L'importance des hash en cybersécurité

Objectifs du Workshop

- Comprendre les bases des fonctions de hachage et leur rôle en cybersécurité.
- Explorer leurs applications pratiques: stockage de mots de passe, intégrité des fichiers, signatures numériques.
- Mettre en pratique des outils pour générer et cracker des hashs.

Prérequis

- **Hashcat** ou **John the Ripper** **installés**.
- Python.
- Un terminal ou environnement Linux.
- Les fichiers nécessaires du repo https://github.com/N1borg/Workshop-hash.

Introduction

I. Comprendre les fonctions de hachage

1.1 Qu’est-ce qu’une fonction de hachage ?

Une **fonction de hachage** est un algorithme qui prend une donnée en entrée (texte, fichier) et produit une **empreinte numérique unique de taille fixe**, appelée **hash**.

Propriétés essentielles des fonctions de hachage:

1. Déterminisme:

- Un même message produit toujours le même hash.
- Exemple:

```
echo -n "password" | sha256sum
```

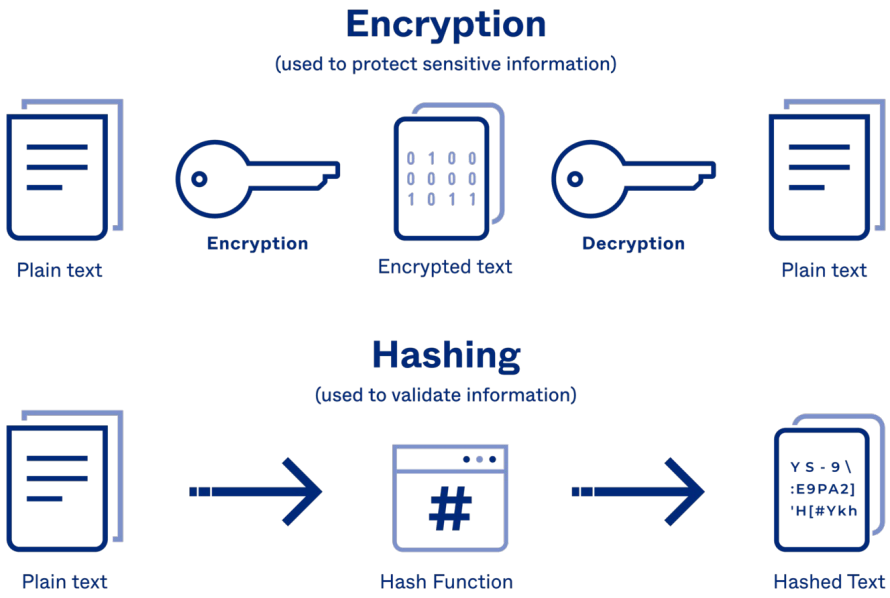
Hash: `5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8`.
Répétez la commande avec "password" et vous obtiendrez toujours le même résultat.

2. Résistance aux collisions:

- Deux entrées différentes ne doivent pas générer le même hash.
- Avec des algorithmes anciens comme MD5 ou SHA-1, il existe des collisions connues, ce qui les rend non sécurisés pour certaines applications.

3. Irreversibilité:

- Un hash ne peut pas être "inversé" pour retrouver l'entrée d'origine.
- Exemple: Avec `5e884898da280471...`, il est impossible de savoir directement que l'entrée était "password".
- Cependant, des attaques comme le bruteforce ou les rainbow tables peuvent contourner cette propriété pour des hashs faibles ou non salés.



Exemples courants d’algorithmes de hachage:

1. MD5:

- Rapide mais obsolète. Vulnérable aux collisions.
- Utilisé encore pour des cas non critiques (exemple: vérification rapide d'intégrité de fichiers).

2. SHA-1:

- Plus sécurisé que MD5, mais des collisions ont été démontrées. Déconseillé pour la sécurité.

3. SHA-256:

- Fiable, utilisé pour de nombreux cas (exemple: signatures numériques, certificats SSL, blockchain).

4. Bcrypt et Argon2:

- Lent, conçu spécialement pour le stockage sécurisé des mots de passe.

1.2 Applications pratiques des fonctions de hachage

1. Stockage sécurisé des mots de passe

Au lieu de stocker un mot de passe en clair, on stocke son hash. Lorsqu'un utilisateur tente de se connecter, on hache le mot de passe qu'il fournit, puis on compare les deux hashs.

- Exemple sans sel (non recommandé):

```
echo -n "password" | sha256sum
```

Stockez le résultat dans une base de données.

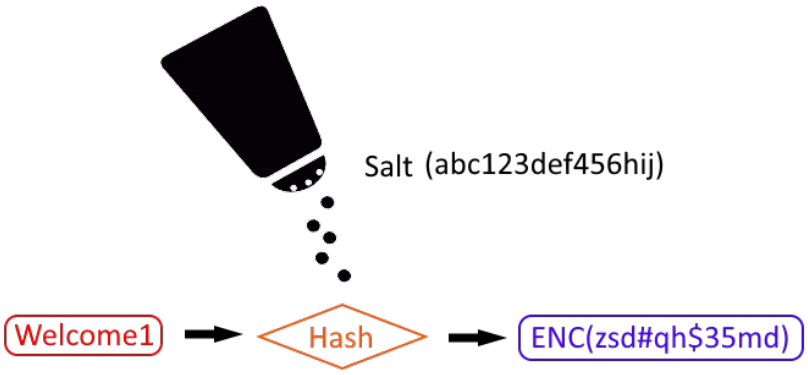
- Exemple avec un sel (sécurisé) en Python:

```
import bcrypt

password = b"mon_mot_de_passe"

salt = bcrypt.gensalt()

hashed_password = bcrypt.hashpw(password, salt)
print(hashed_password)
```



2. Vérification de l’intégrité des fichiers

On calcule un hash pour un fichier. Si ce fichier est modifié, son hash change. Cela permet de détecter des corruptions ou des manipulations malveillantes.

Exemple pratique:

1. Calculez le hash d’un fichier:

```
sha256sum fichier1.txt
```

```
Hash obtenu: 74603ef588...
```

1. Modifiez le fichier (ajoutez un espace, changez un caractère).
2. Recalculez le hash. Vous constaterez que même un petit changement entraîne un hash complètement différent.

3. Signatures numériques et certificats

Les hash sont utilisés dans les signatures numériques pour garantir l’authenticité et l’intégrité d’un message. Par exemple, les certificats SSL/TLS utilisent SHA-256 pour sécuriser les connexions internet.

---

2. Partie Théorique: Comparaison et Sécurité des Algorithmes

2.1 Comparaison entre MD5, SHA-1 et SHA-256

Algorithme	Taille du Hash	Sécurité	Utilisation actuelle
MD5	128 bits	Faible	Vérifications rapides, pas pour la sécurité.
SHA-1	160 bits	Moyenne	Déprécié, remplacé par SHA-256.
SHA-256	256 bits	Forte	Signatures numériques, blockchain, stockage sécurisé.

2.2 Pourquoi utiliser Bcrypt ou Argon2 pour les mots de passe ?

Les algorithmes comme Bcrypt ou Argon2 incluent:

1. **Un sel (salt)**: Rend chaque hash unique même pour un mot de passe identique.
2. **Un facteur de travail (cost)**: Contrôle la lenteur du calcul, ce qui rend le bruteforce coûteux pour un attaquant.

2.3 Exemple de collision avec MD5

Voici deux fichiers (`message1.bin et message2.bin`) différents qui produisent le même hash MD5 (collision):

```
$ diff message1.bin message2.bin
Binary files message1.bin and message2.bin differ
$ md5sum message1.bin
008ee33a[...] message1.bin
$ md5sum message2.bin
008ee33a[...] message2.bin
```

Cela montre que MD5 est vulnérable, et ne doit pas être utilisé pour des cas nécessitant de la sécurité.

Calculez-leur hash.

---

3. Partie Pratique

Exercice 1: Génération de Hashs

1. Générer un hash avec MD5:

$ echo -n "password" | md5sum

- Résultat: `5f4dcc3b...`
- Changez l’entrée:

$ echo -n "Password" | md5sum

- Résultat: `dc647eb6...`

Remarquez comment une simple différence (majuscule) change complètement le hash.

Exercice 2: Brute Force et Rainbow Tables

1. Créez un fichier contenant un hash MD5 (par exemple `482c811da5d5b4bc6d497ffa98491e38` pour "password123").

2. Utilisez John the Ripper pour cracker ce hash:

```
john --format=raw-md5 hashfile.txt
```

John tentera toutes les combinaisons possibles jusqu'à trouver le mot de passe correspondant.

Exercice 3: Attaque sur hash salé

Crackez les hashs fournis dans un fichier de test. Certains hashs sont avec sel, d'autres sans.
Utilisez les outils comme Hashcat et John the Ripper pour relever le défi.

Exercice bonus

1. **Vérifiez l'intégrité d'un fichier**: Modifiez un fichier et observez comment le hash change.
2. **Générez une collision MD5**: Essayez des outils en ligne ou cherchez des collisions connues.
3. **Implémentez un algorithme de hachage**: Créez une fonction Python pour comprendre le principe.

Exercices Bonus

1. Vérification de l'intégrité d'un fichier:

- Calculer le hash d’un fichier et comparer avec un hash fourni:

  ```
  sha256sum fichier1.txt
  ```

- Cas pratique: Modification d’un fichier pour observer le changement de hash.

2. Collision MD5:

- Montrer deux fichiers différents ayant le même hash MD5 (exemple théorique ou avec des outils).

3. Implémentation d’un mini-algorithme de hachage:

- Utilisation d’un langage comme Python pour créer une fonction de hachage simple (non sécurisée) afin de comprendre le fonctionnement.

Ressources

- Hash
- Documentation SHA-256
- John the Ripper
- Hashcat
- OWASP Password Storage Cheatsheet