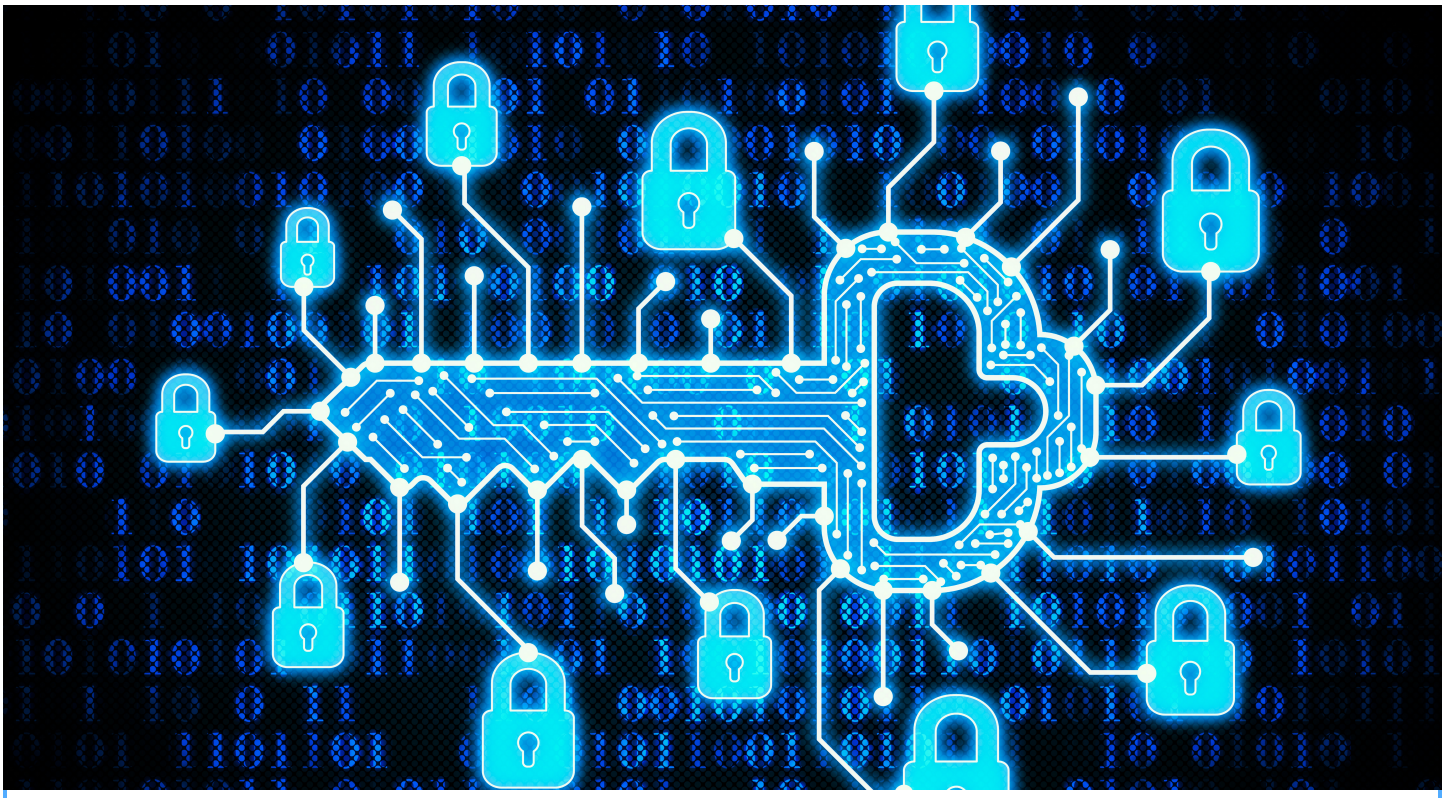
Autenticazione e password sicure

DI MASSIMO VAILATI
03/03/2026

PASSWORD HASH AUTENTICAZIONE SICUREZZA INFORMATICA FORZA BRUTA
SCIENTIFICO S.A. , TE SIA , TT INFORMATICA



Contenuto: l'importanza di scegliere password robuste e di gestirle correttamente, illustrando i rischi delle password deboli e le tecniche di attacco più comuni, come il brute force e l'uso di informazioni personali, e introduce il concetto di hash per proteggere le credenziali.

Attività pratica: simulazioni di controllo e attacco a password deboli, analisi della complessità degli algoritmi di brute force, e esercizi sull'uso degli hash per comprendere come vengono gestite e protette le password nei sistemi reali.

Autori

MASSIMO VAILATI

Un laboratorio per mostrare quanto sia facile forzare password deboli

La sicurezza informatica inizia spesso da un elemento semplice, ma fondamentale: **la password**. Nonostante anni di sensibilizzazione, molti utenti continuano a scegliere chiavi di accesso deboli, prevedibili o riutilizzate. Per questo, una delle attività più efficaci con gli studenti consiste nel mostrare - in un ambiente controllato e sicuro - quanto sia facile violare una password poco robusta.

In questo articolo proponiamo una simulazione di **attacco a forza bruta** su *password fittizie* memorizzate localmente. L'obiettivo non è imparare a "craccare", ma comprendere **perché servono password lunghe, complesse e gestite correttamente**.

Al termine dell'attività, gli studenti dovrebbero:

- comprendere il funzionamento di base dell'autenticazione con password;
- riconoscere le caratteristiche di una password forte;
- comprendere la differenza tra confronto diretto e confronto tramite **hash**;
- osservare come cambia il tempo di attacco al variare della complessità della password;
- sviluppare senso critico nell'uso quotidiano delle credenziali.

COS'È UNA PASSWORD SICURA?

Una password robusta dovrebbe includere:

- lunghezza minima di almeno **12 caratteri**;
- combinazione di lettere maiuscole e minuscole, numeri e simboli;
- assenza di parole del dizionario o informazioni personali;
- unicità: non deve essere usata su più servizi.

PERCHÉ LE PASSWORD DEBOLI SONO VULNERABILI?

Perché le tecniche di attacco automatizzate possono provare migliaia (o milioni) di combinazioni al secondo. Anche senza accedere a strumenti avanzati, un semplice programma scritto in classe può dimostrare la vulnerabilità di password troppo brevi o prevedibili.

Anche una password relativamente lunga può essere **facile da indovinare** se contiene:

- parole comuni ("casa", "password", "soleluna", "ciaoamici")
- nomi propri ("mario", "luisa", "federico")
- informazioni personali ("mariorossi1999", "giulia2005", "torino2024")

Questo accade per diversi motivi:

1. Gli attacchi a dizionario testano prima parole reali, non combinazioni casuali

Gli aggressori non provano tutte le combinazioni in ordine alfabetico: usano **wordlist** con milioni di parole frequenti (dizionari reali, nomi, cognomi, luoghi, sport, slang, password più usate).
In questo modo, anche una password come "`gattorosso2023`" può cadere in pochi millisecondi perché:

- contiene una parola del dizionario ("gatto")
- contiene una parola comune ("rosso")
- numeri finali molto prevedibili ("2023")

2. Le informazioni personali sono spesso pubbliche

Molte tecniche di attacco sfruttano anche dati presi da:

- social network
- indirizzi email
- nomi dei familiari
- luogo di nascita
- compleanni

Se uno studente usa una password come: “Martina2008” o “Juventusforever!” può diventare facilmente prevedibile per chiunque conosca qualcosa su di lui/lei.

3. Gli attacchi moderni sono “intelligenti”, non casuali
Gli strumenti di cracking attuali:

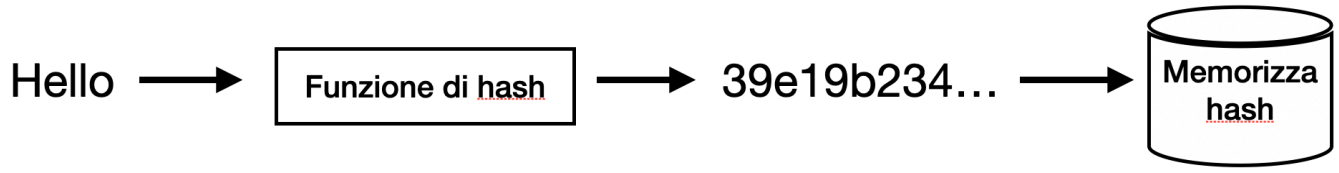
- combinano parole del dizionario tra loro
- aggiungono numeri comuni alla fine (123, 2024…)
- applicano trasformazioni tipiche (“a” → “@”, “o” → “0”, “e” → “&”)

Quindi anche password come: “P@ssw0rd2024!” sono considerate estremamente deboli, perché sono semplici varianti di “password”.

PERCHÉ SI USA L’HASH NELLE PASSWORD?

Quando un sistema deve utilizzare una password, non la memorizza in chiaro, ma salva solo il suo **hash**, cioè il risultato di una funzione matematica che trasforma il testo in una sequenza di caratteri apparentemente casuale. L’aspetto importante è che questa trasformazione è **a senso unico**: dall’hash non è possibile risalire alla password originale. Questo offre due vantaggi fondamentali.

1. Anche se un aggressore ottiene l’archivio delle password di un sito, non troverà le password vere ma solo gli hash, molto più difficili da sfruttare.
2. Durante l’autenticazione, il sistema non confronta la password inserita con quella salvata, ma confronta gli **hash**, riducendo ulteriormente il rischio di esposizione. È un meccanismo semplice ma potentissimo per limitare i danni in caso di violazione dei dati.



ATTIVITÀ PRATICA N. 1

Simulazione di confronto di una password

Il primo esempio (*controllapwd.c*) mostra semplicemente come funziona un controllo elementare.
Questo codice serve per capire:

- come avviene il confronto;
- perché memorizzare una password in chiaro non è una buona pratica (è leggibile nel codice, se trasmessa in chiaro può essere intercettata ed usata).

```
#include <stdio.h>
#include <string.h>

int main() {
    char password_inserita[50];
    const char password_corretta[] = "abc123";

    printf("Inserisci la password: ");
    scanf("%49s", password_inserita);
```

```
    if (strcmp(password_inserita, password_corretta) == 0) {
        printf("Accesso consentito.\n");
    } else {
        printf("Accesso negato.\n");
    }
    return 0;
}
```

ATTIVITÀ PRATICA N. 2

Simulare un attacco a forza bruta su password molto semplici

Implementiamo un programma (*forzapwd.c*) che prova combinazioni su una password **brevissima e fittizia**, ad esempio composta solo da lettere minuscole e lunga 3.

Questa dimostrazione permette agli studenti di osservare:

- il numero di tentativi;
- il tempo di esecuzione;
- l'aumento esponenziale della complessità al crescere della lunghezza.

Suggerimento per il docente: provare password più lunghe e far stimare agli studenti il tempo teorico necessario.

```
#include <stdio.h>
#include <string.h>

int brute_force(const char *target) {
    char attempt[4];
    int tentativi = 0;

    for (char a = 'a'; a <= 'z'; a++) {
        for (char b = 'a'; b <= 'z'; b++) {
            for (char c = 'a'; c <= 'z'; c++) {
                attempt[0] = a;
                attempt[1] = b;
                attempt[2] = c;
                attempt[3] = '\0';

                tentativi++;

                if (strcmp(attempt, target) == 0) {
                    printf("Trovata password: %s\n", attempt);
                    return 1;
                }
                else {
                    printf("Tentativo %d fallito: %s\n", tentativi,
attempt);
                }
            }
        }
    }
    return 0;
}
```

```
int main() {
    const char password[] = "dog";  // password fittizia
    brute_force(password);
    return 0;
}
```

Osservazione: l'algoritmo esplora tutte le possibili combinazioni di tre caratteri in ordine alfabetico, partendo da 'a' fino a 'z'. Di conseguenza, se la password da individuare inizia con lettere poste verso la fine dell'alfabeto - ad esempio 'z' - il numero di tentativi necessari sarà maggiore, poiché tutte le combinazioni precedenti devono essere generate e confrontate prima di raggiungere quella corretta.

Per questo motivo, nell'analisi delle prestazioni di un algoritmo di brute force è importante considerare il *caso peggiore* (password trovata solo all'ultimo tentativo) o almeno il *caso medio*. La **complessità computazionale** dipende principalmente dal numero di possibili combinazioni, che a sua volta è determinato dal numero di caratteri disponibili e dalla lunghezza della password.

Nel caso specifico, con un alfabeto di 26 caratteri e una password di lunghezza n, il numero totale di combinazioni cresce secondo la formula: 26^n.
Questo comportamento esponenziale evidenzia perché gli attacchi brute force diventano rapidamente impraticabili al crescere della lunghezza della password.

ATTIVITÀ PRATICA N. 3

Hash delle password

Per provare password reali non memorizzate in chiaro, si può introdurre un semplice hash. In questo esempio usiamo una funzione di hash creata a scopo didattico, nei programmi reali si utilizzano funzioni di hash standard (ad esempio SHA-256) fornite da librerie affidabili. L'obiettivo è mostrare:

che confrontiamo hash, non password;
che il brute force richiede ora di calcolare l'hash di ogni tentativo.

Questo è il codice della funzione di hash che utilizziamo (*simplehash.c*).

```
#include <stdio.h>

// Funzione di hash semplice
// Calcola un hash sommando i valori ASCII dei caratteri
// moltiplicati per una costante (31) elevata alla posizione del carattere
unsigned int simple_hash(const char *str) {
    unsigned int hash = 0;

    while (*str) {
        hash = hash * 31 + (unsigned char)(*str);
        str++;
    }

    return hash;
}

int main() {
    const char *test = "hello!";
    printf("Hash: %u\n", simple_hash(test));
```

```
    return 0;
}
```

L'output del programma mostra l'hash della parola "hello!".

```
Hash: 3074032015
```

Modifichiamo il programma che prova combinazioni su una password **brevissima e fittizia**, ad esempio composta solo da lettere minuscole e lunga 3 memorizzata tramite il suo valore hash (*brute_hash.c*).

```
#include <stdio.h>
#include <string.h>

/* Funzione di hash semplice */
unsigned int simple_hash(const char *str) {
    unsigned int hash = 0;
    while (*str) {
        hash = hash * 31 + (unsigned char)(*str);
        str++;
    }
    return hash;
}

int brute_force(const char *target) {
    char attempt[4];
    int tentativi = 0;

    unsigned int target_hash = simple_hash(target);  // hash della password da indovinare

    for (char a = 'a'; a <= 'z'; a++) {
        for (char b = 'a'; b <= 'z'; b++) {
            for (char c = 'a'; c <= 'z'; c++) {
                attempt[0] = a;
                attempt[1] = b;
                attempt[2] = c;
                attempt[3] = '\0';

                tentativi++;

                if (simple_hash(attempt) == target_hash) {
                    printf("Trovata password: %s\n", attempt);
                    return 1;
                }
                else {
                    printf("Tentativo %d fallito: %s\n", tentativi, attempt);
                }
            }
        }
    }
    return 0;
}

int main() {
    const char password[] = "dog";  // password fittizia
    brute_force(password);
```

```
    return 0;
}
```