Challenge Distributeur Automatique 26 étage 2

Write-up : Attaque par Collision sur SHA-256 Tronqué

1. Contexte du Challenge

Nous ciblons l'interface d'administration d'un distributeur automatique. Pour obtenir les droits d'administration, le système demande de fournir deux clés différentes qui produisent la même empreinte cryptographique.

Données fournies :

- **Opérateur :** Scarlette
- **Fonction de hachage :** SHA-256 tronqué aux 56 premiers bits.
- **Format des clés :** `Scarlette_[suffixe_aléatoire]`

L'objectif est de trouver une **collision** : deux chaînes distinctes K_1 et K_2 telles que $H(K_1) = H(K_2)$ sur les 56 premiers bits.

2. Analyse de la Vulnérabilité

2.1 Identification de la Faille

La sécurité repose sur une version tronquée de SHA-256 ne conservant que 56 bits. Bien que SHA-256 complet (256 bits) soit résistant aux collisions, réduire l'espace de sortie à 56 bits rend l'attaque réalisable grâce au **paradoxe des anniversaires**.

Points clés :

1. **Espace de recherche :** 2^{56} possibilités pour le hash.
2. **Paradoxe des anniversaires :** La probabilité de trouver une collision atteint 50% après avoir généré environ $\sqrt{2^{56}} = 2^{28}$ hashs.
3. **Complexité estimée :** $2^{28} \approx 268$ millions d'itérations. C'est un nombre calculable en un temps raisonnable (quelques minutes à une heure) sur un ordinateur moderne.

2.2 Stratégie d'Attaque

Nous utilisons une approche par **dictionnaire naïf** (bien que la méthode ρ de Pollard consommerait moins de mémoire, la méthode naïve est plus simple à implémenter si l'on dispose de ~16 Go de RAM).

1. Générer des clés aléatoires valides (`Scarlette_...`).
2. Calculer leur hash tronqué (56 bits).

3. Stocker chaque couple `(hash: clé)` dans un dictionnaire.
4. Si un hash calculé existe déjà dans le dictionnaire avec une clé différente, la collision est trouvée.

3. Implémentation de l'Exploit

3.1 Code de l'Attaque

Le script Python suivant implémente l'attaque des anniversaires. Il génère des millions de candidats jusqu'à trouver une correspondance.

```
import hashlib
import secrets
import time

def get_56bit_prefix(data_bytes: bytes) -> int:
    """Extrait les 56 premiers bits du hash SHA-256."""
    h = hashlib.sha256(data_bytes).digest()
    # On prend les 7 premiers octets (7 * 8 = 56 bits)
    return int.from_bytes(h[:7], byteorder='big')

def generate_random_candidate(operator_name: str) -> tuple:
    """Génère un candidat: Scarlette_[16_chars_hex_random]"""
    random_suffix = secrets.token_hex(16)
    clef_str = f"{operator_name}_{random_suffix}"
    return clef_str, clef_str.encode()

def naive_collision_attack(operator_name="Scarlette"):
    D = {} # Dictionnaire pour stocker les hashs vus
    iteration_count = 0
    start_time = time.time()

    print(f"[*] Démarrage de l'attaque collision sur {operator_name}...")

    while True:
        iteration_count += 1

        # 1. Générer candidat
        x_str, x_bytes = generate_random_candidate(operator_name)

        # 2. Calculer hash tronqué
        h = get_56bit_prefix(x_bytes)

        # 3. Vérifier collision
        if h in D:
```

```
            existing_key = D[h]
            if existing_key != x_str:
                # Collision trouvée !
                elapsed = time.time() - start_time
                return existing_key, x_str, elapsed, iteration_count

        # 4. Stocker
        D[h] = x_str

        # Log périodique
        if iteration_count % 1000000 == 0:
            print(f"Iter: {iteration_count:,} | Dict size: {len(D):,}")
```

4. Résultats

Après exécution du script, nous avons obtenu une collision valide.

Statistiques de l'attaque :

- **Temps écoulé :** ~27 minutes (1643.23s)
- **Itérations :** 411,681,115 (approx. 1.5×2^{28})
- **Vitesse :** ~250k hash/s

4.1 Clés trouvées

Les deux clés suivantes produisent exactement la même empreinte SHA-256 sur les 56 premiers bits.

Clef 1 :
`Scarlette_dccbe03f029dc4fc84fb7d20348a7a93`

Clef 2 :
`Scarlette_a867d99ee45b7225c289d298c7229c4c`

4.2 Exploitation sur le Distributeur

Pour valider le challenge, nous convertissons ces clés en hexadécimal et les injectons dans le menu ADMIN du distributeur.

Valeurs Hexadécimales à saisir :

1. **Clef 1 (Hex) :**

```
536361726c657474655f6463636265303366303239646334666338346662376432303334386137613933
```

2. **Clef 2 (Hex) :**

```
536361726c657474655f6138363764393965653435623732323563323839643239386337323239633463
```

Résultat final :

`[✓] Accès administrateur accordé par collision !`