Password hashing in PHP

Salt, costo computazionale e algoritmi

Prof. Fedeli Massimo - Tutti i diritti riservati

Perché usare una salt

La **salt** è una stringa casuale aggiunta alla password prima del calcolo dell'hash.

Serve a:

- impedire l'uso di rainbow table
- evitare hash identici per password identiche
- rendere gli attacchi offline più costosi

Ogni password deve avere una salt diversa.

password_hash: cosa produce

La funzione password_hash restituisce una stringa strutturata che contiene:

- algoritmo utilizzato
- parametri di costo
- salt generata automaticamente
- hash finale

Tutte queste informazioni sono incorporate nell'hash stesso.

Come funziona `password_verify`

Quando si usa `password_verify(password, hash)`:

1. PHP legge algoritmo e parametri dall'hash
2. estrae la salt contenuta nell'hash
3. ricalcola l'hash con la password inserita
4. confronta i risultati in modo sicuro

La verifica funziona anche con salt diverse per ogni password.

Il concetto di costo computazionale

Il **costo** indica quanto è oneroso calcolare l'hash.

Aumentare il costo significa:

- rallentare il login (di pochi millisecondi)
- rallentare enormemente gli attacchi a forza bruta

Il costo è un compromesso tra sicurezza e prestazioni.

Costo negli algoritmi

Esempi:

- **bcrypt**: costo espresso come numero di iterazioni (cost)
- **Argon2**: costo basato su tempo, memoria e parallelismo

Il costo è memorizzato nell'hash e usato automaticamente in fase di verifica.

Algoritmi disponibili in password_hash

PHP supporta diversi algoritmi:

- **PASSWORD_BCRYPT**
- **PASSWORD_ARGON2I**
- **PASSWORD_ARGON2ID**
- **PASSWORD_DEFAULT**

PASSWORD_DEFAULT seleziona l'algoritmo migliore disponibile nella versione di PHP.

Confronto tra gli algoritmi

- **bcrypt**
  - molto diffuso e collaudato
  - resistente, ma non memory-hard
- **Argon2**
  - vincitore del Password Hashing Competition
  - progettato per contrastare GPU e ASIC

Argon2id è oggi la scelta consigliata.

Perché non gestire salt e costo manualmente

Gestire manualmente questi aspetti:

- aumenta il rischio di errori
- non migliora la sicurezza
- riduce la manutenibilità

Le funzioni di PHP sono progettate per un uso sicuro di default.

Conclusione

- ogni password ha una salt unica
- l'hash contiene algoritmo, costo e salt
- `password_verify` ricostruisce il processo automaticamente
- il costo rende gli attacchi computazionalmente proibitivi

In sicurezza, la semplicità controllata è una virtù.