliser cette attaque, on va utiliser l'outil hashcat.

Cet outil permet de cracker les hashes de manière générale, il contient différents modes pour déchiffrer des hash veracrypt.

En cherchant dans ces modes, nous remarquons un mode SHA512 a 512 bit.

13711	VeraCrypt RIPEMD160 + XTS 512 bit	Full-Disk Encryption (FDE)
13712	VeraCrypt RIPEMD160 + XTS 1024 bit	Full-Disk Encryption (FDE)
13713	VeraCrypt RIPEMD160 + XTS 1536 bit	Full-Disk Encryption (FDE)
13741	VeraCrypt RIPEMD160 + XTS 512 bit + boot-mode	Full-Disk Encryption (FDE)
13742	VeraCrypt RIPEMD160 + XTS 1024 bit + boot-mode	Full-Disk Encryption (FDE)
13743	VeraCrypt RIPEMD160 + XTS 1536 bit + boot-mode	Full-Disk Encryption (FDE)
13751	VeraCrypt SHA256 + XTS 512 bit	Full-Disk Encryption (FDE)
13752	VeraCrypt SHA256 + XTS 1024 bit	Full-Disk Encryption (FDE)
13753	VeraCrypt SHA256 + XTS 1536 bit	Full-Disk Encryption (FDE)
13761	VeraCrypt SHA256 + XTS 512 bit + boot-mode	Full-Disk Encryption (FDE)
13762	VeraCrypt SHA256 + XTS 1024 bit + boot-mode	Full-Disk Encryption (FDE)
13763	VeraCrypt SHA256 + XTS 1536 bit + boot-mode	Full-Disk Encryption (FDE)
13721	VeraCrypt SHA512 + XTS 512 bit	Full-Disk Encryption (FDE)
13722	VeraCrypt SHA512 + XTS 1024 bit	Full-Disk Encryption (FDE)
13723	VeraCrypt SHA512 + XTS 1536 bit	Full-Disk Encryption (FDE)
13771	VeraCrypt Streebog-512 + XTS 512 bit	Full-Disk Encryption (FDE)
13772	VeraCrypt Streebog-512 + XTS 1024 bit	Full-Disk Encryption (FDE)
13773	VeraCrypt Streebog-512 + XTS 1536 bit	Full-Disk Encryption (FDE)
13731	VeraCrypt Whirlpool + XTS 512 bit	Full-Disk Encryption (FDE)
13732	VeraCrypt Whirlpool + XTS 1024 bit	Full-Disk Encryption (FDE)
13733	VeraCrypt Whirlpool + XTS 1536 bit	Full-Disk Encryption (FDE)

Nous lançons alors hashcat avec le dictionnaire rockyou.

```

> hashcat -a 0 -m 13721 cryptevolum.tc /home/doudi/HACKTHEBOX/postman/rockyou.txt
hashcat (v6.1.1-120-g15bf8b730) starting...

OpenCL API (OpenCL 1.2 LINUX) - Platform #1 [Intel(R) Corporation]
=====
* Device #1: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 15890/15954 MB (3988 MB allocatable), 8MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 64

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

Applicable optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Slow-Hash-SIMD-LOOP
* Uses-64-Bit

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 2 MB

Dictionary cache hit:
* Filename..: /home/doudi/HACKTHEBOX/postman/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace...: 14344385

```

Après 6 minutes, hashcat parvient à retrouver le mot de passe dans la wordlist utilisée.

```

cryptevolum.tc:Password

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: VeraCrypt SHA512 + XTS 512 bit
Hash.Target.....: cryptevolum.tc
Time.Started.....: Tue Dec 15 12:09:40 2020 (6 mins, 9 secs)
Time.Estimated...: Tue Dec 15 12:15:49 2020 (0 secs)
Guess.Base.....: File (/home/doudi/HACKTHEBOX/postman/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 22 H/s (5.43ms) @ Accel:1024 Loops:15 Thr:1 Vec:4
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 8192/14344385 (0.06%)
Rejected.....: 0/8192 (0.00%)
Restore.Point...: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:499995-499999
Candidates.#1...: 123456 -> whitetiger

Started: Tue Dec 15 12:09:38 2020
Stopped: Tue Dec 15 12:15:50 2020

```

Attaque par Mask

L'attaque par mask consiste à essayer toutes les combinaisons possibles ou suspectes pour pouvoir trouver le mot de passe.

Dans notre exemple, on a tenté de cracker un mot de passe de 8 caractères qui contient seulement des majuscules et des minuscules .

```

> sudo hashcat -a 3 -1 "?l?u" -m 13721 cryptevolum.tc "?1?1?1?1?1?1?1?1"
[sudo] Mot de passe de doudi :
hashcat (v6.1.1-120-g15bf8b730) starting...

```

Nous remarquons que pour une attaque comme celle-ci hashcat annonce 75 643 années, ce qui est un chiffre énorme contenu de la taille du mot de passe, rendant l'attaque irréalisable.

```
Session.....: hashcat
Status.....: Running
Hash.Name.....: VeraCrypt SHA512 + XTS 512 bit
Hash.Target.....: cryptevolum.tc
Time.Started.....: Tue Dec 15 12:32:27 2020 (3 mins, 31 secs)
Time.Estimated...: Next Big Bang (75643 years, 361 days) ←
Guess.Mask.....: ?1?1?1?1?1?1?1?1 [8]
Guess.Charset....: -1 ?l?u, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 22 H/s (5.60ms) @ Accel:16 Loops:1000 Thr:1 Vec
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 4608/53459728531456 (0.00%)
Rejected.....: 0/4608 (0.00%)
Restore.Point....: 0/1028071702528 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:36-37 Iteration:444000-445000
Candidates.#1....: Rarierin -> RICKERIN
```

Complexité

Attaque par dictionnaire :

En réalisant cette attaque on a vu que le temps estimé est de 5 jours pour passer toute la liste rockyou alors que cette liste ne contient que 14 344 385 mots de passe cela est dû à plusieurs éléments qui sont :

- La qualité du matérielle
- La longueur du hash à comparer
- Le nombre de mots de passe à comparer

Dans notre cas et surtout dans cette attaque, vu qu'on utilise un dictionnaire, la taille du mot de passe n'est pas un élément important.

Ce qui va rendre le processus plus long est surtout la taille du hash qu'on récupère, vu qu'on a un processeur I7 8 ème génération.

On peut voir dans notre capture que **hashcat** effectue 28 Hash/second.

Le processus de transformation d'un mot de passe en SHA512, puis de comparaison avec le hash ciblé par l'attaque, engendre un coût très important en matière de temps.

```

Session.....: hashcat
Status.....: Running
Hash.Name.....: VeraCrypt SHA512 + XTS 512 bit
Hash.Target.....: cryptevolum.tc
Time.Started.....: Tue Dec 15 14:35:53 2020 (2 secs)
Time.Estimated...: Mon Dec 21 10:09:23 2020 (5 days, 20 hours)
Guess.Base.....: File (/home/doudi/HACKTHEBOX/postman/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 28 H/s (4.20ms) @ Accel:1024 Loops:15 Thr:1 Vec:4
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 0/14344385 (0.00%)
Rejected.....: 0/0 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:3720-3735
Candidates.#1....: 123456 -> whitetiger

```

Attaque Par Mask :

Cette attaque est déjà en temps normal très coûteuse en temps car elle dépend principalement de la taille du mot de passe. Vu que dans notre cas on a choisi de prendre que les mots de passe avec 8 caractères soit majuscule soit minuscule.

Donc on a une liste de mot de passe qui est de 52^8 ce qui nous fait en soit "9.1343852e46" mot de passe, ce chiffre est très grand, ajouté à cela la taille du hash qui est de 512 bits ce qui réduit le temps de comparaison à 22 Hash/second.

```

Session.....: hashcat
Status.....: Running
Hash.Name.....: VeraCrypt SHA512 + XTS 512 bit
Hash.Target.....: cryptevolum.tc
Time.Started.....: Tue Dec 15 12:32:27 2020 (2 mins, 26 secs)
Time.Estimated...: Next Big Bang (75684 years, 275 days)
Guess.Mask.....: ?1?1?1?1?1?1?1?1 [8]
Guess.Charset....: -1 ?l?u, -2 Undefined, -3 Undefined, -4 Undefined
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 22 H/s (5.73ms) @ Accel:16 Loops:1000 Thr:1 Vec:4
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 3200/53459728531456 (0.00%)
Rejected.....: 0/3200 (0.00%)
Restore.Point....: 0/1028071702528 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:25-26 Iteration:248000-249000
Candidates.#1....: Barrierin -> BICKERIN

```

Conclusion

Après étude de VeraCrypt, on déduit que l'algorithme de Hachage et de chiffrement utilisés sont robuste. Surtout si on prend en compte les recommandations de l'organisme qui dit d'utiliser un mot de passe de 20 caractères avec des caractères spéciaux inclus.

Néanmoins, si le mot de passe utilisé est un mot de passe standard qui est connu du monde du hacking, alors l'attaque de celui-ci ne sera pas très difficile malgré la complexité du hash.

Si on utilise un mot de passe de 8 caractères seulement, l'attaque de celui-ci sera très complexe mais pas impossible avec du bon matériel, le seul moyen de rendre celui-ci presque inviolable est de suivre les recommandations et de privilégier les mots de passe longs en forme de phrase complexe.

ZIP

Description

ZIP est un format de fichier d'archive largement utilisé pour compresser un ou plusieurs fichiers, en réduisant leur taille globale et en facilitant leur transfert. Les fichiers ZIP fonctionnent à peu près de la même manière qu'un dossier standard sur un ordinateur. La différence est que le contenu d'un fichier zippé est compressé, ce qui réduit la quantité de données utilisées par l'ordinateur.

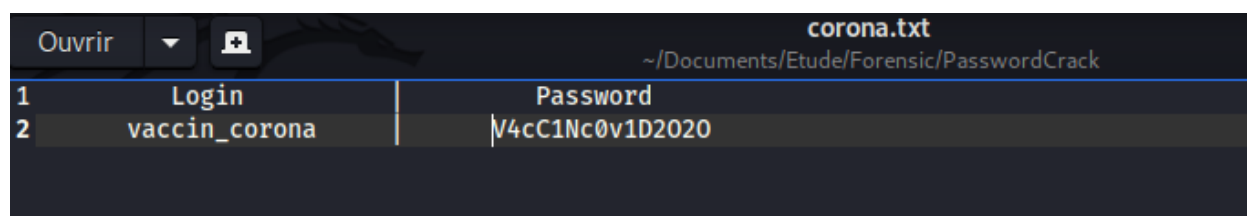
Les fichiers ZIP encodent les informations en réduisant le nombre d'octets, donc la taille du ou des fichiers, et en supprimant les données redondantes. C'est ce qu'on appelle la "compression sans perte de données", qui garantit que toutes les données originales restent intactes.

Ce format d'archivage propose un chiffrement des fichiers zippé, afin de sécuriser le transfert de ces derniers, lorsque nous voulons communiquer des données sensibles. C'est ce que nous allons étudier dans la suite de ce rapport.

Mise en place

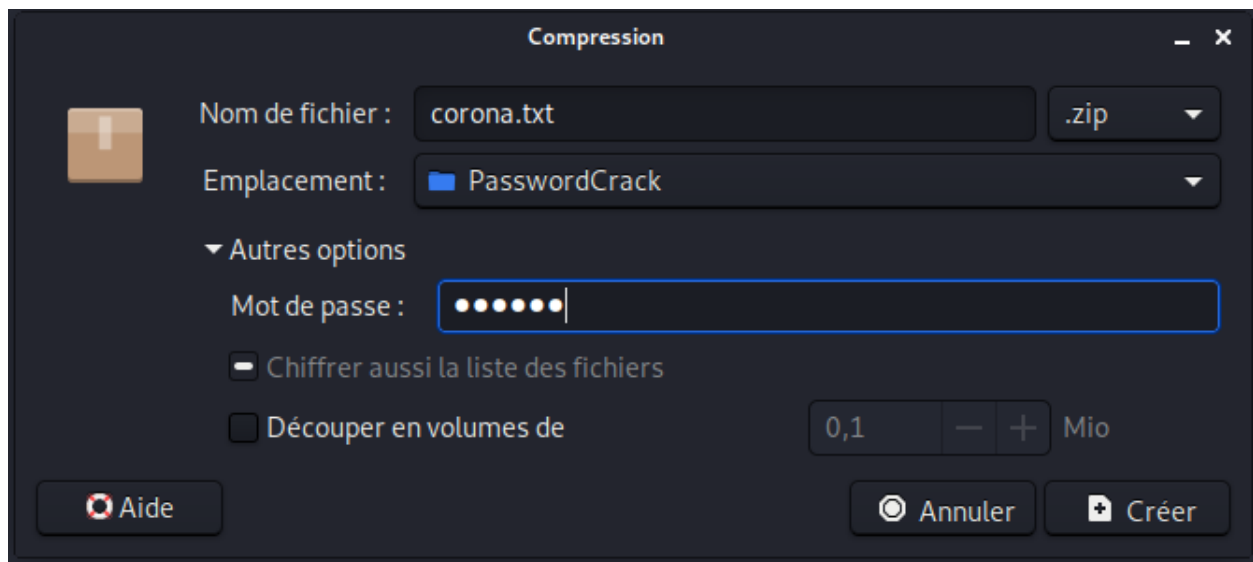
Nous n'avons pas besoin d'installer un logiciel en particulier pour utiliser le format .zip, la solution est nativement installée sur Kali, le système d'exploitation utilisé pour mener cette attaque.

Nous voulons protéger l'envoi d'un fichier contenant le mot de passe d'accès à un compte, renfermant des données sensibles sur la stratégie de lutte contre le coronavirus, liée au vaccin qui va être mis en place prochainement.

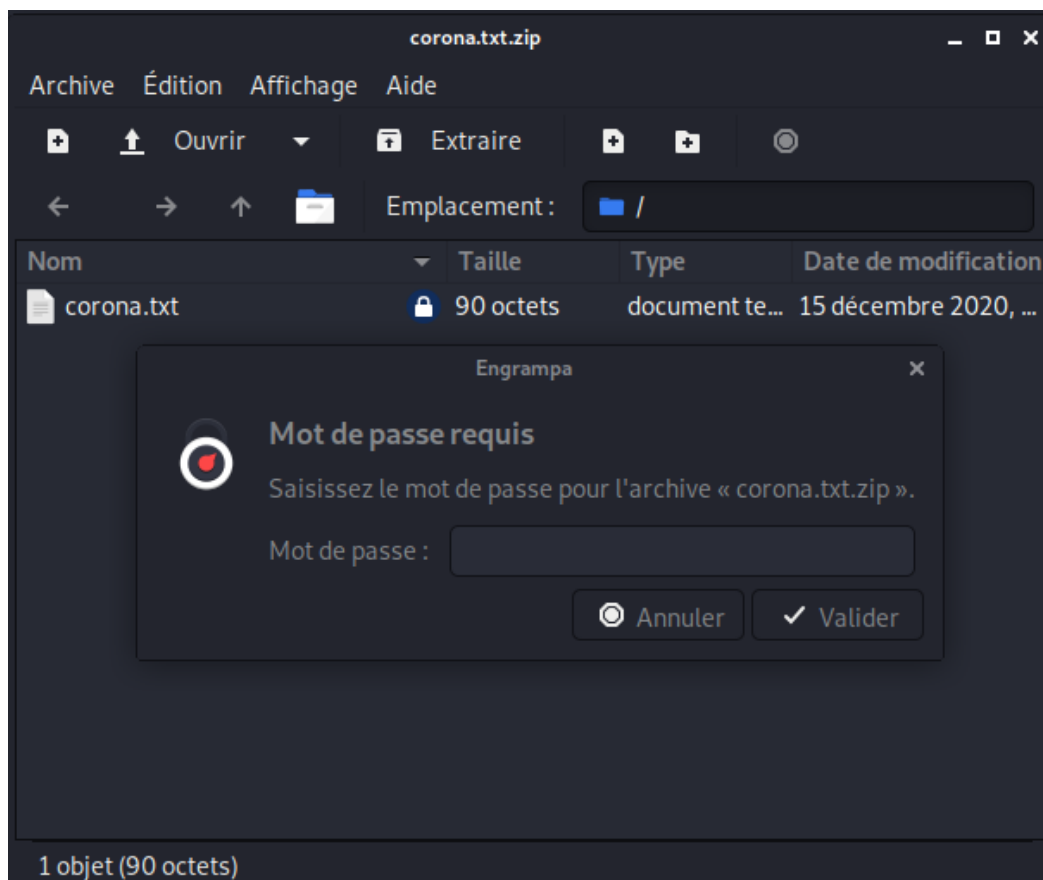


corona.txt	
~/Documents/Etude/Forensic/PasswordCrack	
1	Login
2	Password
	vaccin_corona
	4cC1Nc0v1D2020

Nous avons donc compressé ce fichier "**corona.txt**" en format .zip, en le sécurisant avec le mot de passe "**VACCIN**".



Suite à ces manipulations, nous pouvons constater que chiffrement s'est bien effectué, car lorsque nous essayons d'extraire le fichier, un mot de passe est demandé.



Extraction du hash

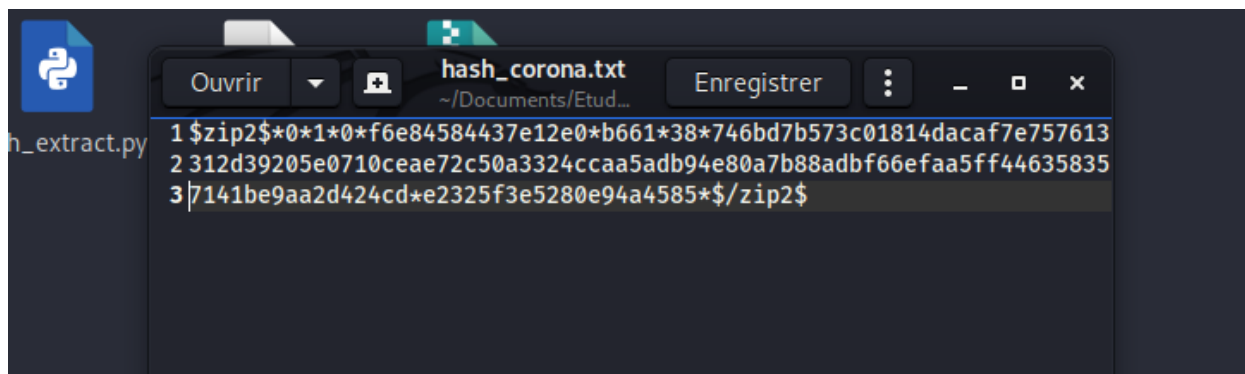
Pour extraire le hash du fichier zip chiffré, nous avons utilisé un élément du composant "John The Ripper", qui est *zip2john* permettant de faire l'extraction.

J'ai réalisé un script en python qui permet d'exécuter la commande et de récupérer son résultat, en isolant la partie de la réponse qui nous intéresse.

Commande utilisée pour faire l'extraction : *zip2john corona.txt.zip*

```
hash_extract.py X
home > kali > Documents > Etude > Forensic > PasswordCrack > hash_extract.py
1 import subprocess
2
3 output = subprocess.check_output("zip2john corona.txt.zip", shell=True)
4 output = str(output).split(':')
5
6 f = open("hash_corona.txt", "w")
7 f.write(output[1])
8 f.close()
```

Le fichier "hash_corona.txt" contient maintenant le hash que nous allons tenter d'attaquer avec le module "hashcat", dans la suite de cet exercice.



```
Ouvrir hash_corona.txt Enregistrer
~/Documents/Etud...
1 $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01814dacaf7e757613
2 312d39205e0710ceae72c50a3324ccaa5adb94e80a7b88adb66efaa5ff44635835
3 7141be9aa2d424cd*e2325f3e5280e94a4585*$/zip2$
```

Attaque par dictionnaire

Nous avons décidé de réaliser l'attaque par dictionnaire sur la hash du fichier chiffré, avec la wordlist connue "rockyou.txt".

Voici la commande utilisée pour mener l'attaque :

```
hashcat -m 13600 -a 0 hash_corona.txt rockyou.txt -D 1
```

Le mot de passe "VACCIN" a été trouvé par la wordlist en 3 min 57 sec.

```

$ hashcat -m 13600 -a 0 hash_corona.txt rockyou.txt -D 1
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.5, None+Asserts, LLVM 9.0.1, RELOC, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]

* Device #1: pthread-Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, 13779/13843 MB (4096 MB allocatable), 4MCU

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

Applicable optimizers applied:
* Zero-Byte
* Single-Hash
* Single-Salt
* Slow-Hash-SIMD-LOOP

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 65 MB

Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace...: 14344385
* Runtime...: 1 sec

$zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01814dacaf7e757613312d39205e0710ceae72c50a3324ccaa5adb94e80a7b88adb66efaa5ff446358357141be9aa2d424cd*e2325f3e5280e94a4585*$ /zip2$:VACCIN

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: WinZip
Hash.Target.....: $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01... /zip2$
Time.Started.....: Tue Dec 15 14:06:09 2020 (3 mins, 57 secs)
Time.Estimated...: Tue Dec 15 14:10:06 2020 (0 secs)
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 46628 H/s (10.35ms) @ Accel:128 Loops:999 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 10519040/14344385 (73.33%)
Rejected.....: 0/10519040 (0.00%)
Restore.Point...: 10518528/14344385 (73.33%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-999
Candidates.#1...: VALERIA05 -> VA5SONR

Started: Tue Dec 15 14:06:07 2020
Stopped: Tue Dec 15 14:10:07 2020

```

Attaque par mask

Afin de réduire le coût de l'attaque au vu du contexte expérimentale, nous avons effectué un bruteforce directement sur 6 caractères, avec un mask "?u?u?u?u?u" correspondant à des lettres en majuscule :

```
u | ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Le temps complet de l'attaque est estimé à 1 heure et 57 minutes, mais le mot de passe utilisé a été trouvé en 5 minutes et 16 secondes :

```

$zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01814dacaf7e757613312d39205e0710ceae72c50a3324ccaa5adb94e80a7b88adb66efaa5ff446358357141be9aa2d424cd*e2325f3e5280e94a4585*$ /zip2$:VACCIN

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: WinZip
Hash.Target.....: $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01... /zip2$
Time.Started.....: Tue Dec 15 14:20:32 2020 (5 mins, 16 secs)
Time.Estimated...: Tue Dec 15 14:25:48 2020 (0 secs)
Guess.Mask.....: ?u?u?u?u?u [6]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 43931 H/s (10.72ms) @ Accel:128 Loops:999 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 14294016/308915776 (4.63%)
Rejected.....: 0/14294016 (0.00%)
Restore.Point...: 549376/11881376 (4.62%)
Restore.Sub.#1...: Salt:0 Amplifier:19-20 Iteration:0-999
Candidates.#1...: VFPRYA -> VQMSSE

```

Complexité

Le chiffrement utilisé dans cet exercice repose sur la fonction cryptographique de hachage standard du format ZIP2, qui est SHAd-256(m) qui correspond à la fonction suivante : SHA-256(SHA-256(m)), et utilise l'algorithme de chiffrement AES-128. ([Lien vers la source](#))

Attaque par dictionnaire :

En ce qui concerne l'attaque par dictionnaire pour cet algorithme, en utilisant un processeur I7 de 7ème génération, le temps est estimé à environ 5 minutes pour la wordlist rockyou.txt contenant 14 344 392 mots de passe, correspondant approximativement à 40 000 essais par seconde.

```

Session.....: hashcat
Status.....: Running
Hash.Name.....: WinZip
Hash.Target.....: $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01 ... /zip2$
Time.Started.....: Tue Dec 15 15:43:27 2020 (1 min, 19 secs)
Time.Estimated...: Tue Dec 15 15:49:13 2020 (4 mins, 27 secs)
Guess.Base.....: File (rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 40410 H/s (10.54ms) @ Accel:128 Loops:999 Thr:1 Vec:8
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 3522048/14344385 (24.55%)
Rejected.....: 0/3522048 (0.00%)
Restore.Point....: 3522048/14344385 (24.55%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-999
Candidates.#1....: stnick77 → stloney

```

Attaque par mask :

Concernant l'attaque par mask, pour des raisons expérimentales nous avons limité l'attaque à 6 caractères avec des lettres majuscules seulement, le temps était estimé à 1 heure et 50 minutes pour 308 915 776 de combinaisons possibles, correspondant à 26^6 (26 correspondant aux nombres de lettre [A-Z] et 6 la longueur du mot de passe).

```

Session.....: hashcat
Status.....: Running
Hash.Name.....: WinZip
Hash.Target.....: $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01 ... /zip2$
Time.Started.....: Tue Dec 15 15:54:03 2020 (58 secs)
Time.Estimated...: Tue Dec 15 17:44:46 2020 (1 hour, 49 mins)
Guess.Mask.....: ?u?u?u?u?u [6]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 46486 H/s (10.42ms) @ Accel:128 Loops:999 Thr:1 Vec:8
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 2801152/308915776 (0.91%)
Rejected.....: 0/2801152 (0.00%)
Restore.Point....: 107520/11881376 (0.90%)
Restore.Sub.#1...: Salt:0 Amplifier:11-12 Iteration:0-999
Candidates.#1....: KSTHYO → KOFICE

```

Bien sûr, en situation réelle, un attaquant devra essayer toutes les combinaisons possibles avec différents types de caractères, des lettres en minuscules et en majuscules, des chiffres et des caractères, avec une longueur de mot de passe généralement entre 6 et 10 caractères.

Nous avons réalisé des tests pour avoir une estimation de temps sur ce type d'attaque en situation réelle, en utilisant la commande suivante :

```
hashcat -m 13600 -a 3 hash_corona.txt -i --increment-min=6 --increment-max=10 ?a?a?a?a?a -D 1
```

Sur la commande exécutée, le `?a` correspond à n'importe quel type de caractère.

```

Session.....: hashcat
Status.....: Running
Hash.Name.....: WinZip
Hash.Target.....: $zip2$*0*1*0*f6e84584437e12e0*b661*38*746bd7b573c01 ... /zip2$
Time.Started.....: Tue Dec 15 16:03:57 2020 (2 mins, 45 secs)
Time.Estimated...: Wed Jun 30 03:06:31 2021 (196 days, 9 hours)
Guess.Mask.....: ?a?a?a?a?a [6]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 43316 H/s (10.59ms) @ Accel:128 Loops:999 Thr:1 Vec:8
Recovered.....: 0/1 (0.00%) Digests
Progress.....: 7375872/735091890625 (0.00%)
Rejected.....: 0/7375872 (0.00%)
Restore.Point....: 77312/7737809375 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:61-62 Iteration:0-999
Candidates.#1....: @;GBON → @6F123

```

Le temps est estimé à 196 jours et 9 heures, pour tester toutes les combinaisons entre 6 et 10 caractères.

Conclusion

Le chiffrement standard du format ZIP2 est relativement faible, sa sécurité repose la complexité du mot de passe.

Le mot de passe utilisé ne doit pas être commun, car comme nous avons pu le voir, avec un processeur ordinaire, nous pouvons parcourir une wordlist de plus de 14 millions de mots de passe en 5 minutes. Il doit avoir une longueur d'au moins 12 caractères, avec plusieurs types de caractères pour faire augmenter le nombre de combinaisons possibles.

Cette méthode n'est pas conseillée pour sécuriser des fichiers de haute importance, comme recommandé sur différents sites, ce chiffrement reste adapté à des utilisateurs utilisant le chiffrement de zip par mot de passe à une fréquence relativement basse.

L'avantage à l'utiliser est que tout le monde est en mesure de déchiffrer s'il a le mot de passe car c'est un chiffrement standard, nativement compris par la majorité des machines.

Cependant si on souhaite privilégier le transfert des données de manières sécurisées, il est recommandé d'utiliser des solutions proposant un chiffrement AES-256 pour sécuriser des fichiers de haute importance. Bien sûr l'inconvénient est que l'interlocuteur doit posséder une solution comprenant le même algorithme de chiffrement/déchiffrement pour pouvoir accéder au fichier.