Attacchi a password

Introduction

Nonostante le funzioni di hash siano irreversibili, risultano comunque attaccabili.
Sono due i principali attacchi alle funzioni hash: **attacco dizionario**, **attacco rainbow table**

Attacco dizionario

Uno dei problemi è che ci sono un numero infinito di password che possono risultare nello stesso hash. Come facciamo quindi a scoprire qual è quella giusta?

Fortunatamente per gli attaccanti gli utenti non vogliono (o non possono) ricordarsi password complesse e non vogliono ricordarsene molte diverse
Come conseguenze si ha che le password risultano brevi e semplici e che la stessa password probabilmente è riusata molte volte per servizi diversi

L'attacco del dizionario consiste nel compilare una lista di password comunemente utilizzata e fare un *bruteforce*. Dunque per ogni password nella mia lista, calcolo l'hash e verifico se è uguale a quello della password

> **Info**
>
> Esistono oggi dizionari di svariati GB di password (es. rockyou)

Vantaggi

- Molto semplice da effettuare (richiede solo una lista di password)
- Versatile → funziona per qualsiasi funzione hash, password, ecc
- Moltissimi tool disponibili per automatizzare il tutto (es. John the Ripper)
  Svantaggi
- Può essere molto lento, in quanto richiede la computazione in real time degli hash
- La password può non essere presente nel dizionario

Attacco Rainbow Table

Questo risulta essere un attacco dizionario ma migliorato sull'efficienza. Infatti in questo caso il nostro dizionario è composto da password e hash corrispondente, in modo tale che non bisogna calcolare l'hash in real-time ma mi basta confrontare l'hash calcolato con l'hash da trovare e, in caso di corrispondenza, vedere a quale password corrisponde (nella realtà utilizza un sistema più complesso di funzioni di riduzione per mantenere trattabili le dimensioni della tabella)

Vantaggi

- Molto semplice da effettuare, data la rainbow table precompilata
- Molto più veloce rispetto a dictionary bruteforce

  Svantaggi
- Rigidità → funziona solo per la funzione di hash per la quale è stata creata la rainbow table (se vogliamo più funzioni servono più rainbow tables)

Salvati dal sale

A proteggersi da questi attacchi ci pensa il `salt`
Il `salt` infatti è un valore randomico, generato quando un utente sceglie la password, che viene aggiunto alla computazione dell'hash (prima $d = hash(x)$, dopo $d = hash(x, s)$)

Il salt viene poi salvato in chiaro assieme all'hash calcolato rendendo impossibile l'uso di rainbow tables (se per ogni utente c'è un salt randomico diverso, non posso precomputare gli hash). Come conseguenza inoltre si ha che due utenti con la stessa password hanno hash diversi