

Mini-challenge: Password Cracking e Analisi della Protezione PDF e degli Hash di Sistema

Questa mini-challenge ha l'obiettivo di fornire un'esperienza pratica su due tecniche di attacco: il cracking delle password per file PDF protetti e l'analisi degli hash di sistema su ambienti Unix-based.

1. Obiettivo generale

L'obiettivo principale della mini-challenge è sviluppare e applicare strategie di attacco per recuperare o aggirare password che proteggono risorse. Si mira a:

- Verificare la robustezza delle password
- Comprendere le modalità di protezione
- Valutare l'efficacia delle tecniche di cracking

2. Brute force su file PDF

2.1 File PDF protetti

Sono stati forniti tre file PDF protetti da password, ognuno con una robustezza differente:

- **Weak:** Password debole
- **Medium:** Password di media robustezza
- **Strong:** Password forte

Ogni file PDF è stato protetto con due livelli di password:

- **User password:** necessaria per aprire il documento.
- **Owner password:** necessaria per modificarlo (es. copiare/incollare, stampare, ecc.).

2.2 Scopo operativo

- Accedere al contenuto dei file PDF senza la password originale, utilizzando tecniche di brute force.
- Verificare se è possibile bypassare le restrizioni imposte dall'owner password.
- Se necessario, creare una versione "libera" del file con contenuti accessibili.

2.3 Strumenti e tecniche utilizzati

- **John the Ripper**
- **Hashcat**
- **Rainbow tables**

Sono stati impiegati diversi tipi di attacco:

- Attacchi a **wordlist**
- **Attacco singolo**
- **Attacco combinato**

Inoltre, sono stati usati dizionari preesistenti, come "rockyou.txt", e sono stati creati dizionari personalizzati per i contesti specifici.

3. Hashing e cracking su sistemi operativi

3.1 Obiettivo

L'obiettivo della seconda fase è analizzare gli algoritmi di hashing utilizzati per gestire le password nei seguenti sistemi operativi Unix-based:

- **macOS**
- **Ubuntu**

Le attività principali sono:

- Identificare i file di sistema che contengono le password hashate (es. /etc/shadow su Linux, .plist su macOS).
- Determinare quale algoritmo di hashing viene utilizzato nei due ambienti (es. **SHA512Crypt** su Ubuntu, **PBKDF2-HMAC-SHA512** su macOS).
- Testare la sicurezza degli hash creando account di test con password deboli e tentare il cracking degli hash.

Questa mini-challenge combina tecniche di cracking su file PDF e la gestione degli hash di password in sistemi Unix, con l'obiettivo di sviluppare una comprensione pratica delle vulnerabilità legate alla gestione delle password.

4. Cosa è una funzione hash?

Una **funzione hash** è un algoritmo che trasforma un input di lunghezza variabile in una stringa di lunghezza fissa, detta **digest** o **valore hash**. Il processo è **unidirezionale**: è facile calcolare l'hash a partire dai dati originali, ma è difficile (idealmente impossibile) risalire ai dati originali dall'hash.

4.1 Proprietà principali di una funzione hash

Le funzioni hash presentano diverse caratteristiche fondamentali:

Determinismo

- Lo stesso input produce sempre lo stesso hash.

Lunghezza fissa dell'output

- Indipendentemente dalla lunghezza dell'input, l'hash risultante ha una lunghezza fissa (ad esempio, 256 bit per SHA-256).

Efficienza

- L'hash viene calcolato rapidamente, anche per input di grandi dimensioni.

Resistenza alle collisioni

- È computazionalmente difficile trovare due input distinti che producono lo stesso hash.

Resistenza alla preimmagine

- Dato un hash, è difficile trovare un input che lo generi.

Resistenza alla seconda preimmagine

- Dato un input e il suo hash, è difficile trovare un altro input che produca lo stesso hash.

4.2 Applicazioni delle funzioni hash

Le funzioni hash vengono utilizzate in vari contesti, tra cui:

Verifica dell'integrità dei dati

- Per garantire che i dati non siano stati alterati.

Archiviazione sicura delle password

- Le password vengono memorizzate come hash, rendendo difficile il recupero in caso di violazione dei dati.

Firme digitali

- Per garantire l'autenticità e l'integrità di un messaggio o documento.

Strutture dati come hash table

- Per una ricerca e un accesso rapido ai dati.

Blockchain e criptovalute

- Per garantire la sicurezza delle transazioni e collegare i blocchi.

4.3 Esempio pratico

Utilizzando l'algoritmo **SHA-256**, l'hash della parola "p4ssw0rd" è:

5450c59b739affeff1a06dbba82c2e77a5dce5a548a0522e0f014aa625bc054

Anche una piccola modifica nell'input, come cambiare una lettera, produrrà un hash completamente diverso.

5. Protezione e cracking dei file PDF versione 1.4

5.1 Protezione dei file PDF nella versione 1.4

La protezione tramite password nei file PDF esiste dalla versione 1.1, ma la versione 1.4, rilasciata nel 2001, ha introdotto la cifratura **RC4 a 128 bit**, migliorando la sicurezza rispetto ai 40 bit precedenti. I file PDF possono essere protetti da due password:

- **User password**: necessaria per aprire il documento.
- **Owner password**: consente di impostare restrizioni su stampa, copia e modifica.

La chiave di cifratura è derivata attraverso un processo crittografico che utilizza le password, i permessi, l'ID del file e un padding fisso. Viene utilizzato l'algoritmo **MD5** per generare l'hash da cui verrà derivata la chiave **RC4** per cifrare il PDF.

5.2 Processo di decifrazione

Per decifrare un PDF protetto (versione 1.4), è necessario conoscere la **user password**, che è richiesta per aprire il file. Se la password è lunga e complessa, può risultare difficile individuarla. Tuttavia, strumenti di cracking possono aggirare questa protezione con attacchi di vario tipo:

- **John the Ripper**: consente attacchi a dizionario e brute-force sugli hash delle password.
- **Hashcat**: utilizza la GPU per attacchi ad alte prestazioni, particolarmente efficaci per hash complessi.
- **Rainbow tables**: efficaci per password corte e semplici, ma richiedono risorse significative per essere generate.

5.3 Processo di attacco

Il cracking inizia con l'estrazione dell'hash della password dal file PDF, utilizzando uno script come **pdf2john.pl** incluso in John the Ripper. Una volta ottenuto l'hash, si può avviare l'attacco con l'approccio più adatto (dizionario, brute-force, rainbow table). Se la user password viene trovata, è possibile:

- Decifrare completamente il documento.
- Rimuovere la cifratura con strumenti come **qpdf**.
- In alcuni casi, **qpdf** può rimuovere anche le restrizioni imposte dall'owner password (ad esempio, impedire copia/incolla o stampa), senza necessitare della conoscenza di quest'ultima, purché la user password sia valida.

6. Estrazione degli hash dai file PDF

6.1 Procedura

Per estrarre gli hash dalle password PDF è necessario operare nella directory john/run ed eseguire i comandi su ciascun file PDF protetto.

```
cd john/run
```

```
pdfprotected_weak 2.pdf
```

```
MacBook-Air-di-Emanuel@~/john/run % ./pdf2john.pl "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/pdfprotected_weak 2.pdf" > "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_weak.txt"
```

```
pdfprotected_medium 2.pdf
```

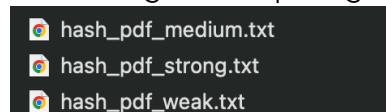
```
MacBook-Air-di-Emanuel@~/john/run % ./pdf2john.pl "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/pdfprotected_medium 2.pdf" > "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_medium.txt"
```

```
pdfprotected_strong 2.pdf
```

```
MacBook-Air-di-Emanuel-emmanuel@~/john/run % ./pdf2john.pl "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/pdfprotected_strong 2.pdf" > "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_strong.txt"
```

6.2 File generati

Verranno creati file **.txt** contenenti le stringhe hash per ogni PDF:



6.3 Contenuto dei file .txt e struttura dell'hash

pdfprotected_weak 2.pdf:

pdfprotected_medium 2.pdf:

pdfprotected_strong 2.pdf:

6.4 Struttura dell'hash PDF

Ogni hash ha la seguente struttura:

\$pdf\$<version>*<revision>*<keybits>*<P>*<enc_metadata>*<len_u>*<u>*<len_o>*<o>*<len_id>*<id>

Dettaglio della stringa

| Campo | Valore | Significato |
|---------|--------------------------|---|
| \$pdf\$ | - | Identifica il formato PDF |
| 2 | Versione 2 di encryption | PDF v1.4 |
| 3 | Revision | Usa algoritmo RC4 e user password hashata con MD5 |

| | | | |
|----------------------------------|--|------------------|---|
| 128 | | Key Length | 128 bit di chiave |
| -3904 | | Permissions (P) | Flag di permessi (in formato signed int) |
| 1 | | Encrypt Metadata | 1 = metadati cifrati |
| 16 | | Lunghezza ID | Lunghezza dell'ID del documento |
| d7b11ca047eb6171e6e8fe8a64b04cfa | | Document ID | ID univoco del documento PDF |
| 32 | | Lunghezza U | Lunghezza del campo user password cifrata |
| 820970...00000000 | | User Password | Encrypted user password (hash di "user password") |
| 32 | | Lunghezza O | Lunghezza della owner password cifrata |
| b21927...e374 | | Owner Password | Owner password hashata e completata (padding) |

6.4.1 Valore dei permessi

| Valore | Significato |
|---------------------|------------------------------------|
| -3904 | Valore decimale signed |
| FFFFF140 | Rappresentazione 32-bit signed hex |
| 40F1FFFF | Versione little-endian (PDF style) |
| b'\x40\xf1\xff\xff' | In byte nel file binario |

6.4.2 Significato dei bit di permesso

| Bit # | Valore binario | Significato |
|-------|----------------|--|
| 3 | 1 | Stampare il documento |
| 4 | 1 | Modificare il contenuto |
| 5 | 1 | Copiare o estrarre contenuti |
| 6 | 1 | Aggiungere/modificare annotazioni |
| 9 | 1 | Compilare moduli |
| 10 | 1 | Estrarre per accessibilità |
| 11 | 1 | Assemblare il documento |
| 12 | 0 | Stampare in alta qualità (DISABILITATO) |

I permessi sono codificati in bit all'interno del valore P, secondo lo standard PDF. Un valore negativo (es. -3904) indica che alcune funzionalità sono disattivate.

7. Strategie di attacco per recuperare la password

7.1 Brute-force attack

- Prova sistematica di tutte le combinazioni di caratteri.
- Efficace solo per password **brevi o semplici**.
- **Molto lento** su password complesse.

7.2 Dictionary attack

- Utilizza una lista di parole comuni o già trapelate.
- Più **veloce ed efficiente** rispetto al brute-force.
- Efficace contro password **deboli o prevedibili**.

7.3 Rainbow table attack

- Usa tabelle precomputate che associano hash a password.
- Molto veloce **se l'hash è noto e non salato**.
- Richiede molta memoria e non è efficace se è presente il **salting**.

8. Tool utilizzati: John the Ripper vs Hashcat

8.1 John the Ripper

- Tool **versatile** e adatto a una vasta gamma di formati hash.
- Supporta dictionary e brute-force.
- Ideale per hash **meno complessi** o per file di sistema (es. /etc/shadow).
- **Facile da usare**, adatto a utenti meno esperti.

8.2 Hashcat

- Tool ad alte prestazioni per attacchi basati su GPU.
- Estremamente **veloce e scalabile**, ottimo per hash **robusti** (es. bcrypt, yescript).
- Indicato per **attacchi intensivi** su password complesse.
- Richiede hardware potente e maggiore configurazione.

9. John the Ripper

9.1 Descrizione generale

John the Ripper è uno strumento open source per il cracking delle password, utile su:

- Sistemi Unix (es. /etc/shadow)
- File cifrati
- Archivi protetti

Supporta molti algoritmi di hashing e combina semplicità con potenza operativa.

9.2 Comandi base

john [opzioni] [file_hash]

Principali Opzioni:

- --wordlist=FILE: attacco con dizionario
- --incremental: attacco brute-force
- --format=FORMAT: specifica il tipo di hash (es. pdf)
- --show: mostra le password trovate
- --test: test delle performance

9.3 Utilizzo su PDF

9.3.1 PDF con password weak

john --format=pdf hash_pdf_weak.txt

john --show hash_pdf_weak.txt

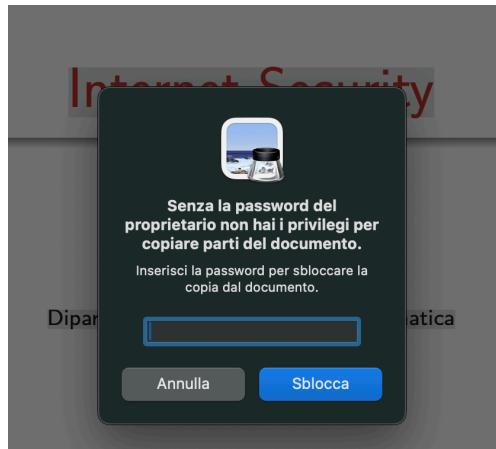
```
[MacBook-Air-di-Emanuel-emmanuel@~ % john --format=pdf "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_weak.txt"
Using default input encoding: UTF-8
Loaded 1 password hash (PDF [MD5 SHA2 RC4/AES 32/64])
No password hashes left to crack (see FAQ)
[MacBook-Air-di-Emanuel-emmanuel@~ % john --show "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_weak.txt"
/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/pdfprotected_weak 2.pdf:bene
1 password hash cracked, 0 left
```

- Password trovata: "**bene**"
- Owner password **non trovata**
- Accesso consentito ma **con limitazioni (no copia/modifica)**

9.3.2 PDF con password medium

```
john --format=pdf --incremental=ASCII hash_pdf_medium.txt  
john --show hash_pdf_medium.txt
```

- Password trovata: "**cod67a**"
- Owner password **non trovata**
- Accesso consentito ma **limitazioni attive**



Rimozione restrizioni con QPDF:

Weak

```
qpdf --password=bene --decrypt protected_weak.pdf  
output_unlocked_weak.pdf
```

Medium

```
qpdf --password=cod67a --decrypt protected_medium.pdf  
output_unlocked_medium.pdf
```

9.3.3 PDF con password strong

```
john --format=pdf --incremental hash_pdf_strong.txt  
• Nessun risultato dopo lungo tempo → si passa ad Hashcat
```

10. PDFCrack

10.1 Scopo

Recuperare **owner password** per rimuovere completamente le restrizioni del PDF.

10.2 Caso weak

```
pdfcrack -f pdf_weak.pdf --owner
```

- Owner password trovata: "**ciao**"
 - Password molto debole → recuperata in pochi secondi

10.3 Caso medium

```
pdfcrack -f pdf_medium.pdf -p cod67a -c  
abcdefghijklmnopqrstuvwxyz0123456789 -n 6 --owner
```

- Owner password trovata: "**br34io**"
 - Recuperata dopo circa un'ora

11. Hashcat

11.1 Caratteristiche

- Tool GPU-based per il cracking veloce
 - Supporta una vasta gamma di algoritmi di hash moderni

11.2 Opzioni principali

- -m <tipo>: hash mode (es. -m 10500 per PDF v1.4)
 - -a <modalità>: tipo di attacco (es. -a 3 brute-force)
 - --increment: aumenta la lunghezza delle password testate
 - -o: salva l'output

- --force: forza esecuzione anche su CPU

11.3 Modalità di attacco

| Attack-Mode | Hash-Type | Example command |
|------------------|-----------|---|
| Wordlist | \$P\$ | hashcat -a 0 -m 400 example400.hash example.dict |
| Wordlist + Rules | MD5 | hashcat -a 0 -m 0 example0.hash example.dict -r rules/best64.rule |
| Brute-Force | MD5 | hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a |
| Combinator | MD5 | hashcat -a 1 -m 0 example0.hash example.dict example.dict |
| Association | \$1\$ | hashcat -a 9 -m 500 example500.hash 1word.dict -r rules/best64.rule |

11.4 Maschere e set di caratteri

Uso del placeholder ? per definire set di caratteri:

- ?l = lettere minuscole
- ?u = lettere maiuscole
- ?d = cifre
- ?s = simboli

| ? Charset |
|--------------------------------------|
| l abcdefghijklmnopqrstuvwxyz [a-z] |
| u ABCDEFGHIJKLMNOPQRSTUVWXYZ [A-Z] |
| d 0123456789 [0-9] |
| h 0123456789abcdef [0-9a-f] |
| H 0123456789ABCDEF [0-9A-F] |
| s !#\$%&'()*+,-./:;<=>?@[\]^_`{ }~ |
| a ?l?u?d?s |
| b 0x00 - 0xff |

Esempi:

- ?l?l?l?l?: 4 lettere minuscole
- -1 ?l?d e maschera ?1?1?1?1?1?1: 6 caratteri da lettere e numeri

11.5 Attacchi con Hashcat

11.5.1 Caso weak

```
MacBook-Air-di-Emanuel-emmanuel@~ % hashcat -m 10500 "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/hash_pdf_weak.txt"
"/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/rockyou.txt" -O
```

hashcat -m 10500 hash_pdf_weak.txt rockyou.txt -O

- Password trovata in **~10 secondi**

11.5.2 Caso medium

1. Tentativo con dizionario: **fallito**
 2. Attacco con maschera:

```
[MacBook-Air-d1-Emanuel-emmanuel@ ~ % hashcat -m 10500 -a 3 -i '?!?d' "/Users/emanuel/Desktop/UniCT/Notes/Internet Security/Mini-Challenge Ramaci Emanuel/] hash_pdf_medium.txt" '?!?1?1?1?1?1?1?1'
```

```
hashcat -m 10500 -a 3 -1 ?l?d hash_pdf_medium.txt ?1?1?1?1?1?1
```

- Password trovata in **~4 minuti**

11.5.3 Caso strong

```
hashcat -m 10500 -a 3 -1 ?l?u?d?s hash_pdf_strong.txt ?1?1?1?1?1?1?1?1
```

- Nessun risultato utile
 - Tempo stimato: **79 anni e 95 giorni**

In caso di password complesse, la sicurezza del documento aumenta esponenzialmente: la scelta di una password robusta resta quindi la difesa più efficace.

12. Introduzione alle rules

12.1 Cos'è una regola

Le *rules* (regole) servono per trasformare le parole di una wordlist in varianti più complesse:

- Simulano comportamenti reali delle persone (es. aggiunta di numeri, maiuscole, simboli)
 - Aumentano esponenzialmente le possibilità di successo

- Evitano la necessità di creare manualmente ogni possibile variante

12.2 Funzionamento in Hashcat

12.2.1 Modalità di attacco

Le regole si applicano **solo in modalità -a 0** (dictionary attack).

Non si usano con **-a 3** (mask attack), poiché le combinazioni sono già definite dall'utente.

12.2.2 Esempio di comando

```
hashcat -a 0 -m 10500 hash.txt wordlist.txt -r rules/best64.rule
```

- -a 0: dictionary attack
- -m 10500: hash di PDF v1.4
- -r: applica regole definite nel file best64.rule

12.2.3 Esempio pratico

Wordlist:

password

Regola:

\$1 (append "1")

Varianti testate:

password

password1

12.3 Funzionamento in John the Ripper

12.3.1 Comando base con regole

```
john --format=pdf --wordlist=rockyou.txt hash.txt
```

- Applica automaticamente le regole nella sezione [List.Rules:Wordlist] di john.conf

12.3.2 Uso di regole personalizzate

```
john --format=pdf --wordlist=rockyou.txt --rules=single hash.txt
```

- Specifica la sezione List.Rules:single nel file di configurazione

12.3.3 Dove trovare/modificare le regole

File di configurazione:

- Linux/macOS: /usr/share/john/john.conf o ~/.john/john.conf
- Windows: john.ini

Le regole si trovano in sezioni come:

```
[List.Rules:Wordlist]
c      # Capitalize prima lettera
$[0-9]  # Append cifra da 0 a 9
```

12.3.4 Esempio pratico

Wordlist:

password

Regole:

c → capitalize prima lettera

\$1 → append "1"

Varianti generate:

password

Password

password1

Password1

12.4 Differenze tra regole e maschere

| Metodo | Caratteristiche | Applicazione |
|-----------------|---|----------------------|
| Rules | Trasformano parole da una wordlist | Solo con -a 0 |
| Maschere | Specificano lo schema esatto della password | Solo con -a 3 |

Infine, si è deciso di esplorare l'utilizzo delle **rainbow tables**, non con l'obiettivo di recuperare la password strong, la cui complessità renderebbe l'approccio impraticabile in termini di tempo e spazio, ma piuttosto per **comprendere il funzionamento del metodo** e verificarne l'efficacia in un contesto controllato.

13. Introduzione alle rainbow tables

13.1 Cos'è una rainbow table

- È una **struttura precomputata** usata per invertire hash crittografici.
- Permette di **risalire alla password** a partire dal suo hash.
- Si basa sul compromesso **tempo vs spazio**:
 - Richiede **tempo iniziale** per la generazione.
 - Permette **attacchi molto rapidi** una volta pronta.

13.2 Funzionamento

13.2.1 Creazione delle tabelle

- Si generano molte password e si calcolano i relativi hash.
- Si applica una **funzione di riduzione** all'hash per ottenere una nuova password.
- Si crea una **catena di trasformazioni**: password → hash → riduzione → hash ...
- Si salvano solo l'**inizio e la fine** di ogni catena → risparmio di spazio.

13.2.2 Attacco con rainbow table

- Si cerca l'hash tra le **code delle catene**.
- Se trovato, si **ricostruisce la catena** dall'inizio per recuperare la password originale.

13.3 Quando utilizzarle

13.3.1 Casi favoriti

- Algoritmi **veloci e deboli**: es. MD5, SHA1, NTLM.
- Situazioni **senza salt**.

13.3.2 Limiti

- **Inutili con hash salati**: un salt cambia l'hash anche per password identiche.
- **Spazio disco elevato**: anche centinaia di GB.
- **Inefficaci con algoritmi lenti**: es. bcrypt, scrypt, yescript.

13.4 Uso di RainbowCrack su Ubuntu

13.4.1 Cos'è RainbowCrack

- Tool open source per **attacchi con rainbow tables**.
- A differenza di Hashcat/John the Ripper, **non calcola hash al volo**.
- Funziona solo con algoritmi **veloci e non salati**.

13.4.2 Componenti principali

- rtgen: genera le rainbow tables.
- rtsort: ordina le tabelle per ottimizzare le ricerche.
- rcrack: utilizza le tabelle per recuperare la password da un hash.

13.4.3 Esecuzione passo-passo

13.4.3.1 Generazione della tabella

```
emanuel@ubuntu:~/RainbowCrack-NG/src$ ./rtgen md5 numeric 4 4 0 10000 10000 0
hash routine: md5
hash length: 16
plain charset: 0123456789
plain charset in hex: 30 31 32 33 34 35 36 37 38 39
plain length range: 4 - 4
plain charset name: numeric
plain space total: 10000
rainbow table index: 0
reduce offset: 0

generating...
10000 of 10000 rainbow chains generated (0 m 14 s)
```

rtgen md5 loweralpha-numeric 4 4 0 10000 10000 0

Significato dei parametri:

- md5: algoritmo di hash
- loweralpha-numeric: charset (a-z, 0-9)
- 4 4: lunghezza minima e massima delle password
- 0: indice della tabella (per differenziare tabelle simili)
- 10000: lunghezza di ciascuna catena (iterazioni)
- 10000: numero di catene

- 0: parte dell'indice del file di output

13.4.3.2 Ordinamento delle tabelle

```
emanuel@ubuntu:~/RainbowCrack-NG/src$ ./rtsort md5_numeric#4-4_0_10000x10000_0.rt
available physical memory: 2854412288 bytes
loading rainbow table...
sorting rainbow table...
writing sorted rainbow table...
```

rtsort .

- Ordina le tabelle nella directory corrente.
- Aumenta notevolmente la velocità di ricerca.

13.4.3.3 Cracking dell'hash

```
emanuel@ubuntu:~/RainbowCrack-NG/src$ echo -n 1111 | md5sum
b59c67bf196a4758191e42f76670ceba -
emanuel@ubuntu:~/RainbowCrack-NG/src$ ./rccrack md5_numeric#4-4_0_10000x10000_0.rt -h b59c67bf196a4758191e42f76670ceba
```

rccrack . -h <hash_da_craccare>

- Cerca l'hash nelle tabelle ordinate nella directory.
- Se trovato, restituisce la password in chiaro. Nel nostro caso:

```
1 rainbow tables found
memory available: 1819521843 bytes
memory for rainbow chain traverse: 160000 bytes per hash, 160000 bytes for 1 hashes
memory for rainbow table buffer: 2 x 160016 bytes
disk: ./md5_numeric#4-4_0_10000x10000_0.rt: 160000 bytes read
disk: finished reading all files
plaintext of b59c67bf196a4758191e42f76670ceba is 1111

statistics
-----
plaintext found: 1 of 1
total time: 4.27 s
time of chain traverse: 4.26 s
time of alarm check: 0.00 s
time of disk read: 0.00 s
hash & reduce calculation of chain traverse: 49990000
hash & reduce calculation of alarm check: 2212
number of alarm: 2212
performance of chain traverse: 11.72 million/s
performance of alarm check: 2.21 million/s

result
-----
b59c67bf196a4758191e42f76670ceba  1111  hex:31313131
```

Tuttavia, RainbowCrack non è specificamente progettato per gestire gli hash utilizzati nei file PDF protetti, come quelli delle versioni 1.4 o successive. Gli hash dei PDF spesso impiegano algoritmi di cifratura come RC4 o AES, combinati con salting e iterazioni multiple, rendendoli incompatibili con le rainbow tables predefinite di RainbowCrack.

Nonostante i tentativi effettuati con diverse tecniche, la complessità e la lunghezza della password strong rendono **impraticabile** il cracking nei **tempi ragionevoli**. Di conseguenza, non è stato possibile recuperare la password strong.

14. Cracking delle password su Ubuntu

14.1 Cracking della password dell'utente Emanuel (SHA512crypt)

14.1.1 Strumento utilizzato

- **John the Ripper**
- Algoritmo: sha512crypt

14.1.2 Procedura

- Estrazione dell'hash dal file /etc/shadow.
- Utilizzo di una wordlist per trovare la password corrispondente.

```
root@ubuntu:/john-jumbo/run# sudo grep '^emanuel:' /etc/shadow > emanuel_shadow.txt
root@ubuntu:/john-jumbo/run# ./john --format=sha512crypt emanuel_shadow.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 ASIMD 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Note: Passwords longer than 26 [worst case UTF-8] to 79 [ASCII] rejected
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
0g 0:00:00:01 DONE 1/3 (2025-04-13 09:57) 0g/s 2837p/s 2837c/s 2837C/s Emanuel999991900..E999991900
Proceeding with wordlist:./password.lst
Enabling duplicate candidate password suppressor using 256 MiB
easy          (emanuel)
1g 0:00:00:06 DONE 2/3 (2025-04-13 09:57) 0.1631g/s 3423p/s 3423c/s 3423C/s baby24..Stephen
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

14.2 Cracking della password dell'utente Justin (yescript)

14.2.1 Algoritmo utilizzato

- **yescript**, un algoritmo moderno e sicuro, supportato dalle ultime versioni di John e Hashcat.

```
root@ubuntu:/john-jumbo/run# grep '^justin:' /etc/shadow > justin_shadow.txt
root@ubuntu:/john-jumbo/run# ./john --format=crypt justin_shadow.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cracked 1 password hash (is in ./john.pot), use "--show"
No password hashes left to crack (see FAQ)

root@ubuntu:/john-jumbo/run# ./john --show justin_shadow.txt
justin:hellolkitty:20190:0:99999:7:::

1 password hash cracked, 0 left
```

14.3 Cracking con Hashcat

14.3.1 Comando base

```
hashcat -m 1800 -a 0 -o cracked.txt shadow rockyou.txt
```

14.3.2 Spiegazione dei parametri

- -m 1800: Tipo di hash = SHA512crypt
- -a 0: Modalità dizionario (wordlist attack)
- -o cracked.txt: File di output con le password trovate
- shadow: File contenente gli hash (ottenuto con sudo cat /etc/shadow > shadow)
- rockyou.txt: Wordlist usata per il cracking

15. Cracking delle password su macOS

15.1 Accesso e estrazione dell'hash

15.1.1 Requisiti

- Accesso **root** al sistema.
- I file degli utenti sono in: /var/db/dslocal/nodes/Default/users/

```
[MacBook-Air-di-Emanuel-emmanuel@~ % sudo cat /var/db/dslocal/nodes/Default/users/winnie.plist
```

- I dati sono contenuti nei file .plist (Property List), in formato **binario/Base64**.

15.1.2 Estrazione dell'hash

- Utilizzo di uno script Python: extract_hash_bplist.py



```
import plistlib
import sys
import biplist # pip install biplist

def extract_hash(plist_path):
    with open(plist_path, 'rb') as f:
        user_plist = plistlib.load(f)

    shadow_data = user_plist['ShadowHashData'][0] # È un elemento bplist serializzato
    blob = biplist.readPlistFromString(shadow_data)

    salted = blob['SALTED-SHA512-PBKDF2']
    salt = salted['salt']
    iterations = salted['iterations']
    entropy = salted['entropy']

    with open("winnie_hash.txt", "w") as f:
        # Hashcat format: $m$iterations$salt$entropy
        f.write(f"${iterations}${salt.hex()}${entropy.hex()}\n")

    print("Hash estratto in formato compatibile Hashcat (winnie_hash.txt)")

extract_hash(sys.argv[1])
```

Ln: 1 Col: 0

- L'hash si trova nella chiave ShadowHashData
- Il formato estratto: **PBKDF2-HMAC-SHA512**

```
sudo python3 extract_hash_bplist.py /var/db/dslocal/nodes/Default/users/winnie.plist
```

15.2 Dettagli sul metodo PBKDF2-HMAC-SHA512

15.2.1 Funzionamento

- Aggiunta di un **salt**
- Applicazione di PBKDF2 con **molte iterazioni** (es. 100.000+)
- Hash derivato tramite HMAC con **SHA512**
- Vengono salvati: salt, numero di iterazioni, e l'hash

15.2.2 Sicurezza

- Resistente ad attacchi brute-force e rainbow tables
- Algoritmo lento per design, rallenta i tentativi di cracking

15.3 Cracking con Hashcat su macOS

15.3.1 Comando utilizzato

```
[hashcat -m 7100 winnie_hash.txt rockyou.txt]
```

15.3.2 Spiegazione dei parametri

- -m 7100: PBKDF2-HMAC-SHA512 (formato macOS)
- winnie_hash.txt: File contenente l'hash dell'utente
- rockyou.txt: Dizionario usato per il cracking

15.3.3 Risultato

- Hashcat ha trovato la password dell'utente "winnie"
- Per visualizzarla: usare il file di output generato da hashcat

```
[hashcat --show winnie_hash.txt]
```

Conclusioni

Durante questa analisi, abbiamo esaminato diverse tecniche di cracking delle password, concentrandoci su documenti PDF e sistemi operativi come Ubuntu e macOS.

PDF protetti da password: I documenti PDF utilizzano algoritmi di cifratura come RC4, spesso combinati con salting e iterazioni multiple, rendendo il cracking delle password particolarmente complesso. Le password classificate come "weak" e "medium" sono state facilmente individuate utilizzando strumenti come John the Ripper e Hashcat, sfruttando attacchi basati su brute-force e dizionari. Tuttavia, per la password "strong", la complessità elevata ha reso impraticabile il cracking in tempi ragionevoli.

Cracking su Ubuntu: Utilizzando strumenti come John the Ripper e Hashcat, è stato possibile recuperare le password degli utenti Emanuel e Justin, rispettivamente protette con gli algoritmi sha512crypt e yescript. Questi strumenti sfruttano wordlist e regole per generare varianti delle password, aumentando le probabilità di successo.

Cracking su macOS: L'estrazione degli hash delle password da file .plist ha permesso di utilizzare Hashcat con il modulo PBKDF2-HMAC-SHA512 per recuperare la password dell'utente "winnie". Questo processo ha evidenziato l'importanza di comprendere i metodi di hashing e le tecniche di derivazione delle chiavi utilizzate nei sistemi moderni.

In sintesi, mentre alcune password possono essere recuperate con strumenti e tecniche adeguate, la complessità e le misure di sicurezza avanzate implementate in molti sistemi rendono il cracking delle password una sfida significativa. È fondamentale comprendere

le limitazioni e le potenzialità degli strumenti disponibili per valutare la fattibilità di tali operazioni.