

PASSWORD CRACKING



Autore: Bertolami Carmelo
Data: 15/04/2024
Corso: Internet Security

Contents

1	Introduzione:	2
2	Obiettivi	2
2.1	Teoria hashing	2
3	John the ripper	3
3.1	Tipi di attacco	3
4	Considerazioni preliminari LINUX	4
4.1	Vantaggi di yescrypt:	4
5	Attacco su Linux	5
5.1	Attacco a Dizionario	5
5.2	Forza Bruta	6
5.3	Bonus: Attacco a maschera	6
5.4	Attacco rainbow table	7
5.4.1	Prova Linux	7
6	Attacco su windows	7
6.0.1	Tool utilizzato per windows	7
6.1	Attacco a dizionario	8
6.2	Attacco forza bruta:	8
6.3	Attacco rainbow table	9
7	Tradeoff complessità-prestazione	10
7.1	Criticità	12
8	Conclusioni	13

1 Introduzione:

La mini-challenge presa in carico ha come obiettivo recuperare gli hash delle password associate di sistema associate ad un account utente, (avendo accesso ai file dove gli hash sono memorizzati in forma non intelligibile). Una volta ottenuti gli hash, verranno utilizzati una serie di metodi e strumenti, che saranno presentati di seguito, per risalire quindi alle password corrispondenti degli utenti presi in questione.

2 Obiettivi

Il completamento di questo mini-challenge necessita una panoramica pratica degli strumenti come John the Ripper e Hashcat, e sui vari metodi utilizzati nel cracking delle password sia su sistemi Linux che Windows. Questi obiettivi permettono di acquisire comprendere quelle che sono le tecniche di attacco più comuni, tra cui l'attacco a dizionario, l'attacco a forza bruta e l'utilizzo di tabelle rainbow. Inoltre, uno degli ultimi è quello di comprendere il tradeoff tra la complessità degli algoritmi di hash e le prestazioni, evidenziando come una scelta accurata di algoritmi influenzi la sicurezza complessiva di un sistema. Infine, si approfondirà l'algoritmo Argon2, un algoritmo di hash di password moderno e ampiamente considerato uno dei più sicuri al momento.

1. Tool utilizzati:

- John the ripper
- Hashcat

2. Cracking password Linux tramite:

- Attacco dizionario
- Attacco Forza bruta
- Attacco rainbow table

3. Cracking password Windows tramite:

- Attacco dizionario
- Attacco Forza bruta
- Attacco rainbow table

4. Calcolo tradeoff complessità-prestazioni degli algoritmi di hash

5. Studio empirico argon2

2.1 Teoria hashing

L' **hash** è una funzione che produce una sequenza di bit (o una stringa), detta digest, strettamente correlata con i dati in ingresso. Nel linguaggio matematico e informatico, l'hash è una funzione **non invertibile** che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita. Nelle applicazioni crittografiche si chiede, per esempio, che la funzione hash abbia le seguenti proprietà:

- **resistenza alla preimmagine:** sia computazionalmente intrattabile la ricerca di una stringa in input che dia un hash uguale a un dato hash;
- **resistenza alla seconda preimmagine:** sia computazionalmente intrattabile la ricerca di una stringa in input che dia un hash uguale a quello di una data stringa;
- **resistenza alle collisioni:** sia computazionalmente intrattabile la ricerca di una coppia di stringhe in input che diano lo stesso hash.

In alcune funzioni di Hash viene inserito il procedimento di salting (sale) consiste nell'aggiunta di una stringa aggiuntiva alla password.

Questa stringa può essere generata in modo pseudocasuale prima di calcolarne l'hash, quando un utente imposta una password, il sistema crea un valore casuale aggiuntivo, il salt. Questo valore si aggiunge in coda alla password e da tale stringa composta viene poi calcolato l'hash, che avrà un altro valore, differente da quello generato dalla sola password. In questo modo si rendono molto più onerosi gli attacchi "a dizionario" e quelli "brute force". Lo scopo del salt è infatti quello di moltiplicare il numero di combinazioni hash possibili a fronte di ciascuna password.

Per una data password (che se priva del salt avrebbe un unico hash come risultato), il numero di possibili combinazioni diventa invece pari a: 2^{sLen} dove sLen è la lunghezza del salt in bit.

3 John the ripper

John the Ripper: Modalità di attacco

Il tool utilizzato: John the Ripper è un software libero per la decrittazione forzata delle password. Inizialmente sviluppato per sistemi operativi UNIX, dal 2012 può essere eseguito su 15 differenti piattaforme (DOS, Microsoft Windows, BeOS e OpenVMS).

Una caratteristica interessante di John The Ripper è che il software può rilevare automaticamente l'algoritmo crittografico utilizzato per generare gli hash di ciascuna password. Ma tramite comandi è anche possibile specificarlo e semplificarli il lavoro.

3.1 Tipi di attacco

Attacco con dizionario:

In questa modalità, John the Ripper utilizza un file di parole ("dizionario") per tentare di indovinare la password. Il programma prova ogni parola nel dizionario contro l'hash della password e si ferma se trova una corrispondenza. I dizionari possono contenere parole comuni, nomi, cognomi, parole di diverso tipo e combinazioni di caratteri. L'efficacia di questo metodo dipende dalla qualità del dizionario utilizzato e dalla complessità della password.

Attacco brute force:

Questo metodo è più brutale rispetto all'attacco con dizionario. John the Ripper prova tutte le combinazioni possibili di caratteri fino a trovare quella corretta. L'attacco brute force può richiedere molto tempo, in base alla lunghezza della password e alla potenza di calcolo disponibile. Tuttavia, è garantito che troverà sempre la password se viene eseguito per un tempo sufficientemente lungo.

Attacco ibrido:

Questa modalità combina l'attacco con dizionario e l'attacco brute force. John the Ripper inizia con il provare le parole nel dizionario e poi passa alle combinazioni brute force se non trova una corrispondenza. L'attacco ibrido offre un buon compromesso tra velocità ed efficacia.

Attacco basato su regole:

In questa modalità, John the Ripper utilizza un insieme di regole per generare potenziali password da testare. Le regole possono essere basate su informazioni come il nome utente, la data di nascita o altri dati comuni. L'attacco basato su regole può essere più efficace di un semplice attacco con dizionario se si hanno alcune informazioni sulla password.

Attacco basato su maschera:

Questo metodo consente di specificare una maschera che definisce la struttura della password. Ad esempio, una maschera comune è "?l?l?l?l?", dove ogni "?" rappresenta un carattere. John the Ripper genera tutte le password possibili che corrispondono alla maschera e le testa contro l'hash. L'attacco basato su maschera può essere utile se si conosce la lunghezza della password e alcuni dei suoi caratteri.

Oltre a queste modalità di funzionamento principali, John the Ripper offre anche diverse altre opzioni e funzionalità, tra cui:

- Supporto per diversi tipi di hash di password
- La possibilità di salvare e riprendere gli attacchi
- La possibilità di generare report dettagliati sui risultati degli attacchi

Hashcat: Caratteristiche e Modalità di Funzionamento

Hashcat è un programma per il cracking delle password che, come John the Ripper, offre diverse modalità di funzionamento, ma si distingue per alcune caratteristiche uniche.

Modalità di funzionamento simili:

Hashcat offre modalità di funzionamento simili a John the Ripper, tra cui:

- Attacco con dizionario: Utilizza un dizionario di parole per provare a indovinare la password.
- Attacco brute force: Prova tutte le combinazioni possibili di caratteri fino a trovare la password corretta.
- Attacco ibrido: Combina l'attacco con dizionario e brute force per un compromesso tra velocità ed efficacia.
- Attacco basato su regole: Usa regole per generare potenziali password da testare basate su informazioni come nome utente o data di nascita.
- Attacco basato su maschera: Genera password che corrispondono a una maschera specificata (ad esempio, "?l?l?l?l?" per password di 6 lettere).
- Attacco da file: Utilizza un file contenente un elenco di potenziali password da testare.

Vantaggi di Hashcat:

- Sfruttamento della GPU: Hashcat è ottimizzato per l'utilizzo delle GPU, che offrono una potenza di calcolo significativamente superiore rispetto alle CPU per il cracking delle password.
- Supporto per diversi hash: Hashcat supporta un'ampia gamma di algoritmi di hash, inclusi quelli moderni e complessi.
- Alta velocità: Grazie all'utilizzo della GPU e ad algoritmi ottimizzati, Hashcat è uno dei cracker di password più veloci disponibili.

4 Considerazioni preliminari LINUX

Il tipo di hash che viene utilizzato su questo Sistema operativo è yescrypt, esso è una funzione di derivazione della chiave crittografica utilizzata per l'hashing della password su Fedora, Debian, Ubuntu e Arch Linux. Questa funzione è più resistente agli attacchi di cracking delle password offline rispetto a SHA-512. È basato su Scrypt. yescrypt è una funzione di derivazione della chiave basata su password (KDF) e uno schema di hashing delle password progettato per essere resistente agli attacchi di cracking delle password offline. È basato su scrypt, ma apporta diverse modifiche per migliorarne la sicurezza e le prestazioni.

4.1 Vantaggi di yescrypt:

- **Maggiore resistenza agli attacchi brute force:** yescrypt richiede più calcoli rispetto a funzioni hash come SHA-256, rendendo più difficile per gli aggressori craccare le password mediante attacchi brute force.
- **Scalabilità:** yescrypt può essere regolato per utilizzare più memoria e tempo di calcolo, rendendolo adatto a sistemi con diverse risorse.
- **Compatibilità:** yescrypt è compatibile con le password hashate con scrypt, il che facilita la migrazione da scrypt a yescrypt.

yescrypt

prefisso

"\$ y \$"

Formato dell'hash della passphrase

\$y\$[./A-Za-z0-9]+\$[./A-Za-z0-9],86\$[./A-Za-z0-9]43

Lunghezza massima della passphrase

Illimitata

Dimensione dell'hash

256 bit

Dimensione del salt

Fino a 512 bit

Parametro del costo del tempo CPU

Da 1 a 11 (logaritmico)

Un po' di storia di Linux

L'algoritmo di hash utilizzato per proteggere le password e i dati in Linux è cambiato nel corso degli anni per garantire la massima sicurezza contro le minacce in evoluzione. Ecco una panoramica di questa evoluzione:

- **1991-2004: MD5:** La distribuzione originale di Linux utilizzava l'algoritmo di hash MD5 per la memorizzazione delle password e la verifica dell'integrità dei file. MD5 era considerato sicuro all'epoca, ma in seguito sono state scoperte vulnerabilità che lo rendevano vulnerabile agli attacchi di collisione e di forza bruta.
- **2005-2013: SHA-1:** Per ovviare alle carenze di MD5, nel 2005 è stato adottato lo standard SHA-1. SHA-1 offriva una maggiore sicurezza rispetto a MD5, ma anche questo algoritmo ha subito attacchi con il tempo.
- **2013-presente: SHA-256 e successivi:** A partire dal 2013, le distribuzioni Linux hanno iniziato a migrare verso algoritmi di hash più robusti della famiglia SHA-2, in particolare SHA-256. Algoritmi come SHA-256, SHA-384 e SHA-512 offrono livelli di sicurezza significativamente più elevati rispetto a MD5 e SHA-1 e sono considerati resistenti agli attacchi attuali e prevedibili per il futuro. Oltre agli algoritmi hash standard, alcune distribuzioni Linux supportano anche algoritmi alternativi:
 - bcrypt: Un algoritmo basato su costi elevati che rende gli attacchi di forza bruta più onerosi.
 - scrypt: Un altro algoritmo basato su costi elevati simile a bcrypt, progettato per essere resistente agli attacchi basati su GPU.
 - argon2: Un algoritmo moderno e ad alte prestazioni progettato per resistere agli attacchi attuali e futuri.

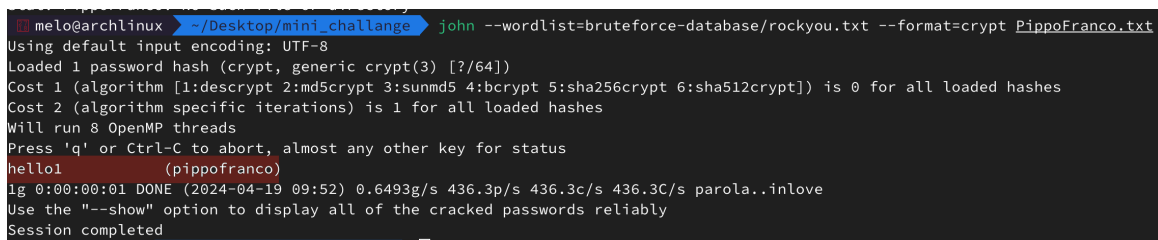
5 Attacco su Linux

```
$ sudo cat /etc/passwd
$ sudo cat /etc/shadow
$ unshadow passwd.txt shadow.txt > Filename.txt.
```

Il comando unshadow combina i contenuti dei file passwd e shadow in modo da creare un unico file che contiene sia le informazioni utente sia le password criptate. Questo è utile per i programmi di cracking delle password come John the Ripper, che richiedono un formato di input specifico per funzionare efficacemente.

5.1 Attacco a Dizionario

E' Possibile scaricare dei dizionari da internet(repository su github), uno dei più famosi è **rockyou.txt** che è facilmente scaricabile e verrà inserito in un'apposita cartella per far sì che i tool possano riuscire ad utilizzarlo.



```
melo@archlinux ~/Desktop/mini_challenge john --wordlist=bruteforce-database/rockyou.txt --format=crypt PippoFranco.txt
Using default input encoding: UTF-8
Loaded 1 password hash (crypt, generic crypt(3) [?/64])
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 0 for all loaded hashes
Cost 2 (algorithm specific iterations) is 1 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
hello1 (pippofranco)
lg 0:00:00:01 DONE (2024-04-19 09:52) 0.6493g/s 436.3p/s 436.3c/s 436.3C/s parola..inlove
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Figure 1: Esempio Attacco a dizionario

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=crypt File.txt
```

Comando	Descrizione
john	Il comando principale per eseguire John the Ripper, un popolare strumento di cracking delle password.
-wordlist=/usr/share/wordlists/rockyou.txt	Specifica il percorso della wordlist da utilizzare per il cracking delle password. In questo caso, viene usata la wordlist RockYou, molto comune in test di penetrazione per il cracking di password.
-format=crypt	Imposta il formato dell'hash che John deve tentare di decifrare. L'opzione "crypt" indica che il file contiene hash generati tramite l'algoritmo Unix crypt.
File.txt	Il file di input che contiene gli hash delle password da crackare.

HASCAT: Al 17 aprile 2024, Hashcat non è in grado di trovare nativamente una corrispondenza con un algoritmo di tipo yescrypt.

5.2 Forza Bruta

```
$ john --incremental=Lower --min-length=4 --max-length=4 --format=crypt passw.txt
```

Comando/Opzione	Descrizione
john	Comando principale di John the Ripper, utilizzato per avviare il programma.
-incremental=Lower	Opzione che specifica un attacco incrementale, utilizzando solo caratteri minuscoli.
-min-length=4	Opzione che imposta la lunghezza minima delle password da testare a 4 caratteri.
-max-length=4	Opzione che imposta la lunghezza massima delle password da testare a 4 caratteri, testando solo password di lunghezza esattamente 4 caratteri.
-format=crypt	Opzione che specifica il formato dell'hash delle password nel file bfjtr.txt , indicando che il file contiene hash generati utilizzando la funzione di hash crittografica standard.
bfjtr.txt	Nome del file che contiene gli hash delle password su cui verrà eseguito l'attacco.

L'aver inserito dei parametri che sono esplicitati nella tabella sovrastante, è stato un mossa puramente a scopo di risparmiare tempo. E' possibile togliere l'attributo Lower e max e min length.

5.3 Bonus: Attacco a maschera

```
john --wordlist=bruteforce-database/rockyou.txt --format=crypt --mask='?w99' passw.txt
```

Comando/Opzione	Descrizione
john	
--wordlist=bruteforce-database/rockyou.txt	Opzione che specifica il percorso della wordlist da utilizzare per l'attacco di forza bruta. Nel comando fornito, la wordlist rockyou.txt è situata nella directory bruteforce-database/ .
--format=crypt	Opzione che specifica il formato dell'hash delle password nel file di input. In questo caso, l'opzione crypt indica che il file contiene hash generati utilizzando la funzione di hash crittografica standard.
--mask='?w99'	Opzione che specifica la maschera da utilizzare per generare le password durante l'attacco di forza bruta. La maschera ?w99 indica che verranno testate tutte le password composte da un carattere alfabetico seguito da 99 caratteri alfanumerici.
passw.txt	Nome del file che contiene gli hash delle password su cui verrà eseguito l'attacco.

5.4 Attacco rainbow table

Le rainbow table sono un metodo utilizzato dagli attaccanti per decifrare le password criptate memorizzate all'interno di un sistema. In sistemi moderni, le password non sono memorizzate in chiaro, ma piuttosto vengono crittografate utilizzando una funzione di hashing. Quando un utente inserisce la sua password per l'autenticazione, il sistema confronta l'hash della password inserita con quello memorizzato. Le rainbow table sono essenzialmente tabelle precompilate di associazioni tra password e i loro hash corrispondenti. Un attaccante che possiede l'hash della password di una vittima può confrontarlo con le voci della tabella per trovare una corrispondenza.

Queste tabelle contengono una vasta gamma di possibili password, comprese quelle comuni e generate casualmente, al fine di massimizzare la probabilità di trovare una corrispondenza. Le tabelle sono costruite a partire da un ampio insieme di password comuni, estese poi con password generate in modo casuale. La costruzione delle rainbow tables si basa sul concetto di funzioni di hashing, che trasformano una stringa di testo in una stringa di lunghezza fissa, e funzioni di riduzione, che mappano un hash in una password. Le *rainbow chains*, basate su queste funzioni, rappresentano una sequenza di trasformazioni alternate:

$$\text{password} \longrightarrow \text{hash} \longrightarrow \text{password} \longrightarrow \text{hash} \longrightarrow \dots$$

Queste sequenze sono utilizzate per costruire le tabelle, facilitando il processo di recupero delle password partendo dai loro hash, riducendo così il tempo necessario per trovare corrispondenze tramite tecniche di brute force.

Per aumentare le possibilità di successo e ridurre le probabilità di collisione, ad ogni passaggio di riduzione viene utilizzata una funzione di riduzione diversa, dando così il nome alle rainbow chain e alle rainbow table. Tuttavia, l'uso del sale (salt) rende le rainbow table inefficienti, poiché il sale aggiunge una stringa casuale alla password prima dell'hashing, rendendo ogni hash unico anche per la stessa password.

5.4.1 Prova Linux

Ho scaricato a scopo didattico una rainbow table di pochi giga che mi permette di avere delle rainbow table risposte a password in pochi secondi

- **numeric#1-12:** Tipo utilizzato per la tabella
- **<http://www.sha1-online.com/>:** Sito web per calcolare l'hash SHA-1 online.
- **<https://tobtu.com/mysqlsha1.php>:** Strumento online per calcolare l'hash SHA-1 utilizzato in MySQL.

```
$ ./rcracki_mt -h 511964B5BF5886B21B9497DD0B363E175848D705 .
```

Comando per eseguire il cracking dell'hash SHA-1 utilizzando il software rcracki_mt.

6 Attacco su windows

Il sistema operativo Windows utilizza invece NTLM, abbreviazione di NT Lan Manager, rappresenta una suite di protocolli e funzioni hash utilizzata per l'autenticazione di computer e utenti all'interno degli ambienti Windows. Sebbene esistano protocolli di autenticazione più sicuri, come Kerberos, NTLM è ancora ampiamente utilizzato come metodo di autenticazione di riserva e per garantire la compatibilità con le versioni precedenti nelle reti Active Directory di Windows.

Quando un utente tenta di accedere a un computer Windows, il Domain Controller responsabile dell'autenticazione cercherà prima di utilizzare l'autenticazione Kerberos. Nel caso in cui questa operazione fallisca a causa di una configurazione errata o di problemi di connettività del dominio, verrà utilizzata l'autenticazione NTLM come soluzione di fallback. NTLM si basa su un protocollo challenge-response che sfrutta gli hash delle password degli utenti per autenticare gli accessi.

6.0.1 Tool utilizzato per windows

Mimikatz è un'applicazione open-source e uno strumento post-esplorazione del sistema operativo Windows che consente agli utenti di visualizzare le credenziali di autenticazione. Questo strumento fornisce hash dal file SAM del sistema operativo Windows agli utenti. Poiché Windows memorizza i dati delle password in un hash NTLM, il team forense può utilizzare lo strumento Mimikatz per ottenere la stringa di hash e poi utilizzare lo strumento hashcat per ottenere il testo in chiaro e utilizzarlo per l'accesso al computer di destinazione.

1. Accedere alla seguente directory:

C:\WINDOWS\System32\config

2. Copiare i seguenti file nella cartella Desktop:

- SAM (file)
- SYSTEM (file)

3. Installare Mimikatz dal repository GitHub.

4. IMPORTANTE: Disabilitare Windows Defender.

Note aggiuntive:

- Mimikatz è disponibile su GitHub per l'installazione.
- È importante disabilitare Windows Defender per garantire il corretto funzionamento di Mimikatz.
- Una volta installato Mimikatz, è possibile utilizzarlo per recuperare le credenziali di autenticazione dal file SAM.

Utilizzare cmd con permessi di amministratore

```
token::elevate  
lsadump::sam  
sekurlsa::logonpasswords
```

Prendiamo l'hash e lo incolliamo in un txt che daremo poi ai nostri tool.

6.1 Attacco a dizionario

```
$ john --wordlist=bruteforce-database/rockyou.txt --format=NT windows.txt
```

Comando/Opzione	Descrizione
john	Comando principale di John the Ripper.
--wordlist=bruteforce-database/rockyou.txt	Opzione che specifica il percorso della wordlist da utilizzare per l'attacco di forza bruta. Nel comando fornito, la wordlist <code>rockyou.txt</code> è situata nella directory <code>bruteforce-database/</code> .
--format=NT	Opzione che specifica il formato dell'hash delle password nel file di input. In questo caso, l'opzione NT indica che il file contiene hash di tipo NTLM.
windows.txt	Nome del file che contiene gli hash delle password su cui verrà eseguito l'attacco.

6.2 Attacco forza bruta:

```
$ john --incremental=Lower --min-length=4 --max-length=4 --format=nt windows.txt
```

Comando/Opzione	Descrizione
john	Comando principale di John the Ripper.
--incremental=Lower	Opzione che specifica il tipo di attacco da eseguire. In questo caso, --incremental indica un attacco incrementale, e Lower specifica che saranno utilizzati solo caratteri minuscoli.
--min-length=4	Opzione che specifica la lunghezza minima delle password da testare. Nel comando fornito, la lunghezza minima è impostata su 4 caratteri.
--max-length=4	Opzione che specifica la lunghezza massima delle password da testare. Nel comando fornito, la lunghezza massima è impostata su 4 caratteri, il che significa che verranno testate solo password di lunghezza esattamente 4 caratteri.
--format=nt	Opzione che specifica il formato dell'hash delle password nel file di input. In questo caso, l'opzione nt indica che il file contiene hash di tipo NTLM.
bfw.txt	Nome del file che contiene gli hash delle password su cui verrà eseguito l'attacco.

6.3 Attacco rainbow table

```
$ ./rcracki_mt -h 12597A048565304F5DE19A4003B79260 ./ntlm_mixalpha-numeric-space#1-7_0
```

```
melo@archlinux ~$ ./Desktop/mini_challenge/rainbow_table/./rcracki_mt -h 12597A048565304F5DE19A4003B79260 ./ntlm_mixalpha-numeric-space#1-7_0
Using 1 threads for pre-calculation and false alarm checking...
Found 11 rainbowtable files...

ntlm_mixalpha-numeric-space#1-7_0_10000x19739301_distrtrtgen[p][i]_10.rti2
Chain Position is now 19739301
118435806 bytes read, disk access time: 0.33s
searching for 1 hash...
cryptanalysis time: 4.27 s

ntlm_mixalpha-numeric-space#1-7_0_10000x67108864_distrtrtgen[p][i]_00.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 1.13s
searching for 1 hash...
cryptanalysis time: 0.19 s

ntlm_mixalpha-numeric-space#1-7_0_10000x67108864_distrtrtgen[p][i]_01.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 1.00s
searching for 1 hash...
cryptanalysis time: 0.18 s

ntlm_mixalpha-numeric-space#1-7_0_10000x67108864_distrtrtgen[p][i]_02.rti2
Chain Position is now 67108864
402653184 bytes read, disk access time: 1.00s
searching for 1 hash...
plaintext of 12597a048565304f5de19a4003b79260 is Toro200
cryptanalysis time: 0.08 s

statistics
-----
plaintext found:          1 of 1(100.00%)
total disk access time:   3.46s
total cryptanalysis time: 0.50s
total pre-calculation time: 4.22s
total chain walk step:    49985001
total false alarm:        1494
total chain walk step due to false alarm: 5798827

result
-----
12597a048565304f5de19a4003b79260 Toro200 hex:546f726f323030
```

Figure 2: Esempio Attacco rainbow table Windows

Comando/Opzione	Descrizione
\$./rcracki_mt	Il comando principale per avviare il tool rcracki_mt per il cracking dell'hash.
h	Opzione che specifica l'hash da decifrare
12597A048565304F5DE19A4003B79260	Nel comando fornito è l'hash specificato
./ntlm_mixalpha-numeric-space#1-7_0	Parametro che specifica il file del set di caratteri. Nel comando fornito, si utilizza il set di caratteri misti alfanumerici, con una lunghezza delle password compresa tra 1 e 7 caratteri.

7 Tradeoff complessità-prestazione

La scelta di un algoritmo di hash appropriato comporta un trade-off tra complessità e prestazioni.

Da un lato, algoritmi di hash più complessi offrono una maggiore resistenza agli **attacchi crittografici**, rendendo più difficile la decifratura dei dati originali a partire dall'hash o la generazione di collisioni. Questa complessità aggiuntiva può essere il risultato di un maggior numero di iterazioni, una maggiore lunghezza dell'output dell'hash o l'utilizzo di tecniche crittografiche avanzate. Tuttavia, questa maggiore complessità può comportare un costo in termini di prestazioni computazionali, rallentando le operazioni di hashing e aumentando il carico sulle risorse di sistema.

Dall'altro lato, algoritmi di hash più veloci possono garantire prestazioni migliori e tempi di risposta più rapidi, consentendo operazioni di hashing più efficienti e meno onerose per le risorse di sistema. Tuttavia, questa maggiore efficienza potrebbe essere ottenuta a spese della sicurezza, poiché gli algoritmi più veloci potrebbero essere più vulnerabili ad attacchi crittografici come la ricerca di collisioni o la forza bruta.

Il trade-off tra complessità e prestazioni dipende anche dalle risorse disponibili nel sistema e dai requisiti di sicurezza specifici del contesto di utilizzo. Su hardware più potente, è possibile utilizzare algoritmi di hash più complessi senza compromettere significativamente le prestazioni. Mentre su dispositivi con risorse limitate potrebbe essere necessario sacrificare la complessità dell'algoritmo per garantire un funzionamento accettabile.

Per marcare il concetto, ho creato uno script che permette di vedere intrinsecamente questo trade-off.

```
1 import time
2 from prettytable import PrettyTable
3 from passlib.hash import (sha256_crypt, sha512_crypt, bcrypt, argon2, des_crypt, lmhash,
4                             nhash,
5                             apr_md5_crypt, bigcrypt, bsdi_crypt, crypt16, cta_pbkdf2_sha1,
6                             dlitz_pbkdf2_sha1,
7                             fshp, grub_pbkdf2_sha512, hex_md4, hex_md5, hex_sha1, hex_sha256,
8                             hex_sha512,
9                             ldap_bcrypt, ldap_bsdi_crypt, ldap_des_crypt, ldap_md5,
10                            ldap_md5_crypt, ldap_salted_md5,
11                            ldap_salted_sha1, ldap_sha1, ldap_sha1_crypt, ldap_sha256_crypt,
12                            ldap_sha512_crypt,
13                            msdcc, msdcc2, mssql2000, mssql2005, mysql323, mysql41, nhash,
14                            oracle10, oracle11,
15                            phpasp, postgres_md5, roundup_plaintext, scram, sun_md5_crypt,
16                            unix_disabled)
17
18 # List of passlib hash algorithms to compare
19 algorithms = ['sha256_crypt', 'sha512_crypt', 'bcrypt', 'argon2', 'des_crypt', 'lmhash',
20               'nhash',
21               'apr_md5_crypt', 'bigcrypt', 'bsdi_crypt', 'crypt16', 'cta_pbkdf2_sha1',
22               'dlitz_pbkdf2_sha1',
23               'fshp', 'grub_pbkdf2_sha512', 'hex_md4', 'hex_md5', 'hex_sha1', 'hex_sha256',
24               'hex_sha512',
25               'ldap_bcrypt', 'ldap_bsdi_crypt', 'ldap_des_crypt', 'ldap_md5',
26               'ldap_md5_crypt', 'ldap_salted_md5',
27               'ldap_salted_sha1', 'ldap_sha1', 'ldap_sha1_crypt', 'ldap_sha256_crypt',
28               'ldap_sha512_crypt',
29               'msdcc', 'msdcc2', 'mssql2000', 'mssql2005', 'mysql323', 'mysql41', 'nhash',
30               'oracle10', 'oracle11',
31               'phpasp', 'postgres_md5', 'roundup_plaintext', 'scram', 'sun_md5_crypt',
32               'unix_disabled']
33
34 algorithm_dates = {
35     'sha256_crypt': '2001', # SHA-256 is part of the SHA-2 set, published in 2001
36     'sha512_crypt': '2001', # SHA-512 is also part of the SHA-2 set
37     'bcrypt': '1999',
38     'argon2': '2015', # Winner of the Password Hashing Competition in 2015
39     'des_crypt': '1975', # DES was developed in the early 1970s and published in 1975
40     'lmhash': '1985', # The LM hash was used in Microsoft LAN Manager, which was released
41                     in 1985
42     'nhash': '1993', # The NT hash was introduced with Windows NT 3.1 in 1993
43     'apr_md5_crypt': '1995', # MD5 was published in 1992, but this variant seems to have
44                     been introduced later
45     'bigcrypt': '1976', # Bigcrypt is a variant of DES, which was published in 1975
```

```

30 'bsdi_crypt': '1994', # BSDi crypt is a variant of DES
31 'crypt16': '1985', # Crypt16 is a variant of DES
32 'cta_pbkdf2_sha1': '2000', # PBKDF2 was published as part of RSA Laboratories' PKCS #5 v2.0
33 'dlitz_pbkdf2_sha1': '2000', # PBKDF2 was published as part of RSA Laboratories' PKCS #5 v2.0
34 'fshp': '2013', # Fully Secure Hash Protocol (FSHP) was proposed in 2013
35 'grub_pbkdf2_sha512': '2000', # PBKDF2 was published as part of RSA Laboratories' PKCS #5 v2.0
36 'hex_md4': '1990', # MD4 was published in 1990
37 'hex_md5': '1992', # MD5 was published in 1992
38 'hex_sha1': '1995', # SHA-1 was published in 1995
39 'hex_sha256': '2001', # SHA-256 is part of the SHA-2 set, published in 2001
40 'hex_sha512': '2001', # SHA-512 is also part of the SHA-2 set
41 'ldap_bcrypt': '1999', # Bcrypt was published in 1999
42 'ldap_bsdi_crypt': '1994', # BSDi crypt is a variant of DES
43 'ldap_des_crypt': '1975', # DES was developed in the early 1970s and published in 1975
44 'ldap_md5': '1992', # MD5 was published in 1992
45 'ldap_md5_crypt': '1992', # MD5 was published in 1992
46 'ldap_saltmd5': '1992', # MD5 was published in 1992
47 'ldap_saltsha1': '1995', # SHA-1 was published in 1995
48 'ldap_sha1': '1995', # SHA-1 was published in 1995
49 'ldap_sha1_crypt': '1995', # SHA-1 was published in 1995
50 'ldap_sha256_crypt': '2001', # SHA-256 is part of the SHA-2 set, published in 2001
51 'ldap_sha512_crypt': '2001', # SHA-512 is also part of the SHA-2 set
52 'msdccc': '1998', # MS DCC is used in Windows NT and was introduced in 1998
53 'msdccc2': '1998', # MS DCC2 is used in Windows NT and was introduced in 1998
54 'mssql2000': '2000', # MS SQL 2000 uses a proprietary hash function
55 'mssql2005': '2005', # MS SQL 2005 uses a proprietary hash function
56 'mysql323': '1995', # MySQL 323 uses a proprietary hash function
57 'mysql41': '2001', # MySQL 4.1 uses a proprietary hash function
58 'oracle10': '2005', # Oracle 10 uses a proprietary hash function
59 'oracle11': '2007', # Oracle 11 uses a proprietary hash function
60 'phpass': '2005',
61 'postgres_md5': '1996', # PostgreSQL started using MD5 in 1996
62 'roundup_plaintext': '2001', # Roundup is a bug-tracking system that was first released in 2001
63 'scram': '2010',
64 'sun_md5_crypt': '1995', # This is a variant of MD5, which was published in 1992
65 'unix_disabled': '1970', # Unix disabled is not a hash function, but a marker for disabled accounts
66 }
67
68 def hash_data(algorithm, data):
69     hash_algo = globals()[algorithm]
70     accepts_rounds = False
71     hash_value = None
72
73     try:
74         try:
75             hash_value = hash_algo.using(rounds=15, relaxed=True).hash(data) # check if the
76                                     algorithm accepts rounds
77             accepts_rounds = True
78         except TypeError:
79             hash_value = hash_algo.hash(data) # if it doesn't accept rounds, hash the data
80                                     without them
81     except TypeError:
82         hash_value = globals()[algorithm].hash(data, user='user') # if the algorithm
83                                     requires a user parameter, provide it
84
85     return hash_value, accepts_rounds
86
87 def create_table_row(algorithm, execution_time, accepts_rounds, creation_date, hash_value):
88     # Create a row for the table
89     display_hash_value = hash_value[:50] + "..." if len(hash_value) > 50 else hash_value
90     return [algorithm, execution_time, accepts_rounds, creation_date, display_hash_value]

```

```

87
88 def create_hash_comparison_table(algorithms, data, algorithm_dates): # Create a table to
    compare the hash algorithms
89     table = PrettyTable()
90     table.field_names = ["Algorithm", "Execution_Time", "Accepts_Rounds_(Work_Factor)",
        "Created_At", "Hash_Value"]
91
92     for algorithm in algorithms:
93         start_time = time.time()
94         hash_value, accepts_rounds = hash_data(algorithm, data)
95         end_time = time.time()
96         execution_time = end_time - start_time
97         creation_date = algorithm_dates.get(algorithm)
98         table.add_row(create_table_row(algorithm, execution_time, accepts_rounds,
        creation_date, hash_value))
99
100     table.sortby = "Execution_Time"
101     return table
102
103 # Data to hash
104 data = 'F'
105 # Define the table
106 table = create_hash_comparison_table(algorithms, data, algorithm_dates)

```

7.1 Criticità

Negli algoritmi di hashing delle password, il tradeoff tra complessità e prestazioni è un aspetto critico da considerare. Da un lato, si desidera un algoritmo sufficientemente complesso da resistere agli attacchi di cracking, dall'altro si vuole che sia efficiente in termini di tempo e risorse di calcolo per garantire un'esperienza utente fluida.

Fattori che influenzano la complessità

- Lunghezza del digest: Un digest più lungo offre una maggiore sicurezza, ma richiede più calcoli.
- Numero di round: Un maggior numero di round aumenta la resistenza agli attacchi, ma aumenta anche il tempo di esecuzione.
- Funzioni utilizzate: L'utilizzo di funzioni complesse e non lineari migliora la sicurezza, ma può rallentare l'algoritmo.

Fattori che influenzano le prestazioni

- Tempo di esecuzione: Quanto tempo impiega l'algoritmo per generare un hash da una password?
- Consumo di memoria: Quanta memoria richiede l'algoritmo per funzionare?
- Consumo di CPU: Quanta potenza di calcolo consuma l'algoritmo?

Esempi

- MD5: Un algoritmo relativamente semplice e veloce, ma con alcune vulnerabilità note.
- SHA-256: Più sicuro di MD5, ma con un tempo di esecuzione leggermente superiore.
- bcrypt: Un algoritmo basato su cost-factor, che regola la complessità in base alle risorse di calcolo disponibili.

Scegliere l'algoritmo giusto

La scelta dell'algoritmo di hashing ideale dipende da diversi fattori, tra cui:

- Livello di sicurezza richiesto: In ambienti ad alto rischio, come quelli che gestiscono dati sensibili, è consigliabile utilizzare algoritmi più complessi.

- **Prestazioni:** Se l'applicazione deve gestire un elevato numero di autenticazioni, è importante scegliere un algoritmo efficiente.
- **Compatibilità:** Assicurarsi che l'algoritmo scelto sia supportato dalle librerie e dai framework utilizzati.

Considerazioni aggiuntive

- **Salt:** L'utilizzo di un salt, ovvero una stringa casuale aggiunta alla password prima dell'hashing, aumenta la sicurezza e rende più difficile il cracking delle password.
- **Evoluzione delle minacce:** È importante monitorare le vulnerabilità emergenti e aggiornare gli algoritmi di hashing di conseguenza.

Efficienza e Innovazione di Argon2

Argon2 è considerato particolarmente efficace e innovativo per diversi motivi:

Resistenza agli attacchi di forza bruta

Argon2 è progettato per essere intensivo sia in termini di memoria che di tempo di elaborazione, rendendo difficile attaccarlo con metodi di forza bruta. Ciò implica che gli attaccanti non possono facilmente generare tutte le possibili combinazioni di password per trovare quella corretta.

Configurabilità

Argon2 offre diversi parametri configurabili, tra cui il costo di memoria, il costo di tempo e il parallelismo. Questo permette agli implementatori di bilanciare la sicurezza e le prestazioni in base alle esigenze specifiche del loro sistema e hardware.

Protezione contro attacchi side-channel

Argon2 cerca di proteggersi dagli attacchi che sfruttano le informazioni derivanti dall'implementazione fisica dell'algoritmo, come i tempi di accesso alla memoria, che possono essere utilizzate per dedurre informazioni sulla password o sulla chiave.

Versioni multiple per diversi casi d'uso

Argon2 presenta tre versioni principali:

- **Argon2d:** Ottimizzata per le applicazioni che richiedono la massima resistenza contro gli attacchi via GPU.
- **Argon2i:** Ottimizzata per le applicazioni che richiedono resistenza contro gli attacchi side-channel basati sulla cache di memoria.
- **Argon2id:** Una combinazione dei due che offre un bilanciamento tra resistenza agli attacchi via GPU e resistenza agli attacchi side-channel.

Uso di memoria e tempo

Argon2 utilizza sia la memoria che il tempo in modo tale che l'allocazione di grandi quantità di memoria per brevi periodi o piccole quantità per periodi lunghi sia inefficiente per gli attaccanti.

8 Conclusioni

La sfida è stata davvero stimolante; perché mi ha permesso di acquisire conoscenze che solitamente non vengono trattate in questo corso. Grazie all'utilizzo di una varietà di strumenti, ho ampliato la mia comprensione della sicurezza informatica, in particolare su quanto possa essere difficile ottenere delle credenziali di accesso e le tecniche di attacco utilizzate nei vari sistemi.

L'esperienza pratica, unita alla teoria appresa, mi ha permesso di sviluppare una visione più approfondita, che comprende sia la prevenzione sia la capacità di proteggere dati o addirittura interi sistemi.

Nel complesso la challenge è stata eseguita sotto tutti i punti di vista ed è stato completato ogni task assegnato.