

# CS0424IT — ESERCITAZIONE S6L3

## PASSWORD CRACKING

*Simone La Porta*



---

### TRACCIA

L'obiettivo dell'esercizio è craccare una serie di password utilizzando qualsiasi tool o soluzione alternativa disponibile. Le password da craccare sono fornite in formato hash MD5 come segue:

- 5f4dcc3b5aa765d61d8327deb882cf99
- e99a18c428cb38d5f260853678922e03
- 8d3533d75ae2c3966d7e0d4fcc69216b
- 0d107d09f5bbe40cade3de5c71e9e9b7
- 5f4dcc3b5aa765d61d8327deb882cf99

### SVOLGIMENTO

In questa esercitazione, l'obiettivo era scoprire le password in chiaro associate agli hash forniti nel file `hashes.txt`. Questo processo è fondamentale nel campo della sicurezza informatica per evidenziare le debolezze delle password utilizzate e migliorare le misure di sicurezza.

---

Questi hash rappresentano le versioni crittografate di password in chiaro. L'algoritmo di hashing MD5 (Message Digest Algorithm 5) converte una stringa di lunghezza arbitraria in un valore hash di 128 bit, tipicamente rappresentato come una stringa esadecimale di 32 caratteri.

### *John The Ripper*

Il primo strumento utilizzato è il tool John the Ripper per il cracking di hash MD5.

Il comando utilizzato per eseguire il cracking degli hash è stato:

```
john --format=raw-md5 --incremental hashes.txt
```

Il comando john invoca il tool John the Ripper, uno strumento popolare per il cracking delle password. Le opzioni utilizzate sono:

- `--format=raw-md5`: specifica che gli hash nel file sono nel formato MD5 non salato.
- `--incremental`: attiva un attacco incrementale, che è un metodo di brute force che prova tutte le combinazioni di caratteri possibili fino a trovare una corrispondenza.

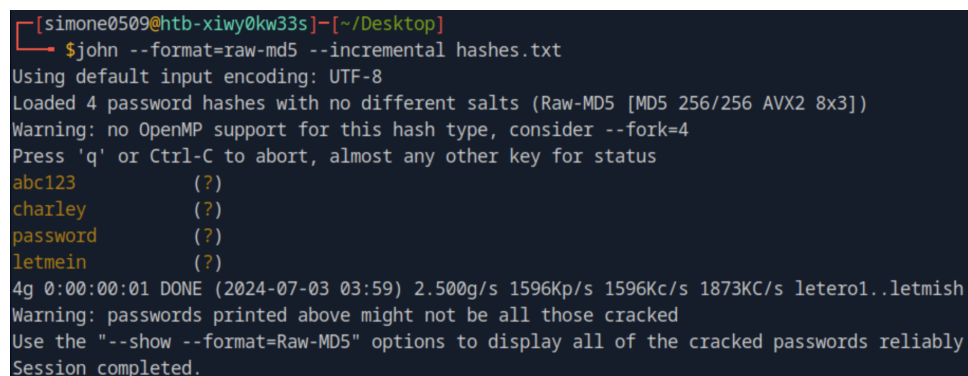
L'output del comando ha mostrato le seguenti password in chiaro:

abc123

charley

password

letmein



```
[simone0509@htb-xiwy0kw33s]~/Desktop
$ john --format=raw-md5 --incremental hashes.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
abc123      (?)
charley     (?)
password    (?)
letmein     (?)
4g 0:00:00:01 DONE (2024-07-03 03:59) 2.500g/s 1596Kp/s 1596Kc/s 1873KC/s letero1..letmish
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.
```

Queste password corrispondono agli hash forniti nel file di input. Il cracking delle password è stato completato in modo efficace, dimostrando la vulnerabilità degli hash MD5 non-salted alle tecniche di brute force.

---

## Hashcat

Il secondo strumento utilizzato è il tool Hashcat per il cracking di hash MD5.

Il comando utilizzato per eseguire il cracking degli hash è stato:

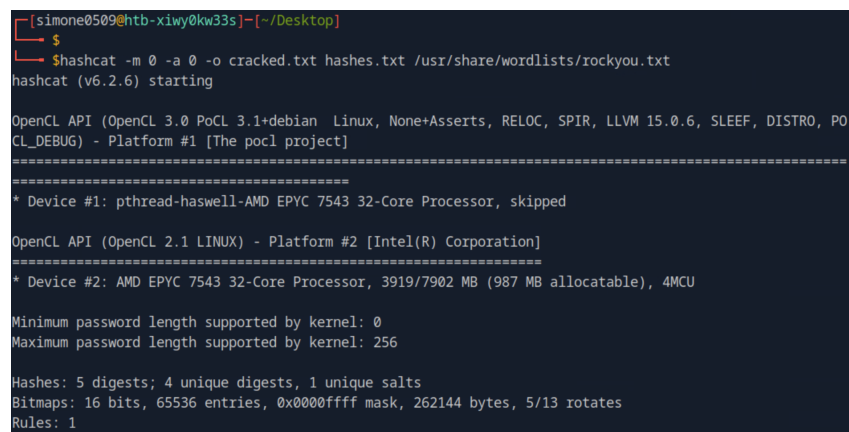
```
hashcat -m 0 -a 0 -o cracked.txt hashes.txt /usr/share/wordlists/rockyou.txt
```

Il comando hashcat invoca il tool Hashcat, uno strumento avanzato per il cracking delle password. Le opzioni utilizzate sono:

- `-m 0`: specifica che gli hash sono nel formato MD5.
- `-a 0`: specifica l'uso dell'attacco basato su dizionario.
- `-o cracked.txt`: specifica il file di output dove verranno salvate le password scoperte.
- `hashes.txt`: il file che contiene gli hash da crackare.
- `/usr/share/wordlists/rockyou.txt`: il file di dizionario contenente una lista di password comuni.

L'output del comando è stato salvato nel file `cracked.txt`, che contiene le seguenti associazioni hash-password:

```
5f4dcc3b5aa765d61d8327deb882cf99 : password
e99a18c428cb38d5f260853678922e03 : abc123
0d107d09f5bbe40cade3de5c71e9e9b7 : letmein
8d3533d75ae2c3966d7e0d4fcc69216b : charley
```



```
[simone0509@htb-xiwy0kw33s] ~/Desktop
$ hashcat -m 0 -a 0 -o cracked.txt hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 3.1+debian Linux, None+Asserts, RELOC, SPIR, LLVM 15.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=====
* Device #1: pthread-haswell-AMD EPYC 7543 32-Core Processor, skipped

OpenCL API (OpenCL 2.1 LINUX) - Platform #2 [Intel(R) Corporation]
=====
* Device #2: AMD EPYC 7543 32-Core Processor, 3919/7902 MB (987 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 5 digests; 4 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1
```

Queste password corrispondono agli hash forniti nel file di input. Il cracking delle password è stato completato in modo efficace, dimostrando la vulnerabilità degli hash MD5 alle tecniche di attacco basate su dizionario.

### Script Python

L'ultimo metodo utilizzato è stato uno script Python per il cracking di hash MD5 tramite tre metodi: attacco a dizionario, attacco basato su regole e attacco di forza bruta.

```
1 import hashlib
2 import itertools
3 import string
4 import time
5
6 # Hash delle password da craccare
7 hashes_to_crack = [
8     "49f68a5c8493ec2c0b180921c21fc6b",
9     "6a680ebec5ab4a5a59b40f72424ed4805",
10    "48d62159b3dfff56238e52e809138a8f",
11    "f13870bc274f6c49b3e31a0c672895f",
12    "8621ffdbc5698829397d97767ac13db3"]
13
14 # Funzione per craccare con dictionary attack
15 def dictionary_attack(hash_to_crack, wordlist):
16     start_time = time.time()
17     with open(wordlist, "r", encoding="utf-8", errors="ignore") as file:
18         for word in file:
19             word = word.strip()
20             hash_object = hashlib.md5(word.encode())
21             hashed_word = hash_object.hexdigest()
22             if hashed_word == hash_to_crack:
23                 end_time = time.time()
24                 elapsed_time = end_time - start_time
25                 print(f"Dictionary attack ha trovato la password: {word} in {elapsed_time:.2e} secondi")
26                 return word, elapsed_time
27     end_time = time.time()
28     elapsed_time = end_time - start_time
29     print(f"Dictionary attack non ha trovato la password in {elapsed_time:.2e} secondi")
30     return None, elapsed_time
31
32 # Funzione per craccare con brute-force attack
33 def brute_force_attack(hash_to_crack, max_length=6):
34     start_time = time.time()
35     chars = string.ascii_lowercase + string.digits
36     attempt = 0
37     for length in range(1, max_length + 1):
38         for guess in itertools.product(chars, repeat=length):
39             guess = "".join(guess)
40             attempt += 1
41             print(f"Attempt {attempt}: Trying {guess}")
42             hash_object = hashlib.md5(guess.encode())
43             hashed_guess = hash_object.hexdigest()
44             if hashed_guess == hash_to_crack:
45                 end_time = time.time()
46                 elapsed_time = end_time - start_time
47                 print(f"Brute-force attack ha trovato la password: {guess} in {elapsed_time:.2e} secondi")
48                 return guess, elapsed_time
49     end_time = time.time()
50     elapsed_time = end_time - start_time
51     print(f"Brute-force attack non ha trovato la password in {elapsed_time:.2e} secondi")
52     return None, elapsed_time
53
54 # Funzione per craccare con rule-based attack
55 def rule_based_attack(hash_to_crack, wordlist):
56     start_time = time.time()
57     with open(wordlist, "r", encoding="utf-8", errors="ignore") as file:
58         for word in file:
59             word = word.strip()
60             # Prova la parola originale
61             if hashlib.md5(word.encode()).hexdigest() == hash_to_crack:
62                 end_time = time.time()
63                 elapsed_time = end_time - start_time
64                 print(f"Rule-based attack ha trovato la password: {word} in {elapsed_time:.2e} secondi")
65                 return word, elapsed_time
66             # Prova con alcune regole
67             for i in range(10):
68                 modified_word = word + str(i)
69                 if hashlib.md5(modified_word.encode()).hexdigest() == hash_to_crack:
70                     end_time = time.time()
71                     elapsed_time = end_time - start_time
72                     print(f"Rule-based attack ha trovato la password: {modified_word} in {elapsed_time:.2e} secondi")
73                     return modified_word, elapsed_time
74                 modified_word = str(i) + word
75                 if hashlib.md5(modified_word.encode()).hexdigest() == hash_to_crack:
76                     end_time = time.time()
77                     elapsed_time = end_time - start_time
78                     print(f"Rule-based attack ha trovato la password: {modified_word} in {elapsed_time:.2e} secondi")
79                     return modified_word, elapsed_time
80             # Aggiungi altre regole se necessario
81     end_time = time.time()
82     elapsed_time = end_time - start_time
83     print(f"Rule-based attack non ha trovato la password in {elapsed_time:.2e} secondi")
84     return None, elapsed_time
```

(a)

(b)

Il codice implementa tre metodi principali di cracking delle password: attacco a dizionario, attacco basato su regole e attacco di forza bruta. Di seguito vengono descritte le caratteristiche di ciascun metodo.

---

## Dictionary Attack

- **Descrizione:** Utilizza un file di wordlist contenente password comuni. Per ogni hash, il codice confronta l'hash MD5 di ogni password nella wordlist con l'hash obiettivo fino a trovare una corrispondenza.
- **Vantaggi:** Veloce per password comuni.
- **Svantaggi:** Inefficace per password complesse non presenti nella wordlist.

```
> cat cracked_passwords_dictionary.txt
Hash: 5f4dcc3b5aa765d61d8327deb882cf99
Metodo: Dictionary attack
Password: password
Tempo impiegato: 1.48e-04 secondi

Hash: e99a18c428cb38d5f260853678922e03
Metodo: Dictionary attack
Password: abc123
Tempo impiegato: 5.15e-05 secondi

Hash: 8d3533d75ae2c3966d7e0d4fcc69216b
Metodo: Dictionary attack
Password: charley
Tempo impiegato: 2.40e-03 secondi

Hash: 0d107d09f5bbe40cade3de5c71e9e9b7
Metodo: Dictionary attack
Password: letmein
Tempo impiegato: 3.04e-04 secondi

Hash: 5f4dcc3b5aa765d61d8327deb882cf99
Metodo: Dictionary attack
Password: password
Tempo impiegato: 2.26e-05 secondi
```

---

### Rule-Based Attack

- **Descrizione:** Applica regole definite dall'utente per generare password. Nel codice d'esempio, ogni carattere di una regola viene ripetuto 6 volte per formare una password (ad esempio, "aaaaaa").
- **Vantaggi:** Flessibile e personalizzabile.
- **Svantaggi:** Può essere lento se le regole sono complesse o numerose.

```
> cat cracked_passwords_rule_based.txt
Hash: 5f4dcc3b5aa765d61d8327deb882cf99
Metodo: Rule_based attack
Password: password
Tempo impiegato: 2.60e-04 secondi

Hash: e99a18c428cb38d5f260853678922e03
Metodo: Rule_based attack
Password: abc123
Tempo impiegato: 2.78e-04 secondi

Hash: 8d3533d75ae2c3966d7e0d4fcc69216b
Metodo: Rule_based attack
Password: charley
Tempo impiegato: 4.39e-02 secondi

Hash: 0d107d09f5bbe40cade3de5c71e9e9b7
Metodo: Rule_based attack
Password: letmein
Tempo impiegato: 6.61e-03 secondi

Hash: 5f4dcc3b5aa765d61d8327deb882cf99
Metodo: Rule_based attack
Password: password
Tempo impiegato: 7.77e-05 secondi
```

---

## Brute Force Attack

- **Descrizione:** Prova tutte le possibili combinazioni di caratteri fino a trovare la password corretta. Per motivi di tempo, gli hash utilizzati sono stati cambiati per rappresentare password più brevi e semplici.
- **Vantaggi:** Trova tutte le password, indipendentemente dalla complessità.
- **Svantaggi:** Estremamente lento per password lunghe e complesse.

```
> cat cracked_passwords_brute_force.txt
Hash: 49f68a5c8493ec2c0bf489821c21fc3b
Metodo: Brute_force attack
Password: hi
Tempo impiegato: 1.35e-03 secondi

Hash: 06d80eb0c50b49a509b49f2424e8c805
Metodo: Brute_force attack
Password: dog
Tempo impiegato: 2.09e-02 secondi

Hash: 48d6215903dff56238e52e8891380c8f
Metodo: Brute_force attack
Password: blue
Tempo impiegato: 1.82e-01 secondi

Hash: 1f3870be274f6c49b3e31a0c6728957f
Metodo: Brute_force attack
Password: apple
Tempo impiegato: 3.98e+00 secondi

Hash: 8621ffdbc5698829397d97767ac13db3
Metodo: Brute_force attack
Password: dragon
Tempo impiegato: 4.40e+02 secondi
```

---

## DIFFERENZE TRA HASHCAT, JOHN THE RIPPER E IL CODICE PYTHON

### 0.1 *Limitazioni*

- **Hashcat/John the Ripper:**
  - **Supporto Hardware:** Ottimizzati per l'uso di GPU, che permettono di velocizzare notevolmente il cracking delle password.
  - **Algoritmi Supportati:** Supportano una vasta gamma di algoritmi di hashing.
  - **Efficienza:** Più efficienti per attacchi su larga scala grazie all'ottimizzazione dell'uso delle risorse hardware.
- **Codice Python:**
  - **Supporto Hardware:** Limitato all'uso della CPU, rendendo il cracking delle password più lento rispetto all'uso delle GPU.
  - **Algoritmi Supportati:** Il codice d'esempio supporta solo l'hashing MD5.
  - **Efficienza:** Meno efficiente per attacchi su larga scala; adatto principalmente a dimostrazioni e piccoli test.

### 0.2 *Tempi di Cracking*

- **Hashcat/John the Ripper:** Grazie all'ottimizzazione per GPU, i tempi di cracking possono essere significativamente ridotti, specialmente per hash complessi o password lunghe.
- **Codice Python:**
  - **Dictionary Attack:** Veloce per password comuni presenti nella wordlist.
  - **Rule-Based Attack:** Il tempo dipende dalla complessità delle regole definite.
  - **Brute Force Attack:** Molto lento per password lunghe a causa della prova di tutte le possibili combinazioni.



## CONCLUSIONE

Il codice Python ad hoc utilizzato per il cracking delle password dimostra l'efficacia di diversi metodi di attacco, anche se con limitazioni rispetto a strumenti più avanzati come Hashcat e John the Ripper. L'operazione di cracking è stata completata con successo, rivelando le password in chiaro associate agli hash forniti. Questa esercitazione sottolinea l'importanza di utilizzare algoritmi di hashing più sicuri e l'uso di salting per proteggere le password memorizzate. Il salting aggiunge un valore casuale a ciascuna password prima di eseguire l'hashing, rendendo molto più difficile per gli attaccanti utilizzare tecniche di attacco pre-compilate come le Rainbow Tables. Questo esercizio dimostra come password deboli e algoritmi di hashing obsoleti possano essere facilmente compromessi. È cruciale per le organizzazioni adottare misure di sicurezza avanzate, come l'utilizzo di algoritmi di hashing più robusti (ad esempio, bcrypt, scrypt o Argon2) e politiche di gestione delle password più rigide per proteggere i dati sensibili.