ALMA MATER STUDIORUM - UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Attacchi offline di tipo brute-force: metodologie per la generazione di dizionari di password e comparazione dei principali strumenti

Presentata da:

Simone Cancello

Relatore:

Rebecca Montanari

Introduzione

Il tirocinio è stato svolto presso Cyberloop srl azienda partner di Imola informatica spa. Presso di loro ho potuto affrontare un tema molto importante ad oggi la sicurezza informatica, in particolar modo le tecniche di penetration test e simulazione di attacco in ambito bruteforce che risultano estremamente più efficienti se accompagnate da informazioni di ricognizione personalizzate sulla base della vittima mediante opportune tecniche di data intelligence.

Questo rapporto considera la generazione di password e la loro effettiva applicazione al problema del controllo dell'accesso alle risorse, dopo aver descritto la necessità e gli usi delle password. Il successo della coppia password/nome utente come autenticazione è dovuto alla loro semplicità, anche per il fatto che si possono conservare, dopo un'operazione di criptazione mediante hashing, in una semplice tabella, che nel caso di siti web si può trovare all'interno di un server remoto, mentre per autenticazione offline si trova all'interno del dispositivo locale. Questa semplicità si rispecchia anche nella possibilità di effettuare attacchi, soprattutto dovuto al punto debole del sistema: l'uomo.

Parole chiave: sicurezza; password; brute force; cracking; identificazione; autenticazione personale.

Sommario

Capitolo 1

1 Introduzione alla crittografia

La crittografia ha una storia lunga e affascinante. Il suo uso iniziale risale agli egiziani circa 4000 anni fa ed è usata tuttora. Nel corso del tempo ha avuto vari approcci che sono tutt'ora in sviluppo. L'utilizzo maggiore delle tecniche crittografiche era associato all'esercito, ai servizi diplomatici e al governo in generale. La crittografia è stata utilizzata come strumento per proteggere i segreti e le strategie nazionali fin dagli albori.

La proliferazione di computer e sistemi di comunicazione negli anni '60 ha portato con sé la richiesta da parte del settore privato di mezzi per proteggere le informazioni in forma digitale e per fornire servizi di sicurezza. Iniziato con il lavoro di Feistel presso IBM nei primi anni '70 e culminato nel 1977 con l'adozione come standard di elaborazione delle informazioni federale degli Stati Uniti per la crittografia di informazioni non classificate, DES, il Data Encryption Standard, è il meccanismo crittografico più noto della storia. Rimane il mezzo standard per garantire il commercio elettronico per molte istituzioni finanziarie in tutto il mondo [1].

Nel corso dei secoli, è stato creato un insieme elaborato di protocolli e meccanismi per affrontare i problemi di sicurezza delle informazioni. Spesso gli obiettivi di sicurezza delle informazioni non possono essere raggiunti solo attraverso algoritmi e protocolli matematici, ma richiedono tecniche procedurali per raggiungere il risultato desiderato. Concettualmente, il modo in cui le informazioni vengono registrate non è cambiato drasticamente nel tempo. Mentre le informazioni venivano generalmente archiviate e trasmesse su carta, gran parte di esse ora risiede su supporti magnetici e viene trasmessa tramite sistemi di telecomunicazione. Ciò che è cambiato radicalmente è la capacità di copiare e alterare le informazioni. Si possono fare migliaia di copie identiche di un'informazione archiviata elettronicamente e ognuna è indistinguibile dall'originale. Ciò che è necessario per una società in cui le informazioni sono per lo più archiviate e trasmesse in forma elettronica è un mezzo per garantire la sicurezza dei dati che sia indipendente dal supporto fisico che li registra o li trasmette e tale che gli obiettivi della sicurezza si basino esclusivamente sull'informazione digitale stessa.

Quindi cosa è la crittografia? La crittografia è lo studio delle tecniche matematiche relative agli aspetti della sicurezza delle informazioni come la riservatezza, l'integrità dei dati, l'autenticazione dell'entità e l'autenticazione dell'origine dei dati. La crittografia non è l'unico mezzo per garantire la sicurezza delle informazioni, ma piuttosto un insieme di tecniche. Gli obiettivi sono:

1. **La riservatezza**: è un servizio utilizzato per mantenere segreto il contenuto delle informazioni a tutti, esclusi gli autorizzati. Segretezza è un termine sinonimo di riservatezza e privacy. Esistono numerosi approcci per garantire la riservatezza, dalla protezione fisica agli algoritmi matematici che rendono i dati incomprensibili.
2. **L'integrità dei dati**: è un servizio che affronta l'alterazione non autorizzata dei dati. Per garantire l'integrità dei dati, è necessario essere in grado di rilevare la manipolazione dei dati da parte di soggetti non autorizzati. La manipolazione dei dati include cose come l'inserimento, la cancellazione e la sostituzione.
3. **L'autenticazione**: è un servizio relativo all'identificazione. Questa funzione si applica sia alle entità che alle informazioni stesse. Questo aspetto della crittografia è solitamente suddiviso in due classi principali: autenticazione dell'entità e autenticazione dell'origine dei dati. L'autenticazione dell'origine dei dati fornisce implicitamente l'integrità dei dati (se un messaggio viene modificato, l'origine è cambiata).
4. **Il non ripudio**: è un servizio che impedisce a un soggetto di negare impegni o azioni precedenti. Quando sorgono controversie a causa di un'entità che nega che siano state intraprese determinate azioni, è necessario un mezzo per risolvere la situazione. Per risolvere la controversia è necessaria una procedura che coinvolga una terza parte di fiducia.

Un obiettivo fondamentale della crittografia è affrontare adeguatamente queste quattro aree sia nella teoria che nella pratica. La crittografia riguarda la prevenzione e il rilevamento di contraffazioni e altre attività dannose.

1.1 Introduzione al password cracking

Il rapido sviluppo delle tecnologie Internet, dei social network e di altre aree correlate, ha portato l'autenticazione dell'utente sempre più importante per proteggere i dati. La RFC 2828 definisce l'autenticazione dell'utente come "il processo di verifica di un'identità

rivendicata da o per un'entità di sistema" [2]. Il servizio di autenticazione deve garantire che la connessione non sia interferita da una terza parte mascherata da una delle due parti legittime. Tipicamente ci sono quattro modi per autenticare l'identità dell'utente in base a:

1. **Un fattore di conoscenza**: qualcosa che l'individuo conosce, ad esempio la password, PIN, risposte a domande prestabilite.
2. **Un fattore di possesso**: qualcosa che l'individuo possiede come un token, una smartcard, chiave elettronica, chiave fisica.
3. **Un elemento identificativo** (biometrica statica): qualcosa che l'individuo è. Esempi sono le impronte digitali, retina, viso.
4. **Biometrica dinamica**: qualcosa che l'individuo fa. Possono essere pattern vocale, scrittura a mano, ritmo di battitura.

Nei diversi metodi, l'autenticazione con password, la prima casistica dell'elenco puntato, è la linea di difesa più diffusa per accedere ai servizi.

L'autenticazione basata su password funziona confrontando le credenziali fornite da un utente con le password archiviate. Poiché gli utenti non autorizzati possono avere accesso a quest'ultime, è necessario che le password vengano crittografate durante l'archiviazione utilizzando funzioni di hash crittografiche, come MD5, SHA-1.

Il crack delle password può essere definito come il recupero di testi di password semplici da una posizione memorizzata che di solito è crittografata. Il crack delle password è il processo per ottenere le password in chiaro dal segreto crittografato memorizzato, o almeno uno equivalente [3]. In generale, il cracking delle password nel mondo dell'hacking spazia dalla decrittografia degli hash delle password sottratti da un database fino all'hacking delle reti wireless. Quindi l'utente fornisce l'identificativo e la password, il sistema confronta la password con quella memorizzata. Tuttavia, alcuni utenti non prestano attenzione alla riservatezza o alla complessità delle loro password pensando di non avere file privati su Internet. Ciò consente ai cracker malintenzionati di danneggiare l'intero sistema se forniscono un punto di ingresso al sistema [4]. Inoltre, l'evoluzione tecnologica e il miglioramento delle componenti hardware hanno reso le minacce di cracker di sistema, furto e danneggiamento dei dati più facili che in passato [5].

Quindi tra il soggetto chiamato attaccante, che proverà a trovare la password, e il soggetto chiamato difendente, che proverà a creare una password più robusta possibile, si instaurerà un rapporto, che sarà sconosciuto al difendente. L'attaccante tenterà di trovare

una stringa che darà la soluzione dell'hash in suo possesso, ci proverà finché avrà tempo, risorse o tentativi a disposizione.

Il problema del difendete, per la maggior parte delle volte, è che tendono a rientrare in queste categorie, facilitando così il lavoro dell'attaccante:

1. **Password popolari**: numerose ricerche, [6] [7] [8] [9], hanno dimostrato che gli utenti spesso scelgono parole semplici come password o creano semplici stringhe trasformate per soddisfare i requisiti della strategia di impostazione della password del sito web. Di norma chiedono stringhe alfanumeriche, l'utente quindi userà "123456a" e soddisferà i requisiti. Queste stringhe sono utilizzate frequentemente dagli utenti.
2. **Riutilizzo della password**. Una serie di interviste per indagare su come gli utenti riescono a gestire i diversi account [10] sottolineano che gli utenti hanno in media più di 20 account. Con l'aumento del numero di account per utente, è più probabile che gli utenti riutilizzino le proprie password o le cambino leggermente per evitare di memorizzarne delle nuove. Quindi riutilizzare le password è un approccio razionale. Allo stesso tempo, il riutilizzo delle password è un comportamento vulnerabile, la chiave è come riutilizzarle. È particolarmente vero quando una politica di sicurezza obbliga gli utenti a modificare frequentemente le password. La ricerca [11] condotta su 470 studenti universitari, personale e docenti ha rivelato che il 60% degli individui ha utilizzato una password con lievi modifiche per account diversi. Ad attestare questi dati un altro studio su set di password trapelati di Durmuth [12] hanno dimostrato che gli utenti spesso applicano semplici trucchi per apportare lievi modifiche alle loro vecchie password. Casi tipici sono:
    a. **Inserimento**: inserire una stringa tra due parole in una parola. Ad esempio, per una password contenente computer, un possibile inserimento è computer1.
    b. **Trasposizione di componenti**: cambiare l'ordine delle parole. Ad esempio, chebelcastello può essere modificato in belcastelloche.
    c. **Eliminazione**: elimina una parola dalla password. Ad esempio, data la password chebelcastello, è possibile creare altre strutture base come belcastello.

3. **Password contenenti informazioni personali**: studi di ingegneria sociale hanno evidenziato che gli utenti di etnia cinese tendono a costruire password numeri rilevanti: come numero di telefono e data di nascita. Gli utenti europei tendono a utilizzare nickname o cose relative ai loro interessi come la loro squadra di calcio. Gli studi hanno rivelato una nuova visione: le lingue native degli utenti influenzano le loro password e in che misura le informazioni personali degli utenti svolgono un ruolo nelle loro password [13] [14] [15].

Svariati utenti hanno una procedura che usano sempre per creare una password. Quindi usano i concetti appena visti per poter accedere ai servizi internet. Inconsapevolmente, alcune password che ritenevano forti e difficili da risalire erano il contrario e facilitavano il compito dell'attaccante che ha diverse armi per poter svolgere il suo intento [16].

1.2 Attacco Online

Quando si parla di attacchi online intendiamo tutte quelle azioni messe in atto dal soggetto attaccante per intercettare informazioni tra comunicazioni, identificarsi sotto mentite spoglie, modificare messaggi. Tutte queste attività avvengono senza aver nessun dato di supporto. Ad esempio un'offensiva ad un sistema connesso ad internet attraverso un pagina di login. Ci sono vari approcci:

- **Denial of Service** (DoS) e **Distributed Denial of Service** (DDoS): un attacco DoS sovra carica le risorse del sistema in modo che non possa rispondere alle richieste di servizio. Un attacco DDoS è come il prcedente ma eseguito da più macchine.
- **Man in the Middle** (MitM): un attacco in cui l'attaccante si inserisce tra le comunicazioni di un client e un server.
- **Phishing** e **spear phishing**: il phishing è la pratica di mandare e-mail che sembrano provenire da fonti affidabili con l'obiettivo di sottrarre informazioni. Lo spear phishing è una tecnica di phishing che è più mirata sulla vittima e cerca di inviare e-mail personali.
- **Drive-by**: l'attaccante inserisce in siti degli allegati che, una volta scaricati installino il contenuto e spesso sono dei malware.
- **Attacco sulle password**: l'attaccante prova ad autenticarsi per un'altra persona.

- **SQL injection**: l'attaccante cerca di sottrarre informazioni al database attraverso l'interrogazione di quest'ultimo, ad esempio utilizzando il campo di login.
- **Cross site scripting** (XSS): gli attaccanti utilizzano risorse web di terze parti per eseguire script nel browser web o nell'applicazione della vittima.
- **Attacco birthday**: l'attaccante cerca di trovare un messaggio che produca lo stesso *message digest* della funzione hash utilizzata. Così la vittima riceverà un messaggio iniettato dall'attaccante senza rendersene conto.
- **Attacco malware**: è un software indesiderato che viene installato nel sistema della vittima senza il suo consenso.

Queste azioni sono tra le principali macro categorie di attacchi che sono sempre in evoluzione e in fase di crescita per sfruttare ogni minima debolezza, carenza, ingenuità del sistema e dell'utilizzatore [17] [18].

1.3 Attacco Offline

Questa tipologia di attacchi vengono effettuati una volta che si sono recuperate delle informazioni. Queste informazioni possono essere ottenute in modi diversi:

- **Compromissione del sistema**: nei moderni sistemi UNIX, le password sono memorizzate in modo crittografato nel file /etc/shadow, che è accessibile solo a chi ha privilegi amministrativi.
- **Compromissione del database di un applicativo**.
- **Con tool dedicati**: ad esempio Mimikatz che è uno strumento progettato per recuperare tutte le password gestite da Windows.
- **Trasmissione di Hash in rete per autenticazione**.
- **Intercettazione attraverso la rete Wi-Fi**.

La differenza dei due attacchi visti, online ed offline, si possono distinguere per la velocità di esecuzione e per le informazioni in possesso. Il primo è lento ed inefficiente, perché richiede la risposta online del sito attaccato. Inoltre molti siti fissano dei limiti ai tentativi di login errati per prevenire qualsiasi tipo di azioni malintenzionate. Gli attacchi offline, come detto in precedenza, possiedono estratti di database criptati con codifica Hash. Una volta ottenute le informazioni l'attaccante è "a metà dell'opera": avrà infatti a disposizione tutto il tempo per cercare di risalire dagli hash alle relative password. In

aggiunta gli attacchi offline sono invisibili al team di sicurezza perché non lasciano traccia sui log [19] [20].

1.4 Tecniche di Password Cracking

Il compromesso delle password è sempre una seria minaccia per la riservatezza e l'integrità dei dati da proteggere. In generale, le password più brevi di 7 caratteri sono particolarmente suscettibili agli attacchi bruteforce. Una buona pratica è quella di usare password con più di 8 caratteri, usare caratteri minuscoli e maiuscoli, numeri, caratteri speciali e non inserire informazioni personali.

1.4.1 Brute force attack

L'attacco a forza bruta consiste nel provare ogni singola possibile combinazione fino a che non si trova quella corretta. L'attaccante calcolerà l'hash della password che si tenta di penetrare e si confronta il risultato con l'hash di destinazione. In caso di successo l'attacco si ferma altrimenti il processo continua tentando una nuova password. Teoricamente è il metodo migliore, in quanto si ha la certezza di trovare tutte le password, però richiede molto tempo, se la password è lunga ci vorranno anni per la forza bruta. Il numero di tentativi richiesti per coprire un dato spazio chiave usando un attacco di pura forza bruta è uguale a $\sum^{n}_{k=0} x^{k}$ dove x è la dimensione del set di caratteri e n è la lunghezza massima delle password prese di mira. Questo tipo di attacco utilizza solitamente un array composto da lettere tutte maiuscole, tutte minuscole e tutte le cifre. Ipotizzando di avere una password la cui lunghezza è pari a 9 caratteri, con un array di soli caratteri minuscoli e di cifre, questo tipo di attacco impiegando un solo calcolatore con la velocità di 500.000 chiavi al secondo, ci impiegherà 6 anni. Ma se la password è breve, può dare risultati più celeri [21].

1.4.2 Dictionary attack

Un file di testo, il dizionario, è caricato all'interno del sistema. Questo file contiene un grande numero di parole, meglio se usiamo un elenco di parole personalizzato. Ogni parola è testata in modo da vedere se combacia con quella dell'utente. I dizionari sono composti da milioni di password, la raccolta più nota è quella nota come rockyou, ma ne esistono diverse originate dai vari leak dei siti. L'attacco al dizionario è utilizzato principalmente in due situazioni:

- **In criptoanalisi**: per trovare la chiave di decrittazione di un testo cifrato;
- **In sicurezza informatica**: per avere accesso in un computer protetto da password.

1.4.3 Rule base attack

Questo attacco è utilizzato quando conosciamo e abbiamo informazioni in merito alla password. La difficoltà di questo approccio è quella di creare regole per uno specifico attacco: bisogna saper utilizzare questo particolare "linguaggio di programmazione" per portare un attacco soddisfacente. Infatti esiste un linguaggio specifico per impartire degli ordini al programma di cracking. Le rules scelte andranno poi inserite in un file di testo, scrivendo una regola per riga di testo. Si basa sul fatto che le persone tendono a riutilizzare le stesse password, modificandole semplicemente di qualche carattere, nell'illusione di renderle più forti pur mantenendo facile la memorizzazione. Sono le cosiddette "modificazioni ovvie delle password". Un esempio fatto con la parola "password" è il seguente: p@ssword, passw0rd, p@$$w0rd, ect. Un altro esempio, se si sa che la password contiene 2 o 3 numeri specifici, possiamo utilizzarli per creare un file ad-hoc e trovarla in meno tempo. Oppure se sappiamo che la password è "castello" ma con numeri aggiunti, creiamo un file con castello1, castello2, etc. e ripetiamo l'attacco.

In questa categoria rientra anche l'attacco al pattern: ci si concentra solo su pattern reali, che sappiamo essere quelli che le persone tendono ad usare. Quando viene richiesta una password contenente almeno una maiuscola, un numero ed un carattere speciale, il pattern statisticamente più utilizzato dagli utenti è: maiuscola all'inizio, numeri e caratteri speciali in fondo.

1.4.4 Mask attack

Quando si conosce parte di una password o dettagli su essa, si possono sfruttare per ridurre la complessità della ricerca. Sapendo la lunghezza della chiave da indovinare, visibile dai classici puntini neri, e sapendo parte della password, si può fare un attacco che migliora il difetto dell'attacco bruteforce: il numero di tentativi da effettuare. Con uno strumento come Hashcat si può creare una password che rispetti le regole richieste. Scrivendo attraverso la codifica Hashcat:

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789

- ?h = 0123456789abcdef
- ?H = 0123456789ABCDEF
- ?s = «space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
- ?a = ?l?u?d?s
- ?b = 0x00 - 0xff

?u?l?l?l007

Hashcat andrà a provare tutte le combinazioni partendo da Aaaa007. Quindi l'utilizzo di questo attacco va a migliorare l'attacco di forza bruta riducendo lo spazio di ricerca [22] [23].

1.4.5 Shoulder surfing

È una tecnica molto simile all'ingegneria sociale usata per ottenere informazioni e dati confidenziali osservando la vittima standole alle spalle. All'attaccante, usando questa tecnica, non è richiesta nessuna abilità particolare, l'attenta osservazione di ciò che sta intorno alla vittima e dei movimenti effettuati nel momento di inserimento della informazioni, sia in presenza che da remoto. È diffuso in posti affollati [24] [25].

1.4.6 Social engineering

È lo studio del comportamento individuale di una persona al fine di carpire informazioni utili. È un insieme di tecniche a metà tra psicologia e ingegneria. Può essere considerata come una manipolazione psicologica che induce la vittima a comportarsi un determinato modo. L'ingegneria sociale fa leva su tratti caratteristici dell'essere umano, come la disponibilità e la buona fede dell'attaccato, l'ignoranza e la disattenzione ed oggigiorno anche dei social media. L'attacco avviene in diverse fasi. L'attaccante studia i comportamenti, abitudini e preferenze della vittima, così da poter entrare in contatto con lui. A seconda della vittima si possono adottare diverse tecniche come quella della paura, del senso di colpa o dell'autorevolezza [26].

Capitolo 2

2 Password cracking

Con il diffondersi delle conoscenze e la consapevolezza informatica si è capito che la password ha un ruolo fondamentale nel sistema di autenticazione. Le motivazioni di un cracking sono svariate, come ad esempio ottenere un guadagno economico, perorare una causa, vincere una sfida o il semplice desiderio di danneggiare gli altri. Pertanto, l'origine del cracking può essere estremamente diversificata. Tuttavia, il cracking, a differenza dell'hacking, si affida a una ripetizione persistente di svariate tecniche al fine di violare un sistema, senza pertanto ricorrere all'abile sfruttamento delle debolezze dello stesso. Ora verranno mostrate le principali tecniche che sono state usate per praticare gli attacchi offline eseguiti. Ovviamente, visto l'importanza di questo argomento si sono studiate diverse tecniche che riescano a rendere più efficiente e velocizzare il compito dell'attaccante. Alcune di queste tecniche sono influenzate da vari fattori come ad esempio il dataset di input.

2.1 Rainbow table

Un metodo popolare che le persone hanno usato per mantenere le password più sicure sono le funzioni di hash crittografiche, come ad esempio MD5, SHA-1. Invece di memorizzare la password nel database in formato testo normale, vengono memorizzati i loro valori hash. Tuttavia, il problema con queste funzioni è che sono state progettate per essere molto veloci: il loro utilizzo principale è quello di garantire l'integrità dei messaggi. La solidità di MD5 e SHA-1 è considerata la principale nemesi delle password, perché consente di eseguire un attacco alla rainbow table. Oggigiorno grazie allo sviluppo tecnologico, si ha la possibilità di utilizzare delle GPU e quindi aumentare la potenza di calcolo fino a cento milioni di hash MD5 al secondo [27]. Ciò implica che una password composte da pochi caratteri, tendenzialmente meno di 7 sono considerate deboli.

L'idea di base è stata proposta per la prima volta da Hellman in cui le catene di hash delle password venivano archiviate e quindi ricreate per attaccare gli hash delle password, che fondò la sua teoria su una tecnica chiamata compromesso tempo-memoria [28], successivamente modificate da Oechslin. L'idea è stata ulteriormente affinata da Oechslin che ha introdotto una diversa funzione di riduzione per ogni passaggio della catena di

hash e ha rimosso l'idea dei punti distintivi. Un'implementazione popolare dell'attacco di Oechslin, Rainbowcrack, è stata successivamente codificata e rilasciata da Shuanglei.

L'obiettivo è precalcolare una struttura dati che, dato qualsiasi output h della funzione di hash, possa individuare un elemento p in P tale che H(p) = h, o determinare che non esiste tale p in P. Il modo più semplice per farlo è calcolare H(p) per tutti i p in P, ma la memorizzazione della tabella richiede $\Theta(|P|n)$ bit di spazio, dove |P| è la dimensione dell'insieme P e n è la dimensione di un output di H, che è proibitivo per grandi |P|. Le catene di hash sono una tecnica per ridurre questo requisito di spazio. L'idea è di definire una funzione di riduzione R che riconduca i valori hash ai valori in P. Tuttavia la funzione di riduzione non è in realtà un inverso della funzione hash, ma piuttosto una funzione diversa con un dominio e un codominio scambiati della funzione hash. Alternando la funzione hash con la funzione di riduzione, si formano catene di password e valori hash alternati. Le dimensioni regolari di una table contenente gli hash di tutte le possibili password composte da 8 simboli (lettere/numeri/caratteri speciali) possono essere grandi 160 GB [29]. Quindi le rainbow table creano un ampio database contenente valori hash precalcolati di possibili modelli di testo in chiaro [30]. Questo significa che se un avversario entra in possesso di un database di password che contiene i valori hash delle password, l'avversario dovrà solo confrontare quei valori hash con quelli nella rainbow table calcolata per conoscere il testo in chiaro corrispondente. Quindi mantenere le password utilizzando solo funzioni di hash crittografiche non è sufficiente.

Un concetto importante delle tabelle arcobaleno è l'idea di un valore di indice. Il valore dell'indice varia da 0 a (key-space max -1). Per dimostrare questo, se l'attaccante stesse cercando di forzare tutte le parole lunghe sei caratteri che contengono solo lettere minuscole, lo spazio massimo della chiave sarebbe 2^{66}. Questo significa che ogni possibile valore della variabile indice rappresenta un'ipotesi di password univoca. Pertanto, qualsiasi attacco che coinvolga tabelle arcobaleno deve avere una funzione di indicizzazione rapida. Ci sono tre funzioni principali utilizzate nella creazione e nell'applicazione delle tabelle arcobaleno e sono cicliche. Queste funzioni sono:

- **IndexToPlain**: accetta un valore di indice di input e restituisce l'ipotesi di password in testo normale corrispondente.
- **PlainToHash**: accetta un'ipotesi di password in testo normale di input e ne restituisce il valore con hash.

- **HashToIndex**: accetta un hash come input e utilizza una funzione di riduzione per restituire un valore di indice.

Le tabelle Rainbow si basano sull'idea delle catene di hash. Quando viene creata una tabella arcobaleno, l'utente specifica la lunghezza della catena e il numero di catene. Il valore della lunghezza della catena può essere considerato come la quantità di compressione da utilizzare. Più lunga è la catena, più hash possono essere archiviati nella stessa quantità di spazio su disco. Lo svantaggio è che una catena più lunga richiede più tempo per la ricerca durante un attacco ed è più soggetta a collisioni. Il numero di catene determina quante catene sono memorizzate e quindi lo spazio su disco occupato dalla tabella arcobaleno. Quando viene creata una catena di hash di tabella arcobaleno tradizionale, viene selezionato casualmente un valore di indice iniziale. Viene quindi eseguito IndexToPlain per determinare l'ipotesi della password in testo normale. Questa ipotesi viene quindi eseguita attraverso PlainToHash dove viene hash. Infine HashToIndex viene eseguito per determinare il valore dell'indice successivo. Questo processo, utilizzando il valore dell'indice precedente come input, viene ripetuto n volte dove n è uguale alla lunghezza della catena. Un esempio è visibile in Figura 1.

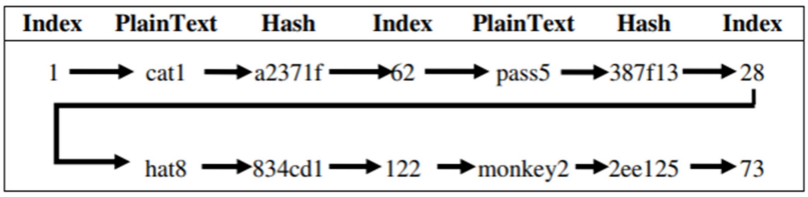

Figura 1. Creazione di una Rainbow table di lunga quattro.

Le tabelle Rainbow ottengono la loro compressione memorizzando solo il primo e l'ultimo valore di indice per ogni catena. Significa che mentre la singola catena stessa può rappresentare diverse migliaia di hash, vengono memorizzati solo due valori. Poiché il processo per creare una catena è deterministico, l'intera catena può essere ricreata dal primo valore di indice. L'ultimo valore dell'indice viene salvato in modo che l'hash di destinazione, l'hash che l'attaccante sta tentando di violare, può essere identificato se esiste in quella catena. Quando tenta di decifrare un hash della password utilizzando una tabella arcobaleno, l'attaccante prende l'hash della password di destinazione e quindi esegue HashToIndex su di esso. Quindi confrontano il valore dell'indice generato con l'ultimo valore dell'indice di ogni catena nella tabella arcobaleno. Se il valore dell'indice dell'hash della password corrisponde a una delle catene, la catena viene quindi ricreata dal suo primo valore dell'indice. Se l'hash target esiste nella catena, l'attaccante torna

semplicemente indietro di un passo per vedere il testo in chiaro che lo ha creato. Nella maggior parte dei casi, questo è più complicato poiché la maggior parte delle tabelle arcobaleno create utilizza una funzione di riduzione diversa a seconda di dove ti trovi nel processo di creazione della catena. Come affermato in precedenza, la funzione di riduzione viene utilizzata da HashToIndex per determinare a quale valore di indice successivo viene assegnato un valore hash. Ciò significa che lo stesso valore hash può generare molti valori di indice diversi a seconda della fase in cui si trova nel processo di creazione della catena. Per questo motivo, deve essere creata una nuova catena di destinazione per ogni possibile posizione in cui il valore hash di destinazione può verificarsi nella catena arcobaleno.

Uno dei problemi principali che affligge la creazione e l'uso delle tabelle arcobaleno è la fusione delle catene. Si dice che una catena si unisca quando due o più catene con valori di indice iniziali diversi condividono lo stesso ultimo valore di indice. Ciò può essere causato da una collisione generata da una delle tre principali funzioni della tabella arcobaleno: IndexToPlain, PlainToHash o HashToIndex. Una collisione si verifica quando una delle funzioni accetta input diversi ma restituisce lo stesso output. Le catene unite sono problematiche poiché tutto il lavoro svolto nella catena dopo il punto in cui si è verificata la collisione viene duplicato. Inoltre, se si verifica una collisione tra una delle catene di destinazione generate per controllare una password e una catena nella tabella arcobaleno, può creare un falso avviso, altrimenti noto come falso positivo. Quando ciò si verifica, la catena nella tabella arcobaleno viene ricreata anche se l'hash di destinazione non è presente. Pertanto, questi falsi avvisi possono aumentare drasticamente il tempo necessario per decifrare un hash di password. A seconda del tipo di hash preso di mira, la funzione PlainToHash è generalmente estremamente resistente alle collisioni; mentre è stato dimostrato che hash come MD5 sono vulnerabili agli attacchi di compleanno [36], la probabilità che una tale collisione si verifichi durante la creazione della tabella arcobaleno è limitata. Anche se si verifica una collisione da questa funzione, è abbastanza raro da non influenzare negativamente le prestazioni della tabella arcobaleno nel suo insieme. Con hash più deboli come MYSQL323, che è lungo solo 64 bit, le collisioni si verificano molto più spesso. Se si verifica la collisione per l'hash di destinazione, la tabella arcobaleno restituisce una password valida per l'utente anche se la password "craccata" è diversa dalla password originariamente selezionata dall'utente. Sfortunatamente, la stragrande maggioranza delle collisioni sarà generata da hash di password che non corrispondono all'hash di destinazione, il che si traduce in lavoro sprecato e catene di

fusione. Tradizionalmente anche la funzione IndexToPlain è stata resistente, se non del tutto priva di collisioni. Ciò è dovuto alla natura degli attacchi di forza bruta che sono stati impiegati. Quando si generano ipotesi, (come funzione di indicizzazione), con un approccio di forza bruta è abbastanza banale evitare ipotesi duplicate e quindi collisioni dalla funzione di indicizzazione. Tuttavia, come vedremo più avanti, quando si applica un approccio basato su dizionario, è necessario prestare particolare attenzione per evitare parole del dizionario duplicate e regole che generano ipotesi duplicate. Per questo motivo, è importante essere consapevoli che le tabelle arcobaleno basate su dizionario richiedono molta più pianificazione nella loro costruzione e che l'attaccante potrebbe dover accettare una quantità maggiore di collisioni nella propria tabella rispetto a se utilizzasse un approccio a forza bruta. Infine, la funzione HashToIndex è il punto in cui si verificano la maggior parte delle collisioni. Questo è dovuto alla natura della costruzione della tabella arcobaleno poiché l'obiettivo nella creazione di una tabella arcobaleno è rappresentare il maggior numero possibile di ipotesi in testo semplice e quindi il maggior numero possibile di valori di indice dallo spazio chiave. A causa della natura probabilistica della creazione di catene (ovvero non hai idea di quale sia il valore dell'indice del passaggio successivo finché non lo crei), un utente malintenzionato non ha modo di evitare ipotesi duplicate generate dalla funzione HashToIndex. Per combattere ciò, la funzione HashToIndex normalmente utilizza una funzione di riduzione diversa a seconda di quale passaggio si verifica nella catena. In questo modo, anche se possono essere generate ipotesi duplicate, a meno che tali ipotesi duplicate non si verifichino nello stesso passaggio nella catena arcobaleno, le catene stesse non si uniscono, limitando il danno causato dalle collisioni dalla funzione HashToIndex.

Uno strumento per creare le strutture dati e svolgere questa tecnica si può utilizzare RainbowCrack [31]. La sintassi del comando per generare le tabelle è il seguente:

rtgen hash_algorithm charset plaintext_len_min plaintext_len_max table_index chain_len chain_num part_index

Una volta generata la tabella, per utilizzarla dobbiamo prima caricarla, eseguendo il comando :

root@kali:~# sudo rtsort

Dopo aver caricato la tabella possiamo passare ad eseguire l'attacco. Per eseguirlo possiamo passare

1. un hash: *root@kali:~# sudo rcrack -h hash_da_attaccare*
2. un file: *root@kali:~# sudo rcrack -l path_file_da_attaccare*

Per evitare di cadere vittime di un attacco Rainbow Table, oltre a seguire tutte le buone pratiche durante la creazione di una password, questo tipo di attacchi possono essere prevenuti utilizzando la tecnica del salt da parte dello sviluppatore del sistema IT. Il salt è un bit casuale di dati che viene passato nella funzione hash insieme al testo in chiaro. Ciò garantisce che ogni password abbia un hash generato univoco, rendendo impossibile eseguire questo tipo di attacco.

2.2 Markov Chains Attack

Le Catene di Markov sono una successione di variabili aleatorie definite su un insieme finito di stati, ad ogni passo della successione sarò in uno degli stati definiti. Le catene di Markov quindi descrivono un processo stocastico particolare, che si presta per la creazione di modelli di sistemi che hanno un comportamento casuale nella loro evoluzione, descrivono bene fenomeni casuali che evolvono in funzione del tempo e che non hanno memoria degli stati precedenti. Questo concetto fu ideato e studiato dal matematico russo Andrej Markov, e trova applicazioni in molti campi.

A questo punto si devono definire alcuni concetti chiave per una maggiore comprensione:

- **Successione** o **sequenza**: possono essere definite come un elenco ordinato costituito da un'infinità numerabile di oggetti, detti termini della successione, tra i quali sia possibile distinguere l'ordine.
- **Spazio di probabilità**: uno spazio di probabilità è una terna (Ω, A, P), dove Ω è un insieme, A è una famiglia coerente di eventi su Ω e P una probabilità su A.
- **Variabile aleatoria**: data la terna (Ω, A, P), una funzione $X : \Omega \rightarrow R$ è una variabile aleatoria se

  $$\omega \in \Omega : X(\omega) \leq a$$

  è un evento per ogni $a \in R$. Una variabile aleatoria è dunque un numero che viene assegnato, mediante una determinata regola, a ciascun punto dello spazio di probabilità.
- **Matrice di Transizione di Markov**: Una Matrice di Transizione è la matrice generata dalla probabilità di transizione da uno stato all'altro del nostro sistema. Per costruzione la matrice dovrà avere valori compresi tra 0 e 1. In aggiunta la somma di ogni riga deve essere 1. La i-esima riga e colonna rappresentano lo stato

del sistema alla i-esima transazione. Perciò se prendiamo l'elemento della seconda riga e terza colonna ci mostrerà la probabilità di passare dal secondo al terzo stato.

$$\begin{bmatrix} 0.3 & 0.4 & 0.3 \\ 0.4 & 0.0 & 0.6 \\ 0.2 & 0.6 & 0.2 \end{bmatrix}$$

Un modello di Markov di ordine zero è l'equivalente dell'utilizzo dell'analisi della frequenza delle lettere poiché non ha memoria. Con un modello di Markov del primo ordine, la probabilità di ogni carattere dopo il primo è data come $P(Q_t=S_i \mid Q_{t-1}=S_j)$ dove i e j rappresentano l'indice del set di caratteri utilizzato. Questo concetto viene sfruttato per il password cracking basandosi sul fatto che le lettere adiacenti nelle password generate dall'uomo non sono scelte indipendentemente, ma seguono determinate regole: ad esempio, nella lingua inglese se appare la lettera "q", è quasi sempre seguita da una "u". In un modello di Markov a n-grammi (sequenze di n parole presenti in un testo), si modella la probabilità del carattere successivo in una stringa in base a un prefisso di lunghezza n-1. Quindi, per una data stringa $c_1, \ldots, c_m$, un modello di Markov stima la sua probabilità come:

$$P(c_1, \ldots, c_m) \approx P(c_1, \ldots, c_{n-1}) \cdot \Pi^m_{i=n} P(c_i|c_{i-n+1}, \ldots, c_{i-1})$$

Per il cracking delle password, si imparano sostanzialmente le probabilità iniziali $P(c_1, \ldots, c_{n-1})$ e le probabilità di transizione $P(c_n \mid c_1, \ldots, c_{n-1})$ dai dati del mondo reale, i quali dovrebbero essere più vicino possibile alla distribuzione che ci aspettiamo nei dati che attacchiamo, quindi si enumerano le password in ordine di probabilità decrescente come stimato dal modello di Markov. Per rendere efficiente questo attacco, si devono considerare una serie di dettagli: i dati limitati rendono difficile l'apprendimento di queste probabilità e l'enumerazione delle password nell'ordine ottimale è difficile. La tecnica della catena di Markov produce prestazioni desiderabili solo a un basso livello di rumore. Quindi per rendere il più efficiente possibile un modello di Markov è importante avere un dataset con basso rumore e di grandi dimensioni. L'assenza di una sorgente dati significativa comporterebbe infatti la bassa precisione nel calcolo delle informazioni vitali dell'algoritmo, come la corretta frequenza delle parole. Altri aspetti da non tralasciare sono l'uso per salvare gli stati e quello di adottare un algoritmo di ordinamento delle password nell'ordine corretto [32] [33] [34].

2.3 PCFG attack

Le Probablistic Context Free Grammar Attack hanno come assunzione di base che le strutture delle password hanno probabilità diverse, cioè non tutte le password generate hanno la stessa probabilità di recuperare una password. Alla base dell'attacco troviamo la generazione di possibili password in ordine decrescente di probabilità per massimizzare la verosimiglianza con la password di destinazione, in modo simile a quello visto con le Markov chains. Le Context-Free Grammar (CFG) sono state a lungo utilizzate nello studio dei linguaggi naturali, dove vengono utilizzate per generare ed analizzare stringhe. Il punto debole di questo attacco è che la generazione di ipotesi in ordine di probabilità è lenta, il che significa che crea in media 50-100k ipotesi al secondo, in cui gli algoritmi basati su GPU possono creare da milioni a miliardi di ipotesi al secondo contro algoritmi di hashing veloci. Pertanto, PCFG viene utilizzato al meglio contro un gran numero di hash salati o altri algoritmi di hashing lenti, in cui il costo delle prestazioni dell'algoritmo è compensato dall'accuratezza delle ipotesi [35].

Una grammatica context-free è definita come G = (V, Σ, S, P), dove:

- V è un insieme finito di variabili (o non terminali),
- Σ è un insieme finito di terminali,
- S è la variabile iniziale,
- P è un insieme finito di produzioni della forma:

$$\alpha \rightarrow \beta$$

dove α è una singola variabile e β è una stringa composta da variabili o terminali. Il linguaggio della grammatica è l'insieme di stringhe costituito da tutti i terminali derivabili dal simbolo di inizio. Le grammatiche probabilistiche context-free hanno semplicemente probabilità associate a ciascuna produzione tali che per una specifica variabile del lato sinistro (LHS) tutte le produzioni associate sommate hanno valore 1. Fornendo come input un set di produzioni e probabilità associate automaticamente da training set di password reali. Queste produzioni tentano di catturare le caratteristiche di come le persone creano le password, insieme alla frequenza con cui vengono utilizzate determinate regole di manipolazione delle parole.

Per semplicità questa sezione illustra solo grammatiche che utilizzano il simbolo di inizio e le variabili della forma Ln, Dn e Sn, per valori specificati di n. Chiamiamo queste variabili variabili alfa, variabili numeriche e variabili speciali rispettivamente. Si noti che la riscrittura delle variabili alfa viene eseguita utilizzando un dizionario di input simile a quello utilizzato in un attacco di dizionario tradizionale. Per questo motivo, la riscrittura completa per le variabili alfa non viene mostrata in molti degli esempi seguenti a causa delle grandi dimensioni dei dizionari di input tradizionalmente utilizzati in un attacco di violazione delle password. Va notato che il pedice *n* nelle nostre variabili, come D_n, è usato per indicare la lunghezza della variabile di sostituzione. Quindi un D_2 rappresenta una variabile che verrebbe riscritta come un numero a due cifre. Significa che le probabilità associate alla riscrittura di un D_1D_1 possono essere molto diverse da quelle associate a un D2, anche se producessero le stesse ipotesi. È dovuto alla probabilità condizionale associata a determinate stringhe. Ad esempio, il numero a due cifre "86" può avere una probabilità molto alta perché è l'anno di nascita di qualcuno, ma i numeri "8" e "6" possono avere una probabilità molto bassa di essere selezionati indipendentemente. La chiave per evitare ipotesi duplicate e quindi assicurarsi che nessuna variabile dello stesso tipo appaia una accanto all'altra.

Ai fini dell'ottimizzazione, è utile introdurre la nozione di contenitori. Un contenitore è una struttura che ci permette di ottimizzare i calcoli relativi a terminali di tipo simile che hanno tutti probabilità identiche. Usiamo il termine pseudo-terminale per indicare la rappresentazione di un contenitore nella nostra grammatica. Gli pseudo-terminali sono una comodità di implementazione e non devono effettivamente apparire nella grammatica formale. Ad esempio, attualmente assegniamo a tutte le parole del dizionario di lunghezza n dello stesso dizionario di input la stessa probabilità $1/k_n$, dove k_n è il numero di parole di lunghezza n nel dizionario di input. Quindi abbiamo un minuscolo pseudo-terminale associato L in k, lo chiamiamo l_k, con una produzione associata:

$$L_k \rightarrow l_k \text{ (con probabilità } 1/n_k)$$

Questo significa che la grammatica vede tutte le parole terminali di lunghezza n come nello stesso contenitore. Sebbene non sia necessario per la teoria del funzionamento del nostro password cracker probabilistico, in pratica i contenitori hanno un grande impatto sull'operazione efficiente della funzione successiva. Proprio come la nozione di contenitori, ci sono diverse fasi dall'elaborazione della variabile iniziale S alla fase finale

in cui tutte le variabili sono state riscritte come valore terminale, dove abbiamo trovato utile definire termini aggiuntivi che ci consentono di ottimizzare al meglio algoritmo di generazione.

La struttura semplice è la prima regola di riscrittura dopo il simbolo di inizio S, ma non contiene informazioni sulla lunghezza. Tali informazioni non vengono catturate nella struttura semplice ed è difficile da reintrodurre quando si utilizza una grammatica priva di contesto. Per questo motivo, si sono utilizzate strutture di base come primo set di regole di riscrittura dopo il simbolo di inizio S. Le strutture di base sono esattamente come le strutture semplici, tranne per il fatto che contengono informazioni sulla lunghezza per le seguenti regole di riscrittura. In questo modo possiamo catturare ciò che essenzialmente sono informazioni sensibili al contesto mentre si utilizza una grammatica libera dal contesto. Le forme sentenziali che contengono solo pseudo-terminali o terminali sono chiamate strutture pre-terminali. Queste strutture pre-terminali sono molto importanti perché la probabilità associata a una struttura pre-terminale non cambia più con eventuali sostituzioni aggiuntive, e quindi è la stessa probabilità dell'ipotesi terminale finale. Questo ci consente di lavorare con strutture pre-terminali quando si generano ipotesi di password in ordine di probabilità, invece di dover compilare tutte le variabili con valori terminali. Poiché una singola struttura pre-terminale può spesso generare migliaia di valori terminali univoci, questo è essenziale per avere un algoritmo efficiente.

Quindi le PCFG (Probabilistic CFG) sono dei semplici CFG ma con associato ad ogni produzione una probabilità in modo tale che la somma di tutte le probabilità sia 1.

Per capire meglio questo concetto ecco alcuni termini che sono usati:

- **L**: si indicano le Alphabet String ovvero una sequenza di simboli alfabetici.
- **D**: si indicano le Digit String ovvero una sequenza di cifre.
- **S**: si indicano le Special String ovvero una sequenza né di simboli alfabetici né cifre.

Ad esempio la parola "ca$tello39" sarà identificata dalla struttura semplice LSLD. In aggiunta viene assegnata ad ogni sequenza della struttura la propria lunghezza, così il risultato sarebbe: $L_2S_1L_3D_3$ che è chiamata struttura base.

Tipi di dati	Simboli	Esempi
Alpha String	abcdefghijklmnopqrstuvwxyzäö	Cane

Digit String	0123456789	432
Special String	!@#$%^&*()-_=+[]{};“:”,./<>?	!!

Tabella 1. Lista delle diverse sequenze con esempio.

Ora che abbiamo definito le stringhe alfa, cifre e speciali, il passo successivo è catturare la loro probabilità di occorrenza. Viene conteggiata ogni stringa di lunghezza n e ad ogni stringa viene assegnata una probabilità. Se non viene applicato alcun livellamento di probabilità, la probabilità di una stringa x viene assegnata utilizzando la seguente formula:

$$\text{Probabilità}(x)=x_n/k_n$$

dove xn è il numero di occorrenze della stringa x nel training set e kn è il numero di occorrenze di stringhe di quel tipo, di lunghezza n che si sono verificate nel training set. È importante notare che una stringa è definita dalla sua lunghezza massima di valori consecutivi dello stesso tipo. La ragione di ciò è che analizzando solo stringhe della stessa lunghezza, ci consente di salvare la probabilità condizionata che i singoli caratteri appaiano insieme. Ad esempio la probabilità che appaia '79' potrebbe essere alta perché è un anno di nascita, anche se le probabilità individuali di '7' e '9' potrebbero essere piuttosto basse.

Definiti questi termini, ecco il funzionamento del PCFG implementato in un attacco ad una password. La fase iniziale, quella di training, verrà analizzato un dataset più simile possibile alle password da attaccare. In questa fase si derivano un set di regole di produzione che generi la struttura base delle parole e un set di produzione che produca simboli terminali costituiti da cifre e caratteri speciali. Un altro approccio è quello di ricavare la probabilità dei caratteri alfabetici in base alla ricorrenza di questi, suddivisi per lunghezza della parola stessa: in pratica si ricava un set di regole anche per le variabili L. Una stringa derivata dal simbolo di partenza è definito come sentential form/forma sentenziale. La probabilità di una forma sentenziale è il prodotto delle probabilità delle produzioni usate nella sua derivazione. La stima di massima della forma sentenziale derivata prima di associarle una stringa alfabetica viene chiamata pre-terminal structure/strutture pre-terminali. Molti framework che utilizzano PCFG (o comunque la struttura delle password simile ad esso) oltre alle variabili L, S e D modellano altri aspetti del linguaggio (date, lettere maiuscole, ecc.) utilizzando ulteriori variabili [36].

Tipo di struttura	Esempio

Semplice	LSLD
Base	$L_2S_1L_4D_2$
Pre-Terminale	$L_2\$L_439$
Password generata	ca$tello39

Tabella 2: Diverse strutture in fasi diverse del processo di Password Generation con esempio.

Nella fase di training si ottiene direttamente un PCFG dal training set. Un esempio di un Probabilistic Context-Free Grammar è di seguito:

Lato sinistro	Lato Destro	Probabilità
$S \rightarrow$	$D_1L_3S_2D_1$	0.80
$S \rightarrow$	$L_3S_2D_1$	0.20
$D_1 \rightarrow$	4	0.60
$D_1 \rightarrow$	5	0.20
$D_1 \rightarrow$	6	0.20
$S_1 \rightarrow$	!	0.70
$S_1 \rightarrow$	#	0.20
$S_1 \rightarrow$	%	0.10
$S_2 \rightarrow$	$$	0.60
$S_2 \rightarrow$	**	0.40

Tabella 3: Esempio di un semplice PCFG.

Con una grammatica simile a quella presentata poco prima, è possibile generare, ad esempio, una struttura pre-terminale del tipo:

$$S \rightarrow L_3D_1S_1 \rightarrow L_34S_1 \rightarrow L_34!$$

Con una probabilità di $0.20 \rightarrow 0.20 * 0.60 \rightarrow 0.25 * 0.60 * 0.70 \rightarrow 0.084$.

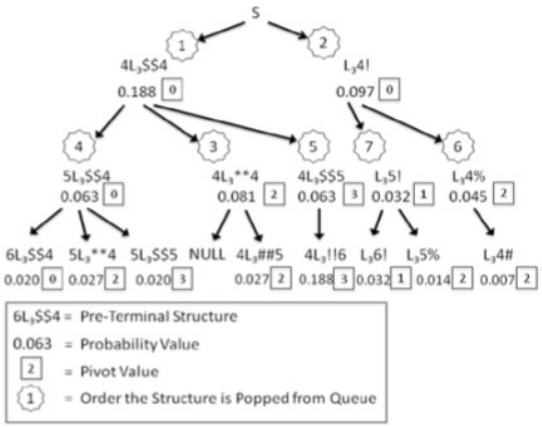

Figura 2. Generazione della struttura in base alle probabilità.

Una volta generato il nostro PCFG si passa alla fase di generazione password, che saranno utilizzate per attaccare il set di password interessato.

2.4 Prince attack

PRINCE, Probability INfinite Chained Elements, è un generatore di password guess e può essere pensato come un attacco Combinator avanzato. Questo tipo di attacco è basato sull'algoritmo PRINCE sviluppato da Jens Steube [37]. Invece di prendere come input due diversi dizionari e quindi emettere tutte le possibili combinazioni di due parole, PRINCE ha solo un dizionario di input e costruisce "catene" di parole combinate. Questo strumento non esegue il rilevamento dei duplicati durante il caricamento di un file. Dopo che PRINCE legge nel dizionario di input, memorizza ogni parola, (elemento), in una tabella composta da tutte le parole della stessa lunghezza. PRINCE costruisce quindi catene composte da 1 a N elementi diversi. Queste catene possono avere da 1 a N parole dal dizionario di input concatenate insieme. Questo numero è indicato come lo spazio delle chiavi della catena. Quindi, ad esempio, se emette ipotesi di lunghezza quattro, potrebbe generarle utilizzando combinazioni dal dizionario di input come:

- Parola di 4 lettere;
- Parola di 2 lettere + parola di 2 lettere;
- Parola di 1 lettera + parola di 3 lettere;
- Parola di 1 lettera + parola di 1 lettera + parola di 2 lettere;
- Parola di 1 lettera + parola di 2 lettere + parola di 1 lettera;
- Parola di 1 lettera + parola di 1 lettera + parola di 1 lettera + parola di 1 lettera.

Un effetto negativo di questo algoritmo, come suggerisce l'acronimo, è che creerà infinite catene di elementi, influendo negativamente sulla memoria e sul tempo. Come indicato nella descrizione, PRINCE combina le parole del dizionario di input per produrre ipotesi di password. Una volta che PRINCE ha prodotto le password della lunghezza x, ordina le diverse catene (di lunghezza complessiva x) in maniera decrescente in base alla dimensione dello spazio delle chiavi e le esaurisce applicando diversi attacchi:

- Attacco a dizionario
- Keyboard walks
- Brute Force

Si può evidenziare quindi che se si vuol creare una password di lunghezza 3, le catene totali che verranno create sono:

$$2^{\text{lunghezza-1}}$$

In questo esempio sarebbero:

1. 1+1+1;
2. 2+1;
3. 1+2;
4. 3.

Si ipotizzi che un dizionario ha a disposizione 20 parole di lunghezza 3, 5 parole di lunghezza 2 e 3 parole di lunghezza 1. Prince terminerebbe le catene nell'ordine indicato nella tabella riassuntiva (Tabella 4), iniziando con le parole di lunghezza 2 concatenate alle parole di lunghezza 1, ed infine combinando le parole di un carattere.

Catena	Elemento	Spazio delle chiavi ↑
1+2	3*5	15
2+1	5*3	15
3	20	20
1+1+1	3*3*3	27

Tabella 4. Esempio di una tabella per password di lunghezza 3.

La domanda principale ovviamente è come si inserisce questo strumento in una sessione di cracking? Sarebbe meglio che ad uno strumento come questo gli venga passato un dataset che contenga liste di parole plausibili. Strumenti utili che possono essere usati per

creare queste wordlist possono essere cercati facilmente in rete sotto la voce di "wordhound". Allo stesso modo con PRINCE a seconda di come strutturi il tuo dizionario di input, può agire come un attacco standard del dizionario, (aggiungendo / anteponendo cifre alle parole di input, ad esempio), attacco combinatore, un attacco di forza bruta pura. Quindi riassumendo si possono vedere evidenti difetti, come:

- Gli elementi nell'elenco di parole richiedono tutte le lunghezze, per avere una maggiore completezza.
- Creazione di parole di eccessiva lunghezza.
- Genera parole duplicate.

I vantaggi sono:

- La semplicità d'uso.
- Esecuzione lunghe, tendenti all'infinito.
- Implementazione di diverse strategie di attacco.

2.5 Deep learning attack

Il Machine Learning è un sottoinsieme dell'intelligenza artificiale che si occupa di creare sistemi che apprendono, o migliorano le performance, in base ai dati che utilizzano in maniera autonoma, senza istruzioni esplicite. Tra le varie tecniche di Machine Learning ce ne è una nota come Deep Learning, apprendimento profondo, è quel campo di ricerca dell'apprendimento automatico che si basa su diversi livelli di rappresentazione, Reti Neurali ,corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso.

L'idea alla base per addestrare una rete neurale che possa determinare in modo autonomo le caratteristiche e le strutture delle password e sfruttare questa conoscenza per generare nuove stringhe che seguano la stessa distribuzione. Le reti neurali sono in grado di analizzare una grande varietà di proprietà e strutture che descrivono la maggior parte delle password scelte dall'utente; allo stesso tempo, le reti neurali possono essere addestrate senza alcuna conoscenza a priori o un'assunzione di tali proprietà e strutture. Questo concetto è in netto contrasto con gli approcci visti fino ad adesso. Si pensi ad un attacco alle password basato sui modelli di Markov. Questa tecnica presuppone che tutte le caratteristiche rilevanti delle password possano essere definite in termini di n-grammi;

o si considerano i rules attack, in grado di generare password solo in base a delle regole. Gli esempi potrebbero continuare ancora. Le password create utilizzando una rete neurale non sono limitate ad un solo aspetto particolare dello spazio delle password.

Nel campo dell'apprendimento automatico, esiste un particolare modello che è molto utilizzato ed è conosciuto come GAN. Generative Adversarial Network, in italiano rete generativa avversaria, è stata introdotta per la prima volta da Ian Goodfellow. Questo modello usa due reti neurali vengono addestrate in maniera competitiva all'interno di un framework di gioco minimax. Questo tipo di framework permette alla rete neurale di apprendere come generare nuovi dati aventi la stessa distribuzione dei dati usati in fase di addestramento.

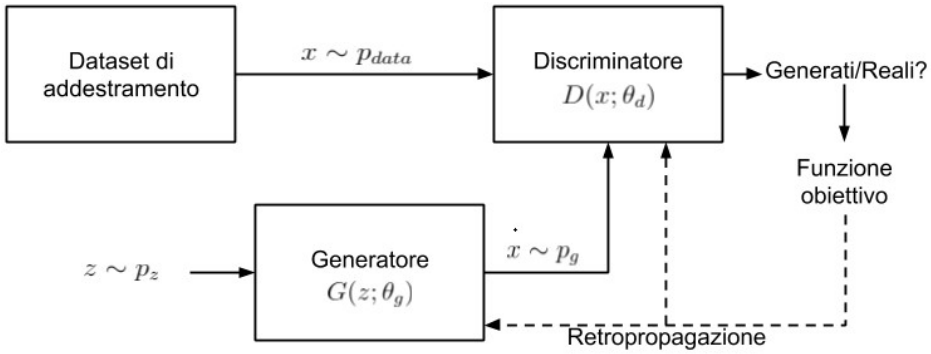

Figura 3. Rete generativa avversaria.

La GAN è composta da due componenti: un modello generativo, o generatore **G**, e un modello discriminativo, o discriminatore **D**, entrambi realizzati tramite reti neurali.

Lo scopo del modello generativo è quello di produrre nuovi dati, mentre il modello discriminativo apprende come distinguere i dati reali da quelli generati artificialmente. L'intenzione è quindi quella di creare un modello che riproduca più fedelmente possibile i dati di addestramento. Una volta raggiunto questo scopo, il discriminatore D, non riuscirà più a distinguere i dati reali da quelli generati. In particolare, dato uno spazio latente *z*, avente una distribuzione a priori $p_z(z)$, il generatore rappresenta una funzione differenziabile $G(z,\theta_g)$ che fornisce in output i nuovi dati secondo una certa distribuzione p_g, dove θ_g sono i parametri del modello generativo. Il discriminatore rappresenta una funzione differenziabile $D(x,\theta_d)$, dove θ_d sono i parametri del modello discriminativo, che produce in output la probabilità che x provenga dalla distribuzione dei dati di addestramento p_{data}. Lo scopo è quello di ottenere un generatore che sia un buon stimatore

di p_{data}. Quando questo avviene, il discriminatore viene raggirato e non riesce più a distinguere i campioni provenienti da p_{data} da quelli provenienti da p_g. La chiave per raggiungere questa situazione è l'addestramento competitivo. La rete discriminativa viene addestrata in modo da massimizzare la probabilità di classificare correttamente i campioni provenienti dai dati di addestramento e i campioni generati. Allo stesso tempo, la rete generativa viene addestrata minimizzando:

$$\log(1-D(G(z))$$

e massimizzando quindi la probabilità del discriminatore di considerare i campioni prodotti dalla rete generativa, ovvero $x \sim p_g$, come provenienti da p_{data}. L'apprendimento consiste quindi nell'ottimizzare un gioco minimax a due giocatori, D e G:

$$\min_G \max_D E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1-D(G(z)))]$$

in cui l'ottimo globale è $p_g = p_{data}$.

D(x) è la funzione che rappresenta il discriminatore che prende in ingresso l'istanze x e restituisce la probabilità che tale istanza sia reale. E_x è il valore atteso su tutte le istanze di dati reali. G(z) è l'output del generatore quando viene dato in input istanze random. D(G(z)) stima del discriminatore su un istanza falsa. Restituisce la probabilità che un'istanza artificiale sia reale. Infine E_Z è il valore atteso su tutte le istanze di dati artificiali.

Le due reti vengono addestrate in maniera alternata tramite retropropagazione dell'errore:

- Prima D viene addestrata in modo da massimizzare la probabilità di classificare in maniere esatta i campioni, sample, provenienti dai dati di addestramento e i campioni generati.
- Successivamente G viene allenata per massimizzare la probabilità di D di considerare i campioni prodotti dalla rete generativa.

Mantenendo invariati i parametri del modello generativo durante l'addestramento del discriminatore e, viceversa, mantenendo invariati i parametri della rete discriminativa durante l'addestramento del generatore [38] [39].

Capitolo 3

3 Studio delle tecnologie utilizzate

In questo capitolo si vogliono illustrare i principali software analizzati, che rappresentano lo stato dell'arte del password cracking offline. In aggiunta si esporranno i componenti e strumenti che faranno parte del progetto finale.

Le strutture che compongono il sistema di testing si dividono in tre parti:

1. **Sezione di training**: in questa prima fase vengono passati in inputi i dataset così che i vari software creino i loro dizionari. Le modalità con cui verranno generate varia da strumento a strumento e da dizionario a dizionario.
2. **Sezione generativa**: in questa fase, che è definita di Password Generation o Password Guessing, dopo aver creato un dizionario il software genererà le password che saranno fase di test nella terza fase.
3. **Sezione recuperativa**: in questa fase si prende l'output della sezione generativa e mediante una funzione di hashing si confronteranno i risultati con la parte del dataset che useremo come riferimento. In questa sezione si rilevano le password sconosciute. Lo strumento utilizzato è Hashcat, ma se ne potevano usare altri come John the Ripper.

Alla fine di questo capitolo si vedranno anche alcuni strumenti utilizzati al fine di analizzare ed eseguire i software in questione [19] [20].

3.1 Password Guessing

In questa parte andremo ad elencare i vari software analizzati. I software analizzati hanno utilizzato in ingresso gli stessi tipi di dataset, quindi i risultati alla fine saranno comparati in base alle esigenze.

3.1.1 PCFG-Cracker

PCFG-Cracker è un tool sviluppato da Matt Weir, e utilizza la logica del PCFG. Questo tool crea un modello (delle regole), che viene addestrato su un dataset di password plausibili, cioè stringhe comuni o parole chiave del target in modo tale da seguire la stessa struttura e distribuzione. Tramite vari comandi si può decidere se il software deve essere

sensitivo al maiuscolo, oppure indicare quanto il dataset in input sia attinente a quello di test. Durante la fase di training, l'algoritmo estrae tutte le informazioni possibili sulla struttura delle password come visto nel capitolo precedente. Verranno misurate le frequenze di dei pattern ricorrenti, saranno analizzate le frequenze etc. Come seconda fase si analizzano la frequenza dei caratteri, in particolar modo delle stringhe numeriche D e speciali S, per questioni di memoria. Il PCFG-Cracker, a differenza della versioni standard che utilizzava solo tre tipi di dato, sradicherà dalla password molte più informazioni. Tra i possibili simboli troviamo: lettere alfabetiche (A), cifre (D), lettere maiuscole (U), lettere minuscole (L), keyboard patterns (K), caratteri speciali (O), alcune informazioni personali come anni (Y) o email (E) e altro ancora. Tutte le informazioni necessarie saranno poi associate ad una probabilità e conservate in tabelle divise per tipologia di dato. Successivamente alla creazione di queste tabelle si passerà alla generazione delle password. La generazione avverrà a partire dalle parole con probabilità maggiore fino a quelle meno probabili. Il processo di creazione andrà avanti finché non sarà scoperta la password o l'attaccante si arrenderà. Altra parte da non sottovalutare di questo tool è la possibilità di riutilizzare queste tabelle/informazioni in altri strumenti di cracking come PRINCE e/o parti del set di regole possono essere incorporate direttamente in attacchi più tradizionali basati su dizionario.

PCFG è scritto in python, al suo interno ci sono due programmi:

1. PCFG-Trainer: crea il modello basato su PCFG allenato sul dataset di input. Estrae quindi tutte le informazioni dalle password di training. Tra le opzioni più interessanti, come citato prima, troviamo la possibilità di far riconoscere al trainer alcune informazioni sensibili –save_sensitive e –coverage per indicare l'affidabilità del dataset.

Figura 4. Schermata di help del file trainer.py di PCFG.

2. PCFG-Guesser: genera possibili password utilizzando il set di regole (ruleset) in ordine di probabilità decrescente. Le password candidate sono inviate su STDOUT e quindi molto facilmente si possono convogliare come input su un altro programma o su un file. Non manca anche la possibilità di salvare una sessione di cracking per usi futuri o per altri motivi in cui ci ritroviamo a bloccare la sessione in corso attraverso il comando -s SESSION_NAME --load.

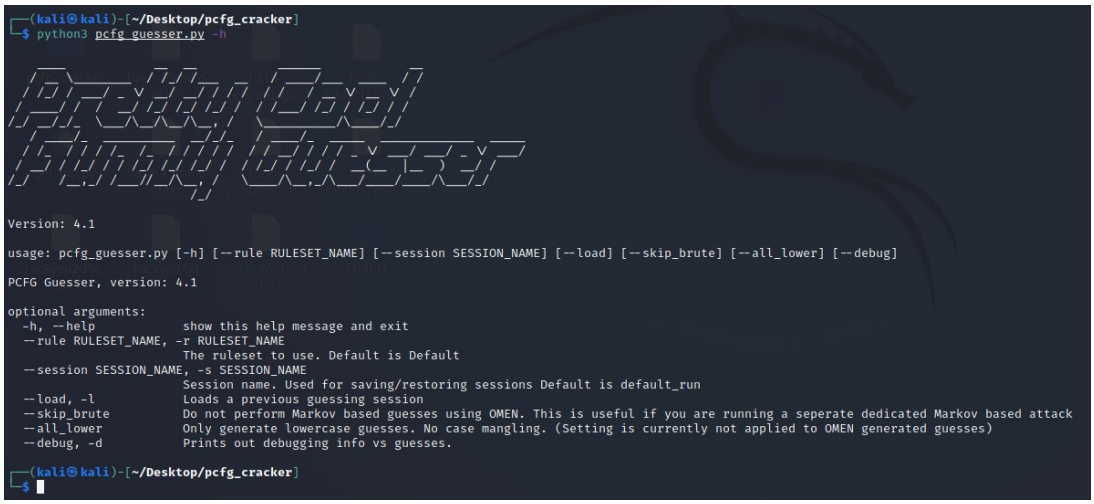

Figura 5. Schermata di help del file pcfg_guesser.py di PCFG.

I requisiti per iniziare ad utilizzare il PCFG-Cracker sono Python3 e il modulo Chardet, che verrà utilizzato in fase di Training per scoprire con quale codifica dei caratteri è stato scritto il training set (ASCII, UTF-8, ecc.). Il PCFG-Guesser è Single-Threaded CPU Buond quindi utilizza principalmente un Thread durante l'utilizzo massivo della CPU: difatti si può notare che la fase generativa è abbastanza lenta (prendendo in confronto gli altri strumenti). È consigliato per aumentare le prestazioni di dedicargli almeno più di 8GB di memoria, soprattutto con sessione di cracking superano le 4/5 ore. Durante questo tempo il PCFG creerà delle enormi tabelle di ruleset che durante l'esecuzione salverà in memoria. PCFG ha a disposizioine un ruleset di default, che però ridotto, è stato allenato su un training set di centomila password. Per ragioni di comodità, nonostante si potesse passare direttamente il risultato dell'esecuzione ad Hashcat, si sono passati i file creati in un secondo momento, per poi poter sapere la sua efficacia.

3.1.2 Princeproscessor

Il tool Princeprocessor è stato implementato con l'algoritmo PRINCE. Come si è potuto vedere nel capitolo 2.4, PRINCE è l'acronimo di PRobability INfinite Chained Elements, cioè gli elementi fondamentali dell'algoritmo. PRINCEProcessor è stato pensato nella sua essenza come un attacco combinatorio avanzato, infatti una delle sue caratteristiche è quello di essere facilmente affiancato da un altro strumento. Al posto di ricevere in input due diversi elenchi di dataset, poi creare tutte le possibili combinazioni di due parole, il tool riceve solo un dizionario di parole in input e costruisce "chain" (catene) di parole combinate. Le "chain" che si formano si basano su concatenazioni da 1 a N parole dall'elenco di input. Il numero di parole concatenate massimo/minimo può variare modificando delle opzioni dello script.

Il Princeprocessor è stato progettato per attaccare hash lenti ma si comporta egregiamente anche con hash più rapidi, come MD5. Ricordiamo infine che uno dei punti di forza di PRINCE, e di conseguenza anche del tool che lo implementa, è il runtime pressochè infinito, quindi questo strumento andrebbe lasciato lavorare per tantissime ore per ottenere risultati soddisfacenti. Ricerche hanno dimostrato come PP sia capace di recuperare più del 60% di password lavorando per 24 ore [40]. Nonostante queste premesse i risultati di questo strumento sono sotto alla media perché il dataset preso in ingresso non rispecchia le specifiche fatte nel capitolo 2.4; oltre ad essere stato usato con i dataset di riferimento è stato usato con un dataset estratto con wordhound, un tool che permette di estrarre informazioni da pagine social. Avendo dato in ingresso un dataset opportuno ha avuto prestazioni eccellenti, ma questi test non saranno riportati perché non seguono la procedura tenuta per gli altri tools.

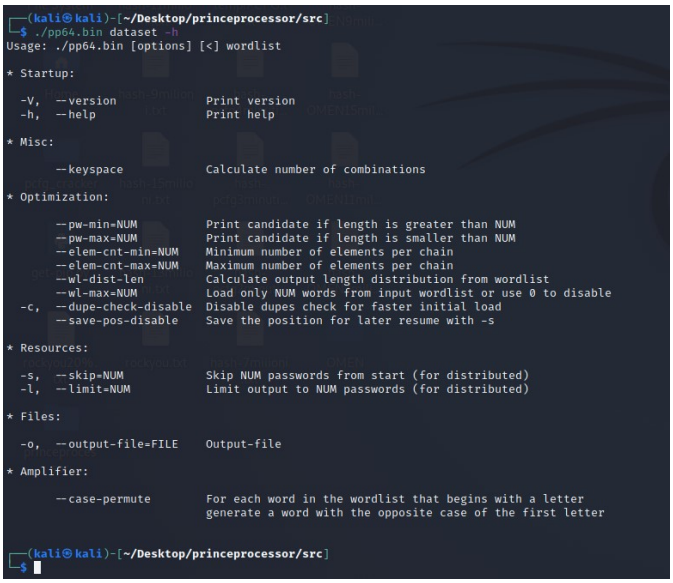

Figura 6. Schermata di help Princeprocessor.

3.1.3 OMEN

OMEN è un software che fa cracking di password basato su un modello di Markov. Il principio sul quale si basa il tool Ordered Markov ENumerator è quello di enumerare le password con probabilità decrescente. L'algoritmo discretizza tutte le probabilità in un certo numero di bin: il numero che viene dato all'insieme rispecchia la probabilità delle password che verranno inserite in esso. Una volta identificati i bin vengono loro associate le corrette password. Successivamente si itera su questi set in ordine decrescente di probabilità ed infine vengono trasmessi in output.

L'algoritmo comincia con il calcolo del logaritmo di tutte le probabilità degli n-grammi e discretizza queste probabilità in livelli che denotiamo con η. I livelli sono discretizzati tendenzialmente in 10 livelli, da 0 a -9. Il numero di livelli può variare: aumentando i livelli si aumenta l'accuratezza ma si aumenta anche il runtime. La formula che descrive questa divisione è:

$$\eta_i = lvl_i = \text{round}(\log(c_1 \cdot prob_i + c_2))$$

dove c_1 e c_2 sono parametri scelti dall'algoritmo in modo tale che gli n-grammi più frequenti abbiano un livello uguale a 0 e che agli n-grammi, che non sono comparsi nell'addestramento, sia assegnata una probabilità ancora più piccola.

L'algoritmo OMEN(η, l) prende in ingresso un livello η e una lunghezza l e avanza come segue:

1. Istanzia tutti i vettori $a = (a_2,..,a_l)$ di lunghezza pari a l - 1 così ogni istanza a_i sia un intero compreso tra 0 e -9 e la somma di questi elementi sia η. Si noti che la lunghezza delle istanze di a deve essere impostata a l - 1 in quanto avremo sempre una probabilità iniziale da cui partire.
2. Per ciascuno di questi a_i, seleziona tutti i 2-grammi x_1x_2 le cui probabilità corrispondono al livello a_2. Per questi 2-grammi, si itera su tutti gli x_3 in modo tale che il 3-grammo $x_1x_2x_3$ abbia livello a_3. Successivamente, per ciascuno di questi 3-grammi, si esegue un ciclo su tutti gli x_4 al fine di avere il 3-grammo $x_2x_3x_4$ abbia il livello a_4, e così via, fino a raggiungere la lunghezza desiderata. Alla fine, questo processo restituisce un insieme di password candidate di lunghezza l e livello η.

Di seguito l'algoritmo in maniera formale:

Algoritmo 1 Enumerazione di password per livello η e lunghezza l. Esempio con $l = 4$.
function OMENPwd(η , l) for each vector $(a_i)_{2 \le i \le 1}$ with $\sum_i a_i = \eta$ for each $x_1x_2 \in \sum^2$ with $L(x_4\|x_1x_2) = a_2$ for each x_3 2 ∈ with $L(x_3\|x_1x_2) = a_3$ for each x_4 2 ∈ with $L(x_4\|x_2x_3) = a_4$ (a) output $x_1x_2x_3x_4$

Con L(xz) e L(y|xz) indico i livelli per le probabilità iniziali e condizionate.

Per comprendere al meglio l'idea di base di OMEN svolgo un esempio che si può ritrovare anche nel lavoro di Castelluccia et al. in maniera meno discorsiva. Si consideri una password di lunghezza pari a l = 3 che costruirò utilizzando un alfabeto $\sum = \{a, b\}$, con probabilità di partenza (ovvero i "livelli", dopo aver discretizzato le probabilità) pari a:

$$L(aa) = 0, L(ab) = -1;$$

$$L(ba) = -1, L(bb) = 0$$

E con probabilità condizionate (transizioni) pari a:

$$L(a|aa) = -1,\ L(b|aa) = -1;$$

$$L(a|ab) = 0,\ L(b|ab) = -2;$$

$$L(a|ba) = -1,\ L(b|ba) = -1;$$

$$L(a|bb) = 0,\ L(b|bb) = -2$$

Si nota che la regola generale: dato un livello η si dovrebbe ottenere una password di lunghezza l tale che la somma delle probabilità/livelli delle "parti" che compongono la password sia pari a η.

- **η = 0**: partendo dal livello η = 0, si deve ottenere una o più password tale che sia la probabilità iniziale sia quella condizionata siano pari a 0: in altre parole un vettore di elementi le cui probabilità sono alla pari di (0,0). Si noti che le stringhe che hanno probabilità di partenza pari a 0 sono *aa* e *bb*. Ora che si ha la stringa iniziale si deve cercare l'ultima parte di password tale che la somma dei livelli rimanga 0. La password *bba* soddisfa questi requisiti: infatti L(*a*|*bb*) = 0 ammette la possibilità di generare la stringa *bba* così da non modificare la somma dei livelli. Non si trovano però corrispondenze per la stringa *aa*. Infatti nessuna delle probabilità condizionate contenente *aa* come evento di partenza, ha livello 0.
- **η = -1**: si procede specularmente al livello superiore. Il livello η = -1 manderà in output due vettori. Il primo (-1,0), con al suo interno *aba* (il prefisso *ba* non ha transizioni di livello 0), ed il secondo vettore (0,-1) che contiene *aaa* e *aab*.
- **η = -2**: il livello η = -2 restituirà tre vettori: (-2,0), senza niente al suo interno (nessuna delle probabilità iniziali vale -2), (-1,-1) con *baa* e *bab* e infine (0,-2) con *bba*.
- **altri η**: si procede così per i restanti livelli.

Il software vero e proprio è composto da due programmi:

- **createNG**: la generazione delle password con OMEN comincia con il calcolo delle probabilità degli n-grammi. Il modulo createNG svolgerà i suoi calcoli utilizzando le probabilità e basandosi sulle impostazioni predefinite (10 livelli, ecc.). Il modello sarà addestrato su un dataset di testo contenente le password di training, che saranno disposte una per riga. Al termine dell'esecuzione il programma genererà quattro file aggiuntivi contenenti informazioni sui n-grammi e sulla lunghezza della password. Questi file hanno tutti estensione .level :

- ❖ IP.level (Initial Probability): Salva le probabilità (della prima parte di password in pratica) del primo (n-1)-grammo per ogni password.
- ❖ CP.level (Conditional Probability): Memorizza le probabilità condizionate.
- ❖ EP.level (End Probability): Le probabilità (dell'ultima parte di password in pratica) dell'ultimo (n-1)-grammo di ogni password sono salvate in questo file.
- ❖ LN.level (Length): Memorizza le probabilità per ogni lunghezza possibile delle password in input. Queste probabilità saranno espresse con un valore compreso tra 0, più probabile, e 10 meno probabile.

- **enumNG**. Una volta che i file necessari alla generazione delle password sono stati creati allora si passerà all'effettiva creazione con il modulo enumNG. Queste password verranno visualizzate a video, quindi facilmente salvabili su file o altro.

Come presentato in precedenza, l'algoritmo di enumerazione (η, *l*) utilizza due parametri. Questi due parametri devono essere impostati correttamente. La selezione di, ovvero la lunghezza della password da indovinare), è un parametro importante, poiché la frequenza con cui la lunghezza di una password appare nei dati di addestramento non è un buon indicatore della frequenza con cui deve essere indovinata una lunghezza specifica. Ad esempio, supponiamo che ci siano tante password di lunghezza 7 e di lunghezza 8, allora la probabilità di successo delle password di lunghezza 7 è tanto maggiore quanto minore è lo spazio di ricerca. Quindi, le password di lunghezza 7 dovrebbero essere indovinate per prime. Pertanto, utilizziamo un algoritmo adattivo che tiene traccia della percentuale di successo di ciascuna lunghezza e pianifica più password da indovinare per quelle lunghezze che erano più efficaci.

Più precisamente, il nostro algoritmo di pianificazione delle password adattiva funziona come segue:

1. Per tutti gli *n* valori di lunghezza di *l* (consideriamo lunghezze da 8 a 15, ovvero n = 7), eseguire enumPwd(0, *l*) e calcolare la probabilità di successo $sp_{l,0}$. Questa probabilità viene calcolata come il rapporto tra password indovinate con successo e il numero di password indovinate generate di lunghezza *l*.
2. Si costruisce una lista L di dimensione *n*, ordinata per probabilità di successo, dove ogni elemento è una tripla (*sp*, *livello*, *lunghezza*).
3. Selezionare la lunghezza con la più alta probabilità di successo, ovvero il primo elemento L[0] = (sp_0, $livello_0$, $lunghezza_0$) e rimuoverlo dall'elenco.

4. Eseguendo l'enumerazione ($livello_{0\text{-}1}$, $lunghezza_0$), si calcola la nuova probabilità di successo sp^* e aggiunge il nuovo elemento (sp^*, $livello_{0\text{-}1}$, $lunghezza_0$) a L.
5. Si ordina L e vai al passaggio 3 finché L non è vuoto o sono state fatte sufficienti ipotesi.

In questa sezione si analizzano le diverse scelte di parametri ed esamina il necessario compromesso tra accuratezza e prestazioni. I tre parametri centrali sono:

1. Dimensione n-grammo: il parametro con il maggiore impatto sulla precisione è la dimensione degli n-grammi. Un n più grande generalmente da risultati migliori poiché n-grammi più grandi forniscono un'approssimazione più accurata alla distribuzione della password. Tuttavia, implica un tempo di esecuzione maggiore, oltre a requisiti di memoria e archiviazione più grandi. Si noti inoltre che la quantità di dati di addestramento è cruciale poiché solo una quantità significativa di dati può stimare con precisione i parametri (cioè le probabilità iniziali e le probabilità di transizione).
2. Dimensione alfabetica: la dimensione dell'alfabeto è un altro fattore che ha il potenziale per influenzare sostanzialmente le caratteristiche dell'attacco. Una dimensione dell'alfabeto più grande significa che è necessario stimare più parametri e che i requisiti di runtime e memoria aumentano. Al contrario, una piccola dimensione dell'alfabeto significa che non tutte le password possono essere generate. Da analisi si è evidenziato che si ha un aumento della precisione da una dimensione dell'alfabeto k da 20 a 62. Un ulteriore aumento di k non aumenta notevolmente il tasso di precisione. Ciò è spiegato principalmente dall'alfabeto utilizzato nel set di dati come RockYou, dove la maggior parte degli utenti preferisce la password con caratteri prevalentemente alfanumerici piuttosto che utilizzare un numero elevato di caratteri speciali. Per essere indipendenti dai dati, abbiamo optato per l'alfabeto a 72 caratteri. Si noti che i set di dati che utilizzano lingue e alfabeti diversi.
3. Numero di livelli per l'enumerazione delle password: come per i parametri precedenti, un numero maggiore di livelli può potenzialmente aumentare la precisione, ma aumenta anche il tempo di esecuzione.

3.1.4 NEMO

NEMO usa l'architettura del software OMEN. Il nome è l'inverso del primo e sono stati progettati dallo stesso team [41].

OMEN utilizza i cosiddetti livelli, chiamati bin, questa implementazione no. Pertanto, l'enumerazione efficiente delle password candidate non è possibile se lo spazio delle chiavi diventa troppo grande. Tuttavia, a causa dell'output non raggruppato, questo software presenta altri vantaggi, ad esempio può produrre stime di resistenza più accurate. Qui sotto saranno analizzate le stime effettuate:

Correlazione. Un modo semplice per misurare la somiglianza, che è stato utilizzato nella maggior parte dei lavori precedenti, è la correlazione tra il riferimento e i valori osservati. Un problema comune con le misure di correlazione è il seguente: trattano le password frequenti e poco frequenti come punti dati ugualmente ponderati. Ad esempio un errore in una singola password rara è valutato ugualmente come un errore in una password frequente che influenza molti più account. Le misure di correlazione ponderate danno pesi specifici ai punti dati, che consideriamo la frequenza nel set di dati di riferimento.

Coefficiente di correlazione di Pearson: la misura di correlazione probabilmente più conosciuta. È definita come la covarianza divisa per entrambe le deviazioni standard. La correlazione di Pearson ha diversi problemi come misura di somiglianza per i PSM (Password strength meters): in primo luogo, è sensibile alle trasformazioni monotone, il che è indesiderabile. Ancora peggio, è molto sensibile alla quantizzazione. Due proprietà della correlazione di Pearson sono alla base di questo comportamento indesiderabile. Innanzitutto, è una misura parametrica e calcolata dai valori dati (anziché, ad esempio, ranghi come la correlazione di Spearman), il che lo rende sensibile alle trasformazioni non lineari dei dati. In secondo luogo, dà a ogni punto dati lo stesso peso, anche se le password deboli hanno un conteggio molto più alto (per definizione), quindi la correlazione di Pearson pesa le deviazioni per le password forti più forti, relativamente parlando.

Coefficiente di correlazione del rango di Spearman: è definito come la correlazione di Pearson sui ranghi dei dati. Pertanto si basa solo sui ranghi dei dati (ordinati). Spearman è robusto contro le trasformazioni monotone e abbastanza tollerante alla quantizzazione,

che è un miglioramento rispetto a Pearson. Tuttavia, dà troppo peso alle password complesse, in modo simile alla correlazione di Pearson.

Correlazione Pearson ponderata: è definita come correlazione Pearson (normale) ma ponderando ogni punto dati con un vettore di peso, dove utilizziamo il riferimento del training set come vettore di peso. Questa misura di somiglianza mostra problemi simili come la correlazione di Pearson non pesata per le trasformazioni monotone, come previsto, anche se l'effetto è meno pronunciato e rimane altamente sensibile ai dati quantizzati.

Correlazione Spearman ponderata: è definita come correlazione Pearson ponderata nei ranghi. È la misura di somiglianza più promettente considerata finora. Come la normale correlazione Spearman (non ponderata), gestisce bene le trasformazioni monotone. Fornisce una correlazione vicina a 1 al caso di test evaluation (e alle altre trasformazioni monotone inclusi i casi di test più quantizzati, che prima era problematico), poiché ora i pesi impediscono la sovrarappresentazione di password complesse.

Errore medio. Un altro insieme di misure di somiglianza è l'errore quadratico medio (MSE) e concetti correlati. Abbiamo testato le variazioni, ispirati ai risultati di cui sopra e alle tecniche utilizzate nel lavoro precedente.

Errore assoluto medio (MAE): è definito come l'errore assoluto medio, con uguale peso per ciascun punto dati. Una misura simile è stata utilizzata di recente, in cui è stato utilizzato un errore logaritmico. Grandi deviazioni nella valutazione per le singole password hanno solo un impatto moderato sulla somiglianza (a causa dell'assunzione solo di errori assoluti). La sua sensibilità alle deviazioni nelle password frequenti è bassa.

Errore quadratico medio (MSE): è definito come la media sull'errore al quadrato, dando più peso a grandi deviazioni. Le proprietà di MSE sono molto simili a quelle di MAE.

Classificato Media Absolute/Squared Error (rMAE/rMSE): qui prima classifichiamo i dati (assegnando i legami con la media rank) e calcolare il MAE o MSE dei ranghi. Come previsto, le misure risultanti sono resistenti alle trasformazioni monotone. Tuttavia, poiché non sono ponderate, non riescono a catturare gli errori per poche password frequenti. Ciò significa che, nel caso di test PSM con prestazioni scadenti, sia rMAE che rMSE non mostrano alcuna differenza rispetto al riferimento rendendoli inadatti.

Errore medio ponderato. Tutte le misure di errore discusse nella sottosezione precedente non sono ponderati e quindi non riescono a catturare gli errori in poche password frequenti sottosezione, consideriamo varianti pesate.

Media ponderata e classificata Ass./Mq. Errore (wrMAE/wrMSE): quando utilizziamo punti dati sia classificati che ponderati, la misura di somiglianza risultante diventa più discriminante, ad esempio: permette di distinguere i casi di test INV-WEAK-5 e DOUBLE. Entrambe le misure funzionano molto bene sui nostri casi di test (ricorda che valori più bassi significano più somiglianza) e sembrano essere una scelta ragionevole.

Errori unilaterali. Come descritto in precedenza, le approssimazioni sulla sicurezza della password possono essere sottostimate o sopravvalutate. Il primo fa sì che un utente selezioni semplicemente un'altra password (presumibilmente sicura), mentre nel secondo caso l'utente crede di aver selezionato una password sicura, dove in realtà è debole.

Media ponderata e classificata Ass./Squared One Sided Lower Error (wrLAE/wrLSE): è possibile definire versioni per MAE/MSE che tengono conto solo degli errori unilaterali. Se questa misura opera sui valori di conteggio, questo approccio favorisce i misuratori che generalmente sottovalutano la sicurezza: un misuratore che valuta tutte le password non sicure (cioè un valore di conteggio elevato) riceverà un punteggio elevato. Ciò può essere evitato operando su dati classificati. Sui set di dati testati e sui casi di test le misure risultanti, wrLAE/wrLSE si comportano in modo simile alle loro versioni a due lati wrMAE/wrMSE. Una probabile spiegazione è che wrLAE/wrLSE operano su dati classificati. Pertanto, sopravvalutare la sicurezza di una password generalmente porta a sottovalutare la sicurezza di un'altra password. (Pesi e differenze di quadratura (wrLSE) significano che i risultati possono ancora differire, tuttavia, questi effetti sembrano uniformarsi sul set di dati che abbiamo considerato.) Per le applicazioni che richiedono metriche unilaterali, si dovrebbero considerare metriche di somiglianza non classificate a costo di perdere la capacità di tollerare trasformazioni monotone.

Errori a coppie. Nei test preliminari, si è osservato che diverse misure di somiglianza danno un basso punteggio di somiglianza ai dati quantizzati. Questo comportamento è indesiderabile, poiché la quantizzazione pesante perde informazioni sulla distribuzione. Abbiamo cercato di affrontare questo problema progettando un punteggio di somiglianza che si basa su due metriche individuali: una metrica di errore che descrive quante password non sono nell'ordine "corretto" e una metrica di utilità che descrive se il

contatore fornisce informazioni "utili" e uscita discriminante. (Per illustrare questo problema, si consideri un misuratore di forza con output binario, in cui vengono "accettate" solo poche password molto forti e "rifiutate" le altre password. Questo misuratore avrebbe un basso indice di errore, poiché preserva principalmente l'ordine, ma una valutazione di utilità bassa, poiché la maggior parte delle password si trova nello stesso cestino). Questo meccanismo si basa sul rango. Abbiamo valutato diverse varianti di questa idea di base.

Pairwise Error/Utility Rate (PE/PU): Considerano la classifica relativa di tutte le coppie di password. PE considera la frazione di coppie in cui il contatore e il riferimento non sono d'accordo (dove un pareggio in uno dei due non viene conteggiato come un disaccordo), mentre il PU considera la frazione di coppie in cui il contatore vede un pareggio. (Un contatore che emette lo stesso livello di sicurezza per tutte le password, ovvero utilizza un singolo bin, ha un PE pari a 0, ma anche un PU pari a 0.)

Tasso di errore a coppie superiore al 5 % (PE-5): poiché in genere si verificano piccole deviazioni considerato un non problema, per questa variante tolleriamo qualsiasi deviazione inferiore al 5% (in termini di rango) e non li conteggiamo per l'errore.

Errori a coppie ponderati. Abbiamo già sostenuto che le misure non ponderate che non tengono conto delle probabilità specifiche delle password distorcono sistematicamente i risultati. Errore ponderato a coppie/Tasso di utilità (wPE/wPU)/Tasso di errore a coppie ponderato superiore al 5 % (wPE-5): utilizziamo versioni pesate delle tre misure introdotte in precedenza, dove pesiamo ciascuna coppia con il prodotto delle probabilità del due password.

3.1.4 PassGAN

Nel processo password cracking è stata applicata l'intelligenza artificiale. Grazie allo sviluppo di nuove tecniche e della potenza di calcolo si possono riconoscere pattern o schemi.

Uno strumento precursore che fa l'utilizzo dell'IA e che ha riscontrato molto successo è il seguente tool: PassGAN. Il problema della comprensione di una struttura della password viene ora stravolto da questo nuovo approccio Deep Learning Based. Invece di utilizzare tecniche probabilistiche o pattern prestabiliti, il core di PassGAN fa uso di Generative Adversarial Network o GAN una particolare rete neurale che gli permetterà di generare

password possibili di altissima precisione. Con le regole classiche di generazione delle password il numero di termini univoci che otterremo sarà prestabilito dal numero di regole stesse e dalla dimensione del set di dati in input. PassGAN può invece produrre un numero illimitato di password, questa affermazione è vera se il training set è di grandi dimensioni. L'implementazione del modello di PassGAN utilizza una specifica rete neurale chiamata WGAN, descritta da Gulrajani et al. [42], implementata sempre da Gylrajani.

Questa rete neurale fa uso di vari iper-parametri, tendenzialmente questi valori sono settati prima della fase di learning che migliorano la fase stessa. I settaggi degli iperparametri sono rimasti quasi tutti invariati rispetto al paper iniziale e sono:

- **Batch Size**: il training set potrebbe essere troppo grande per essere elaborato tutto in una volta. Quindi si può suddividere in sottogruppi uniformi, chiamati batch. Il numero di esempi contenuti in ogni batch è chiamato batch size. Nel nostro caso il batch size sarà il numero di password che dal training set sarà inviato attraverso la GAN per ogni step dell'ottimizzatore. Il batch size è fissato a 64.
- **Numero di iterazioni**: il numero di iterazioni corrisponde al numero di batch necessari a completare un epoch, ovvero quando il modello è stato allenato sull'intero training set. Se ad esempio avessimo un training set di 200 istanze e lo dividessimo in batch da 50, avremmo allora che una epoch è formata da 4 iterazioni. Nel nostro caso specifico il numero di iterazioni indica quante volte il GAN invia i dati e riceve un feedback. Per ogni iterazione di GAN, il generatore compie un'iterazione mentre il discriminatore ne farà una o più. Inizialmente il modello veniva allenato per 200 mila iterazioni ma per la limitazione temporale delle risorse ne saranno eseguite 130 mila.
- **Numero di iterazione del discriminante rispetto alle iterazione del generatore**: come antecedentemente visto ad ogni iterazione del generatore il discriminante può fare molteplici iterazioni. Questo numero quindi sta ad indicare quante ripetizioni il discriminante fa per ognuna del GAN. Per default il valore è 10.
- **Dimensione del modello:** una rete neurale può essere immaginata come composta di diversi "layer", o strati, di nodi, ciascuno dei quali è collegato ai nodi del layer successivo. Questo valore rappresenta il numero di dimensioni per ciascun layer.

Nel modello del PassGAN si avranno 5 layer, che chiameremo Residual Block, sia per la rete generativa sia per quella discriminatoria. Ciascun layer avrà 128 dimensioni ovvero 128 features/parametri che il nostro modello cercherà di ottimizzare al meglio.

- **Gradient Penalty Coefficent o λ**: è un iperparametro che serve per regolarizzare la parte di training, quindi renderla più stabile. In particolare specifica la penalty (riduzione/aggiustamento) che verrà applicata alla norma del gradiente della rete discriminatoria rispetto al suo input. Più l'iperparametro è alto più tende ad essere stabile, entro certi limiti. Nella relazione di PassGAN [43], è impostato a 10 questo valore.
- **Lunghezza massima delle sequenze di output**: indica la lunghezza massima che le stringhe in uscita da PassGAN avranno. Questo valore è impostato a 10 e sarà uno dei valori più critici e su cui ci si dovrà soffermare durante la fase di testing.
- **Dimensione del vettore random/noise**: come si può notare dalla figura 8 e 9 dove viene mostrata la struttura del PassGAN. La rete che genera password artificiali prendendo in ingresso un vettore di numeri random, il così detto "rumore". La quantità di numeri random che verranno forniti in ingresso dipenderà da questo iperparametro, fissato a 128. Il "rumore" segue una distribuzione normale (ovvero una distribuzione di numeri che segue una curva gaussiana) e tali dati saranno di tipo float.
- **Numero massimo di istanze in fase di training**: il parametro indica il numero di oggetti che verranno presi in considerazione durante la fase di training. Qui l'intero dataset di password in input sarà usato come train-set.
- **Iperparametri dell'Ottimizzatore Adam**: il deep learning è un processo iterativo con molteplici parametri da settare propriamente. Il nostro obiettivo è generare un modello capace di generalizzare i dati (ovvero predire correttamente istanze future) con un'alta accuratezza, cioè che abbia la minima differenza possibile tra i valori predetti e quelli reali. La differenza tra valori predetti e reali viene detta loss function e sarà misurata su ogni osservazione. Si deve minimizzare la loss function individuando i valori ottimizzati per ogni parametro del modello. Ecco che entrano in gioco gli algoritmi di ottimizzazione. Attraverso iterazioni multiple questi algoritmi consentono l'individuazione dei valori dei parametri migliori che minimizzano la loss function. L'ottimizzatore Adam è uno

dei più utilizzati e, per funzionare al meglio, deve aver impostato alcuni iperparametri:

- **Learning rate**: o velocità di apprendimento o step size, è un iperparametro di ottimizzazione per un algoritmo che determina la dimensione del passo ad ogni iterazione mentre si sposta verso una funzione minima di perdita. Il valore utilizzato rimane invariato rispetto a quello di default di 10^{-4}.
- **Coefficiente β_1 e β_2**. Valori utili al procedimento di ottimizzazione interno dell'Adam Optimizer. Impostati rispettivamente a 0.5 e 0.9.

Nella figura che seguirà, figura 7, indica l'architettura di PassGAN. Senza entrare troppo nel dettaglio si può notare come entrambe le reti neurali abbiamo 5 layer, che aiutano a ridurre l'errore di training del modello, più layer implica un minore errore, connesse a loro volta con un layer convoluzionale 1-dimensionale. Nell'input delle reti neurali bisogna spesso cercare di riconoscere dei pattern che possano essere presenti in diverse porzioni dell'input. Inserendo all'inizio della rete uno strato Conv1D e impostando sia il numero sia la lunghezza dei pattern da cercare, si ottiene in output un array 2D che indica quali pattern sono stati individuati e in che punto dell'input.

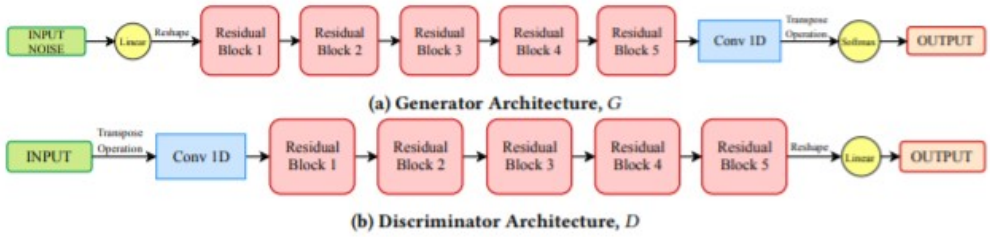

Figura 7. Schermata di help Princeprocessor: Architettura di PassGAN [16].

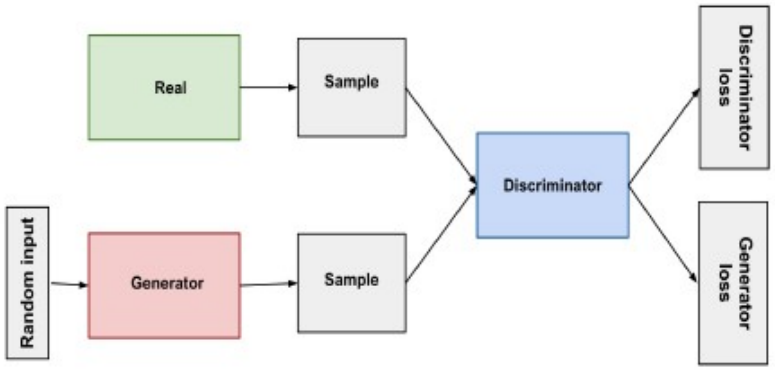

Figura 8. In figura l'intero sistema della rete GAN. Attraverso la retropropagazione il discriminatore fornisce poi un feedback al generatore che utilizzerà per aggiornare i suoi pesi.

3.1.4 Hashcat

Nello studio degli strumenti più utilizzati è stato analizzato anche Hashcat, che avrà un ruolo fondamentale nello step successivo. Hashcat è uno strumento molto duttile e possiede al suo interno un insieme di regole con le quali si possono generare nuove password. Oltre alle suo rules si possono usare le regole create dagli strumenti visti precedentemente, come PCFG, oppure si possono creare le proprie rules attraverso a degli script appositi. Questo è un esempio di utilizzo delle rules di hashcat:

hashcat --force dataset -r /usr/share/hashcat/rules/best64.rule --stdout > hashcatBest64.txt

```
root@kali:~# ls -l /usr/share/hashcat/rules/
total 2588
-rw-r--r-- 1 root root    933 Dec  2  2018 best64.rule
-rw-r--r-- 1 root root    633 Dec  2  2018 combinator.rule
-rw-r--r-- 1 root root 200188 Dec  2  2018 d3ad0ne.rule
-rw-r--r-- 1 root root 788063 Dec  2  2018 dive.rule
-rw-r--r-- 1 root root 483425 Dec  2  2018 generated2.rule
-rw-r--r-- 1 root root  78068 Dec  2  2018 generated.rule
drwxr-xr-x 2 root root  12288 Mar 23 21:02 hybrid
-rw-r--r-- 1 root root 309439 Dec  2  2018 Incisive-leetspeak.rule
-rw-r--r-- 1 root root  35280 Dec  2  2018 InsidePro-HashManager.rule
-rw-r--r-- 1 root root  19478 Dec  2  2018 InsidePro-PasswordsPro.rule
-rw-r--r-- 1 root root    298 Dec  2  2018 leetspeak.rule
-rw-r--r-- 1 root root   1280 Dec  2  2018 oscommerce.rule
-rw-r--r-- 1 root root 301161 Dec  2  2018 rockyou-30000.rule
-rw-r--r-- 1 root root   1563 Dec  2  2018 specific.rule
-rw-r--r-- 1 root root  64068 Dec  2  2018 T0XlC-insert_00-99_1950-2050_toprules_0_F
.rule
-rw-r--r-- 1 root root   2027 Dec  2  2018 T0XlC-insert_space_and_special_0_F.rule
-rw-r--r-- 1 root root  34437 Dec  2  2018 T0XlC-insert_top_100_passwords_1_G.rule
-rw-r--r-- 1 root root  34813 Dec  2  2018 T0XlC.rule
-rw-r--r-- 1 root root 104203 Dec  2  2018 T0XlCv1.rule
-rw-r--r-- 1 root root     45 Dec  2  2018 toggles1.rule
-rw-r--r-- 1 root root    570 Dec  2  2018 toggles2.rule
-rw-r--r-- 1 root root   3755 Dec  2  2018 toggles3.rule
-rw-r--r-- 1 root root  16040 Dec  2  2018 toggles4.rule
-rw-r--r-- 1 root root  49073 Dec  2  2018 toggles5.rule
```

Figura 9. Rules di Hashcat.

Figura 10. Logo Hashcat.

3.2 Strumenti per il Password cracking

Oggigiorno per mantenere informazioni altamente sensibili come le password si usano varie tecniche per garantire la riservatezza e segretezza, le principali strategia consiste nell'applicare una funzione di Hash su di essa e salvarle non in chiaro sul database.

Questa tecnica crittografica permette di convertire una password "in chiaro", in un lungo elenco di caratteri casuali praticamente inutilizzabile. Queste funzioni hash devono essere facili da calcolare ma estremamente difficili da invertire, la così detta proprietà one-way, e la firma hash, ovvero la trasformazione che la funzione hash applica sull'elemento in considerazione, per una chiave dovrà essere univoca: due elementi diversi non potranno avere la stessa immagine per un dato algoritmo di hashing, proprietà clawfree. Le famiglie delle funzioni di hashing sono molteplici: le più importanti e celebri sono MD5 (Message Digest Version 5), SHA (Secure Hash Algorithm) e RIPEMD-160 (Race Integrity Primitive Evaluation Message Digest). Nei test che verranno effettuati le password in chiaro verranno sottoposti alla funzione hash MD5:

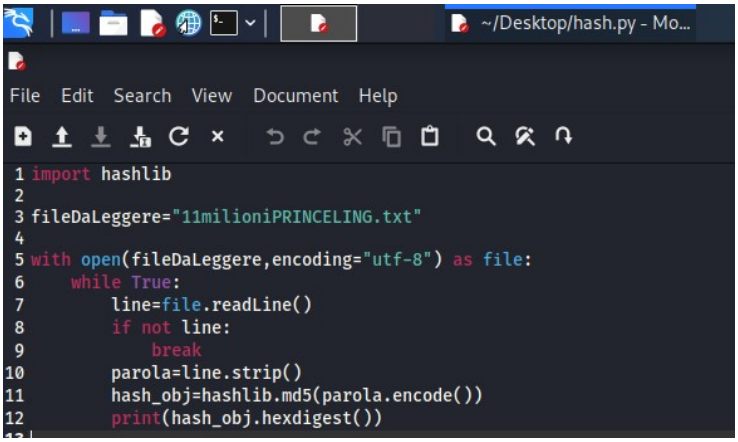

Figura 11. Script in python che esegue funzione hash su un file di password.

Anche se si entrasse in possesso di password criptate sarebbe impossibile invertire la funzione hash che "protegge" la parola, dovuta alla proprietà one-way. L'unica maniera per identificare la password in chiaro è generarne di nuove, applicare ad esse la stessa funzione hash della password target e confrontare le due immagini hash. Gli strumenti in grado di eseguire questa attività in maniera avanzata sono molteplici, John-The-Ripper, CainAbel e soprattutto Hashcat ovvero quello che sarà usato.

3.2.1 Hashcat

Hashcat è il tool di recupero password più veloce e avanzato al mondo, in grado di supportare 7 modalità di attacco per oltre 300 algoritmi di Hashing ottimizzati.

Hashcat permette di preparare attività Generetion Password come visto nel paragrafo 3.1.4, e di Password Recovery e recupero password, altamente flessibili e personalizzabili. Allo stato attuale supporta CPU, GPU (e altro hardware) sui principali sistemi operativi quali Windows, Linux e macOS. Tutti i test saranno eseguiti su una macchina virtuale con sistema operativo Kali Linux. Hashcat fornisce anche la strumentazione per il cracking distribuito delle password, ovvero crackare password con

più sistemi contemporaneamente. Uscito nel 2015, Hashcat continua ad essere un tool estremamente prezioso per chiunque operi nel campo della sicurezza informatica.

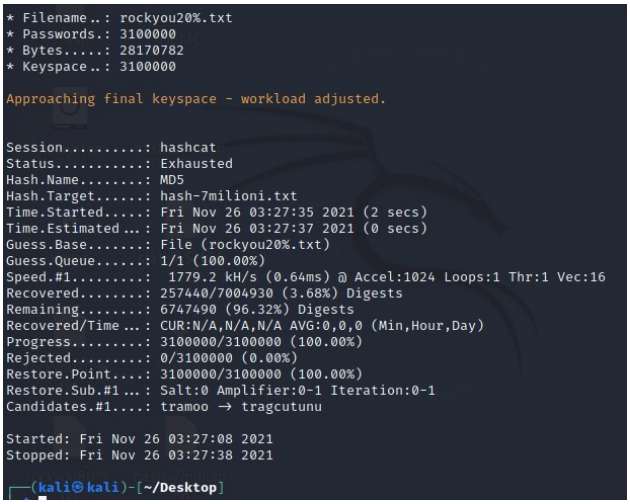

Figura 12. Esecuzione di Hashcat su Kali Linux.

3.2.2 PACK: Password Analysis and Cracking Kit

PACK, Password Analysis and Cracking Toolkit, è un insieme di utility ideate per analizzare elenchi di password. L'obiettivo è migliorare il password cracking attraverso il rilevamento di maschere, regole, set di caratteri e altre caratteristiche delle password. PACK è in grado di creare file utilizzabili da Hashcat. Le features di questo programma sono molteplici, in questo elaborato saranno usate le funzioni base: dato un dizionario verranno analizzate le caratteristiche principali e si avrà in output le frequenza dei caratteri, le password più frequenti e le maschere per portare avanti degli attacchi mirati [44].

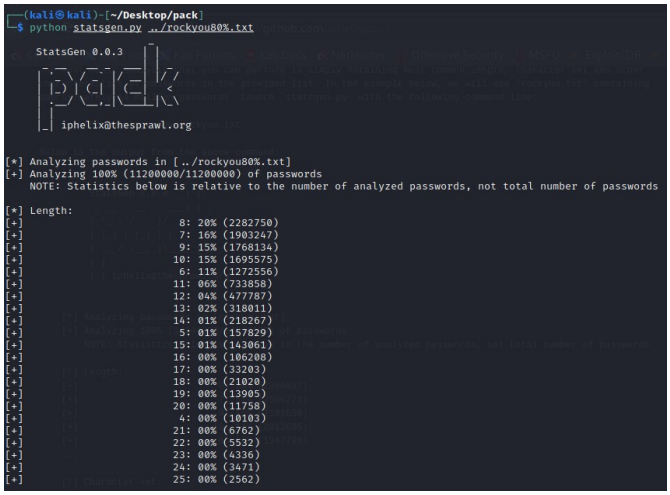

Figura 13. Esecuzione di PACK sul file rockyou.

Capitolo 4

4 Esperimenti ed analisi dei risultati

In questo capitolo verranno mostrati i risultati, completi di analisi sui vari strumenti che fino a qui sono stati analizzati. I risultati delle esecuzioni ci daranno svariate informazioni: quale strategia funziona meglio, cercare di trovare i motivi per cui un tool è più efficiente di un altro. Queste analisi sono state realizzate seguendo questi step:

1. Ricerca del dataset;
2. Divisione in training-set e test-set del dataset trovato;
3. Creazione dei dizionari finali da testare;
4. Generazione delle password con i vari strumenti;
5. Applicazione della funzione hash ai dizionari;
6. Cracking effettivo tramite Hashcat.

4.1 Dataset

In questa fase si sono analizzati e confrontati diversi dataset di password che erano presenti in rete. I test saranno effettuati su tre tipi di dataset per testare gli strumenti in differenti condizioni.

4.1.1 1-milion

È un file creato con il primo milione di password utilizzate più frequente. Questo file viene aggiornato annualmente. Questo primo dataset è stato scelto per differenti ragioni, tra le quali ci sono: testare gli strumenti su un dataset ridotto per verificare la correttezza, quindi sia per motivi di tempo che per motivi di memoria.

4.1.2 RockYou

In Kali Linux è presente un file chiamato RockYou.txt, andando nella cartella:

```
cd /usr/share/wordlist/
```

```
sudo gzip -d rockyou.txt.gz
```

Si avrà accesso ad un file di circa 15 milioni di password. Nel 2009 una società di nome RockYou è stata hackerata. L'attacco riuscì a sottrarre password in chiaro creando un dizionario base per i cracker. È la raccolta delle password più utilizzate e diffuse al mondo. Questo dizionario è annualmente aggiornato da una folta comunità di hacker e ne esistono varie versioni disponibili in rete.

4.1.3 LinkedIn

Durante la fase di ricerca di dataset, si è potuto vedere che esistono molti file sottratti a dei social network e che forniscono una moltitudine di informazioni. Per questi test useremo un file di 60 milioni di password. Questo file a differenza degli altri due visti, ha avuto bisogno di una fase di preparazione: non conteneva solo le password ma anche altre informazioni come il nome del profilo, l'ID LinkedIn, il link al profilo, l'email ed altre informazioni.

4.1.4 Manipolazione dei dataset

Oltre alla pulizia dei dati e la riscrittura secondo il formato il quale tutti gli strumenti visti accettano i dati in input, cioè una stringa per riga, si è dovuto dividere il dataset in modo che la parte principale dovesse svolgere la funzione di train e la parte secondaria come test passandola come input a Hashcat. La divisione è stata effettua tramite una divisione 80-20, per poter permettere un giusto trade-off e consumo di risorse ed ottenere risultati accettabili. Per rendere la divisione più parziale possibile è stato usato uno script dedicato che oltre a dividere il dataset in due lo filtrava da password con più di 15 caratteri.

```
└─$ cat prep-data.py
import sys
import random

random.seed(1337)


with open('../data/rockyou.txt', 'r') as f:

    # we can't line buffer because we need everything to randomize
    lines = f.readlines()

    # filter only passwords with 15 characters or fewer
    print('[info] filtering rockyou to include only 15 character or less passwords')
    lines = filter(lambda x: len(x) <= 15, lines)

    # randomize order
    print('[info] shuffling rockyou')
    random.shuffle(lines)

    split = int(len(lines) * 0.80)

    with open('../data/train.txt', 'w') as f:
        print('[info] saving 80% ({}) of dataset for training in ../data/train.txt'.format(split))
        f.write(''.join(lines[0:split]))

    with open('../data/test.txt', 'w') as f:
        print('[info] saving 20% ({}) of dataset for testing in ../data/test.txt'.format(len(lines) - split))
        f.write(''.join(lines[split:]))
```

Figura 14. Programma in python che divideva il dataset randomicamente.

4.1.5 Training Set

In questa fase dei test si è voluto mostrare come erano distribuite le password dei training set per avere una visione più globale del problema. Le analisi sono state effettuate con PACK, strumento già descritto nel paragrafo 3.2.2.

Per ogni dataset saranno mostrati due grafici, il primo grafico mostra la distribuzione della lunghezza delle password (come anticipato le password, come nel test-set, hanno lunghezza minore o uguale a 15), mentre nel secondo grafico si evidenzia, in percentuale, che tipo di grammatica/charset utilizzano. È chiaro che una limitata diversificazione di caratteri implica una password poco robusta.

- **1-milion**

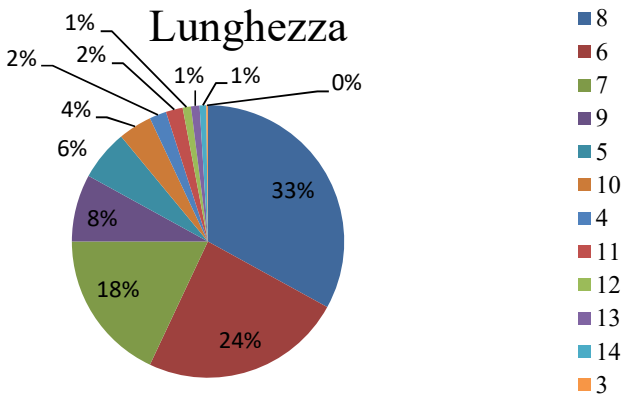

Grafico 1. Lunghezza caratteri training set 1-milion.

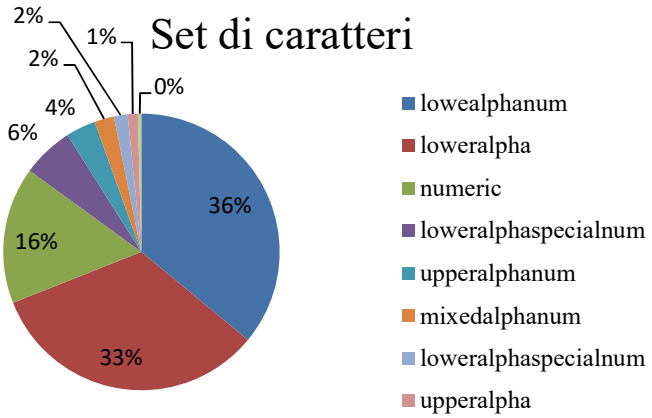

Grafico 2. Set di caratteri training set 1-milion.

- **Rockyou**

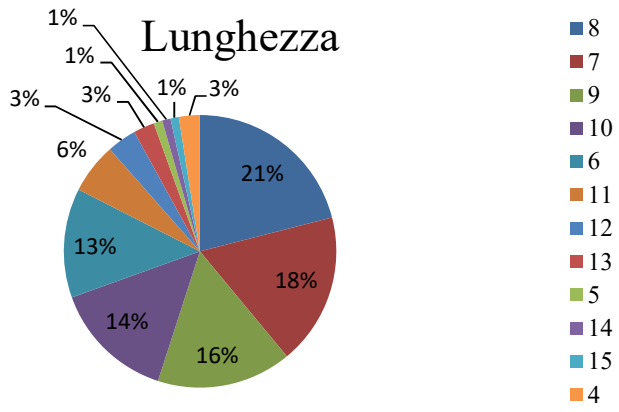

Grafico 3. Lunghezza caratteri training set rockyou.

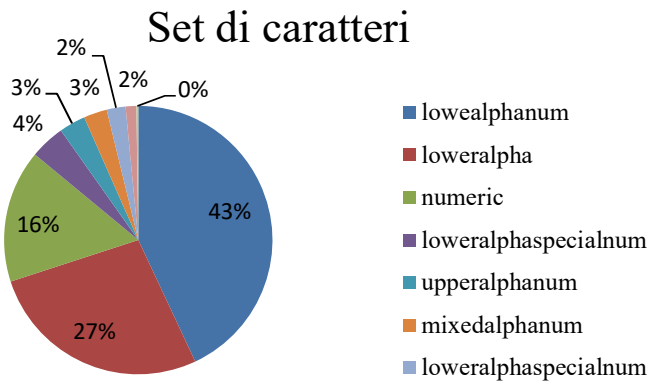

Grafico 4. Set di caratteri training set rockyou.

- **LinkedIn da mettere i dati giusti**

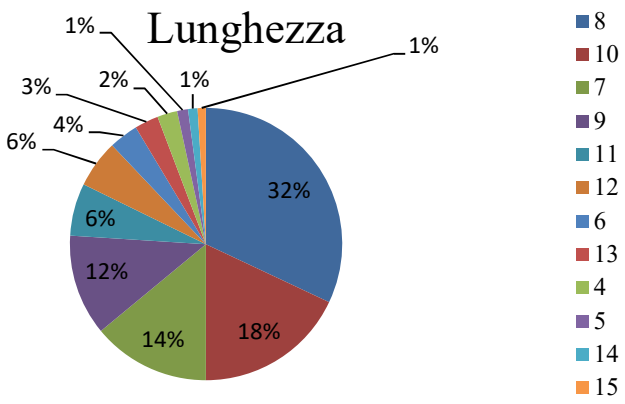

Grafico 5. Lunghezza caratteri training set LinkedIn.

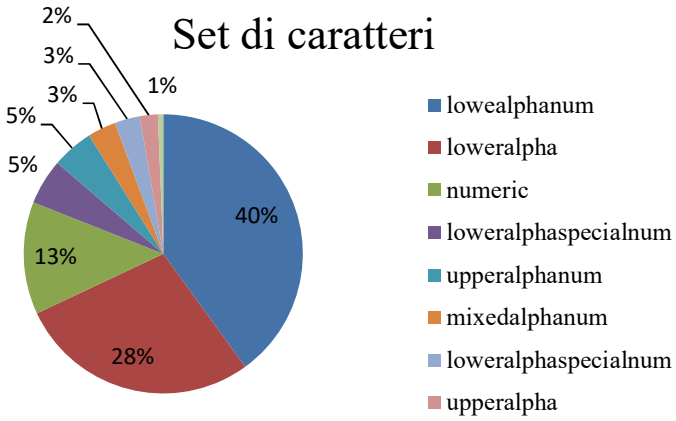

Grafico 6. Set di caratteri training set LinkedIn.

Queste analisi hanno messo in luce quali sono le caratteristiche più vulnerabili. In seguito saranno prese di mira tutte le password di lunghezza compresa tra 7 e 15 caratteri, per motivi di tempo, spazio e veridicità delle password: difficilmente viene richiesta come input una password inferiore agli 8 caratteri. Un'altra debolezza rilevata è la poca diversificazione dei caratteri usati, questo potrebbe essere un parametro implementativo, ma al fine dei test non sarà preso in considerazione.

4.2 Protocollo di testing

In questa fase si sono selezionati alcuni parametri per valutare se un attacco è migliore di un altro. Le metriche usate per misurare la bontà sono tre e sono utilizzate dai principali

lavori che trattano il password cracking. Per limitare ed allo stesso tempo per concentrare la forza di calcolo si è ristretto la creazione di password da 7 a 15 caratteri. Le sessioni dureranno svariate ore: 1, 3, 5, 7, 9. Alcuni strumenti non accetteranno un parametro temporale quindi si calcoleranno i tempi di impego. Gli strumenti avrebbero bisogno di più tempo, ma a causa della limitazione delle risorse si è optato per questi intervalli [45].

Le metriche sono le seguenti:

- **Numero di password craccate**: è una metrica classica che da un risultato numerico. Mostra il numero di password di una sessione di cracking che è riuscito a recuperare. Questo numero è difficilmente paragonabile ad altri risultati perché dovrebbe avere lo stesso numero di password tentato e/o tempo di utilizzo.
- **Numero di password craccate per password tentate**: o anche detto recovered, questo valore indica il rapporto tra le password craccate e quelle tentate. Più il rapporto si avvicinerà ad 1, maggiore sarà rilevante la sessione. In situazioni normali molto vicino a 0.

$$0 \leq \frac{\text{PASSWORDrecuperate}}{\text{PASSWORDtentate}} \leq 1$$

- **Numero di password craccate in relazione al tempo**: questo approccio è simile a quello visto precedentemente, ma si concentra maggiormente sull'avanzamento della sessione lungo lo scorrere del tempo invece che sul numero di tentativi. Permette di analizzare maggiormente la potenza del tool utilizzato per la seduta di cracking.

4.3 Descrizione degli esperimenti

Saranno presentati ora con maggior dettaglio gli esperimenti condotti. Questi test hanno il fine di identificare quali strumenti siano migliori per un attacco di password cracking. Le prove condotte si concentrano anche sul consumo di risorse: non tutti hanno a disposizione infrastrutture estremamente potenti, ma hanno potenza di calcolo nella "norma". In particolare tutte le verifiche sono state svolte su macchina virtuale Kali Linux o Google Colab. Il computer utilizzato aveva queste caratteristiche: 16 GB di RAM tipo: DDR4-SDRAM, processore Intel Core i7-1065G7 1,30GHz (4 core, 8 thread).

Kali Linux è una distribuzione Linux basata su Debian. Il sistema è indirizzato principalmente per l'ambiente della sicurezza informatica. L'utilizzo di Kali Linux era una scelta che era necessaria per i test da effettuare: molti dei software utilizzati risultano pre-installati, come Hashcat, e di facile utilizzo dal terminale. Per la VM di Kali Linux sono stati dedicati 8GB di RAM, 150 GB di hard disk e 2 processori. I processori e la RAM sono due parametri che influenzano i successivi calcoli. Gli esperimenti avranno un consumo molto alto e la scelta dei parametri sono un compromesso per la terminazione dei test. Attraverso Kali Linux si è potuto testare il PCFG-Cracker, PRINCEProcessor, Hashcat, OMEN e NEMO. PassGAN non è stato testato sulla macchina virtuale a causa dell'impossibilità di installare dei requirements obsoleti e varie incompatibilità.

La macchina virtuale messa a disposizione da Google Colab è stata di importanza rilevante con PassGAN. PassGAN è infatti una rete neurale molto pesante e che richiede svariate iterazioni per essere addestrata in modo corretto: molte iterazioni sono sinonimo di runtime e disponibilità hardware enormi. La VM di Google Colab mette a disposizione risorse variabili a seconda dell'impiego fatto nei giorni precedenti e dalla domanda di altri utenti. Nei test era possibile usare 12 GB di RAM, 75 GB di disco fisso e una GPU tra Nvidia K80s, T4s, P4s e P100s che era assegnata a seconda delle disponibilità.

È noto che l'alta capacità di calcolo di hardware come GPU (Graphics Processing Unit) o TPU (Tensor Processing Unit, scheda costruita da Google per essere impiegata con alcune reti neurali) possano incrementare notevolmente le prestazioni del nostro modello di Machine Learning. L'utilizzo di PassGAN su Colab è stato facilitato da alcuni file .ipynb, tipo di documento utilizzato da Google Colab, che si trovano sulla repository GitHub di PassGAN che svolgevano passo per passo la fase di train del modello e la fase di guessing generation.

In situazioni reali una sessione di password cracking può essere strutturata a "layer": utilizzando di volta in volta strumenti e idee diverse, al fine di craccare più password possibili per un determinato dataset. Ad esempio in certe situazioni un gruppo di password può essere più attaccabile con una determinata regola rispetto ad un altro gruppo: si tentano quindi diverse strade per arrivare a più password possibili. Questo documento si focalizzerà su un solo "layer" di questo ipotetico attacco, cercando di massimizzare in esso il recupero di password.

Gli esperimenti tratteranno solamente attacchi con strumenti innovativi, visti nel capitolo 3, l'intenzione del documento è appunto confrontare attacchi alle password all'attuale stato dell'arte. Come ampiamente spiegato per sviluppare queste offensive si utilizzeranno sistemi di password guessing uniti ad Hashcat. Una volta eseguiti gli strumenti si sono salvati i risultati su file. Successivamente questi sono stati dati in input ad hashcat con il seguente comando:

hashcat –potfile-disable inputDataset testSet

Il parametro –potfile-disable disabilita il potfile ovvero un file in cui Hashcat salva tutte le password recuperate da sessioni precedenti. Il primo controllo che Hashcat esegue una volta avviato è quello di controllare se alcune delle password target sono già state recuperate e contenute nel potfile. Il potfile va quindi disabilitato per evitare di falsare i test. E' estremamente utile in situazioni reali in cui si cerca man mano di craccare più password possibili da un dataset.

4.4 Analisi dei risultati

In questo paragrafo verranno illustrati i risultati ottenuti dai test. Nelle schermate di Hashcat saranno indicate alcune voci che andremo a spiegare il significato:

- **Speed**: indica la velocità di hashing di Hashcat (tipicamente indicata in kH/s o MH/s) ovvero la velocità con cui quest'ultimo sta praticando l'hash alle password in ingresso.
- **Recovered**: indica le password recuperate da Hashcat fino a quell'istante.
- **Remaining**: indica le password non ancora recuperate da Hashcat fino a quell'istante.
- **Recovered/Time**: indica una stima del tempo necessario al recupero di tutte le password.
- **Progress**: indica tutte le password generate e testate fino a quel momento.

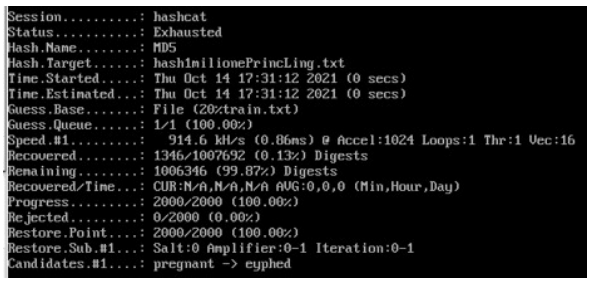

Figura 15. Schermata di fine esecuzione di Hashcat.

4.4.1 PCFG-Cracker

Comando generico per il Training:

```
python3 pcfg_cracker/trainer.py –r RULESET_NAME -t TRAIN_SET_PATH -e latin-1
```

In questa fase di train verranno generate le regole utili al secondo script python del PCFG (RULESET_NAME). L'opzione -e latin-1 indica il charset da utilizzare in questa fase per riconoscere le parole in input. Tutti i vocabolari sia di test sia di train utilizzano la codifica latin-1.

Comando generico per il Guessing + Cracking:

```
python3 pcfg_cracker/pcfg_guesser.py -r RULESET_NAME > pcfgPassword
```

In questa fase si genereranno effettivamente le password basate sulle rules identificate prima, che saranno inviate ad Hashcat tramite il file.

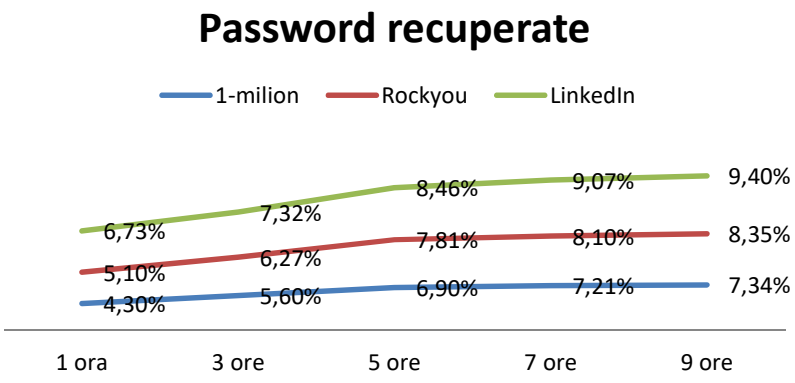

Grafico 7. Password recuperate PCFG-Cracker.

4.4.2 OMEN

OMEN essendo scritto in C, una volta installato deve essere compilato dal comando make. Per lanciare il comando che esegue il train bisogna scrivere:

```
./createNG --iPwdList password-training-list.txt
```

Per eseguire l'attacco OMEN permette sia di effettuare un attacco senza limiti e stampare password "all'infinito", oppure stampare un numero terminato di tentativi attraverso il parametro –m. Il parametro –p permette di avere la stampa:

./enumNG -p -m 10000

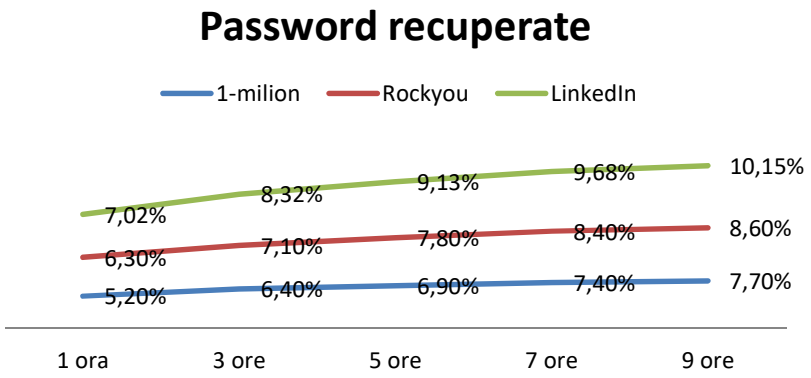

Grafico 8. Password recuperate OMEN.

4.4.3 NEMO

NEMO a differenza di OMEN è scritto in python, quindi i comandi pur essendo due programmi molto simili sono differenti. Il comando per eseguire la fase di train è il seguente:

python train.py

Qui il file di training lo prende in automatico senza doverlo passare come parametro. Invece per eseguire il comando di test, bisogna scrivere:

python meter.py

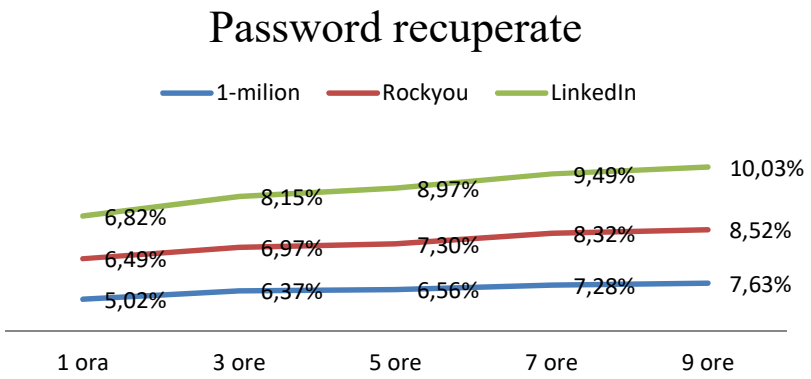

Grafico 9. Password recuperate NEMO.

4.4.4 PassGAN

La fase di training e di password guessing per PassGAN, come è stato anticipato nei paragrafi precedenti è stato utilizzato Colab. Nel capitolo precedente si sono spiegate le limitazioni e le modifiche fatte al codice per permettere l'esecuzione, queste modifiche saranno tenute in considerazione in fase di valutazione.

Qui non abbiamo un comando specifico visto che il file .ipynb è un insieme di istruzioni che vanno eseguite a blocchi. Inoltre PassGAN non aveva come parametro il tempo di utilizzo, quindi verranno calcolati i tempi di inizio e fine per poter fare un confronto con gli altri strumenti.

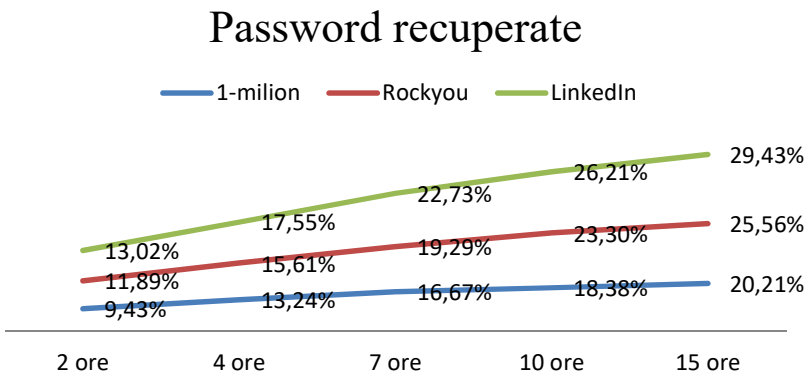

Grafico 10. Password recuperate PassGAN.

4.4.5 PRINCEProcessor

PRINCEProcessor ha un solo comando per train, guessing e cracking. La facilità d'uso del PRINCEProcessor si ritrova anche nel comando da lanciare per far partire l'attacco.

/princeprocessor-0.22/pp64.bin TRAIN_SET_PATH

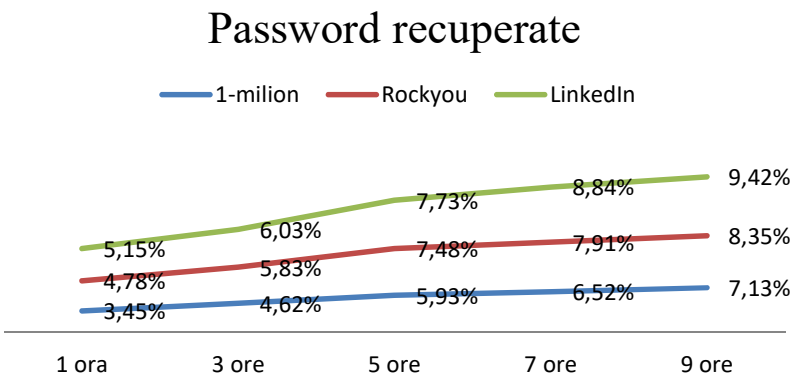

Grafico 11. Password recuperate PRINCEProcessor.

4.4.6 Hashcat

Hashcat, oltre come strumento di analisi e cracking delle password è stato utilizzato come generatore di dizionari. Questo strumento è molto duttile se affiancato ad altri o se si hanno a disposizione dei file di regole generati ad hoc. Per usare questo strumento bisogna eseguire il comando:

```
hashcat --force dataset -r /usr/share/hashcat/rules/best64.rule
```

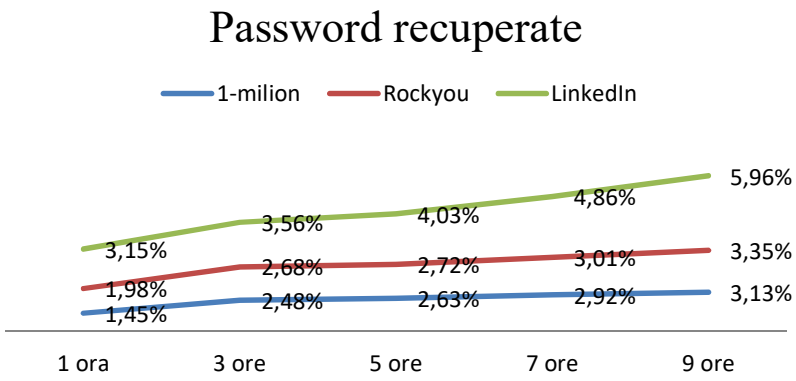

Grafico 12. Password recuperate Hashcat.

Capitolo 5

5 Conclusioni

Durante l'esecuzione di questo progetto sono stati testati strumenti allo stato dell'arte del password cracking in situazioni nuove, dovuti alla limitazione dell'hardware, rispetto a quanto visto nei paper citati. Oltre ai test sui singoli tool si è portato avanti lo sviluppo di un software che potesse estrapolare dai lavori l'idea e poterla rielaborare. Il software in questione per motivi di tempo non è ancora terminato ma ha avuto eccellenti, in termini di tempo e spazio di regole create, risultati in fase di training.

I test che sono stati eseguiti sono stati realizzati in situazioni il più reale possibile: cioè replicando quello che un potenziale attaccante potrebbe dover affrontare. I dati che sono il risultato del lavoro sono stati limitati da vincoli come risorse hardware e risorse di tempo. Sono stati provati vari dizionari per capire quale effettivamente fosse il migliore e quanto incide la diversificazione in termini di grandezza e provenienza di dati per il password cracking. Quello che è risaltato, è che sicuramente uno strumento che ha a disposizione un dataset più ampio avrà percentuali di risultato più alte.

Esistono però strumenti, tra quelli analizzati, che possono più o meno bene sfruttare i dati in ingresso. Si è visto come strumenti con implementazioni diverse abbiano risultati molto simili. I risultati su PassGAN hanno rivelato come questo sia un attacco non praticabile da chiunque, vedi il tempo e soprattutto l'utilizzo continuato di risorse. Il tool PRINCEProcessor ha dimostrato come uno strumento dal punto di vista tecnico meno complicato possa essere un valido strumento nel mondo del password cracking. Una nota per quanto riguarda i risultati di Hashcat, sono influenzati da un insieme di regole troppo generico e quindi non adatto ai dataset che sono stati sottoposti.

L'obiettivo di questa tesi era quello di analizzare nuove tecniche nel contesto del password cracking e sviluppare un nuovo software. I risultati ottenuti sono in media ai risultati citati nei paper consultati, quindi l'obiettivo di confronto tenendo conto: all'hardware e con il tempo a disposizione e ai dataset utilizzati si può considerare soddisfacente. Questo conferma come i dati, come le password, delle persone siano una cosa che va tutelata con molta cura, scelta e cambiata frequentemente.

5.1 Sviluppi futuri

Per quanto riguardo allo sviluppo del software, continuare a portare avanti il progetto di un nuovo tool di password cracking, cercando di migliorare la raccolta di strumenti a disposizione.

Per quanto riguarda la parte dei test migliorare i test sotto l'aspetto del tempo e la diversificazione dei dataset e dei requisiti hardware. Ampliare la visione e non limitarsi agli strumenti visti, esplorando nuovi progetti in questo ambito. Per quanto riguarda le possibili implementazioni che si potevano realizzare in questo progetto:

- Sperimentare rules di ottimizzazione per capire quale porti maggiori benefici ad una sessione di password cracking.
- Per capire nello specifico quali password vengono scoperte da un dato strumento si potrebbero ricondurre i test con il potfile di Hashcat attivo.
- Migliorare l'utilizzo di Hashcat come generatore di dizionari in particolare nella creazione di rules specifiche.
- Sperimentare su dataset reali gli strumenti citati.
- Migliorare le prestazioni tenendo in considerazione password di ogni lunghezza.
- Sviluppo di regole che riescano a trovare modifiche non ancora rilevate, come deviazione morfologiche (diminutivi e vezzeggiativi).

Capitolo 6

6 Ringraziamenti

Ringrazio la professoressa Rebecca Montanari per la possibilità che mi ha dato per effettuare questo tirocinio e la disponibilità per redarre la tesi. Un altro ringraziamento vorrei farlo a Marco Canducci, il mio referente in Cyberloop, per la sua professionalità ed impegno dimostrato nei mie confronti durante il tirocinio.

Grazie alla mia famiglia che mi ha sostenuto durante il mio percorso universitario nelle difficoltà e nei momenti felici. In particolare vorrei ringraziare mia mamma: Sandra che ci è sempre stata per qualsiasi cosa ed ha appoggiato tutti i miei passi ed ha creduto in me fin dall'inizio. Mio padre: Ciro che con la sua spensiaratezza mi fa rilassare nei momenti di stress. Alle mie sorelle: Concetta e Silvana che hanno sempre avuto parole e gesti gentile verso di me fin da piccolo. A mio fratello: Roberto che mi fa fare delle avventure roccalmbolesche. A miei cognati: Fabio e Francesco che sono degli ottimi consiglieri e compagni di giochi. A Samuele e Giorgia i due piccoli della famiglia che tra le gite in fattoria e partite allo stadio o alla playstation mi fanno rilassare.

Un altro grande ringraziamento va fatto ai miei amici, da quelli che mi hanno accompagnato fin dalle elementari come Valerio, a quelli che ho incontrato durante il percorso scolastico come Alessandro e Davide.

Un ringraziamento ai miei compagni di "avventura" universitaria Andrea, Lorenzo, Alan e Jacopo: tra progetti, attese ai binari, ansie pre esami, corse per prendere bus irraggiungibili e molto altro.

Grazie a tutti i miei amici extra scolastici: dal gruppo la Crew del garage, Maials, Punkipa, Pubetto.

Bibliografia

[1] Menezes A. J. Oorschot P. C. V. Vanstone S. A. Rivest R. L. Handbook of Applied Cryptography, CRC Press., 1997.

[2] Shirey, R. RFC 2828: Internet Security Glossary, The internet Society 13, 2000.

[3] S. Marechal "Advances in password cracking.," Journal in computer virology, vol. 4, no. 1, pp. 73-81., 2008.

[4] Klein, D.V. Bishop M. Improving system security via proactive password checking. Computers & Security, Volume 14, Issue 3, Pages 233-249., 1995.

[5] E. Landwehr, Computer security. International Journal of Information Security. Volume 1, Issue 1, pp 3-13., Springer, August 2001.

[6] Min, Luo Ninghui Li. Jerry Ma Weining Yang. A study of probabilistic password models. In 2014 IEEE Symposium on Security and Privacy, pages 689–704. IEEE,, 2014.

[7] Qiu, Wei-Dong Li Jian-Hua. Meng Kui Liu Gong-Shen Password vulnerability assessment and recovery based on rules mined from large-scale real data. Chinese Journal of Computers, 39(3):454–467, 2016.

[8] Paar., Daniel V Bailey Markus Dürmuth Christof. Statistics on password re-use and adaptive strength for financial accounts. In International Conference on Security and Cryptography for Networks, pages 218–235., Spinger, 2014.

[9] Xinyi, Huang Gaopeng Jian. Ding Wang Haibo Cheng Ping Wang. Zipf's law in passwords. IEEE Transactions on Information Forensics and Security, 12(11):2776–2791, 2017.

[10] Elizabeth Stobert and Robert Biddle. The password life cycle: user behaviour in managing passwords. In 10th Symposium On Usable Privacy and Security (SOUPS 2014), pages 243–255, 2014.

[11] R. Shay, S. Komanduri, P. Kelley, P. Leon, M. Mazurek, L. Bauer, N. Christin and L. Cranor, Encountering stronger password requirements: User attitudes and behaviors, Proceedings of the Sixth Symposium on Usable Privacy and Security, article no. 2, 2010.

[12] M. Durmuth, F. Angelstorf, C. Castelluccia, D. Perito and A. Chaabane, OMEN: Faster password guessing using an ordered Markov enumerator, Proceedings of the Seventh International Symposium on Engineering Secure Software and Systems, pp. 119–132, 2015.

[13] Li, Z. Xie Z. Zhang M. Yin A. A New Targeted Password Guessing Model. In: Liu J., Cui H. (eds) Information Security and Privacy. ACISP 2020. Lecture Notes in Computer Science,

vol 12248., Springer, Cham. https://doi.org/10.1007/978-3-030-55304-3_1, 2020.

[14] Ding Wang, Ping Wang, Debiao He, and Yuan Tian. Birthday, name and bifacial security: understanding passwords of chinese web users. In 28th USENIX Security Symposium (USENIX Security 19), pages 1537–1555, 2019.

[15] Claude Castelluccia, Abdelberi Chaabane, Markus Dürmuth, and Daniele Perito. When privacy meets security: Leveraging personal information for password cracking. arXiv preprint arXiv:1304.6584, 2013.

[16] "I Added '!' at the End to Make It Secure": Observing Password Creation in the Lab Blase Ur, Fumiko Noma, Jonathan Bees, Sean M. Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor Carnegie Mellon University.

[17] https://it.wikipedia.org/wiki/Attacco_informatico.

[18] https://it.wikipedia.org/wiki/Hacking.

[19] https://www.giorgiosbaraglia.it/attacco-alle-password-tecniche-di-cracking-e-consigli-per-metterle-al-sicuro/.

[20] https://hacktips.it/metodologie-password-cracking/.

[21] https://it.wikipedia.org/wiki/Robustezza_della_password.

[22] https://hashcat.net/hashcat/.

[23] https://hashcat.net/wiki/doku.php?id=mask_attack.

[24] Wiedenbeck, S., Waters, J., Sobrado, L. and Birget, J.C., 2006, May. Design and evaluation of a shoulder-surfing resistant graphical password scheme. In Proceedings of the working conference on Advanced visual interfaces (pp. 177-184). ACM.

[25] Wendy Goucher. Look behind you: the dangers of shoulder surfing. Computer Fraud & Security, 2011(11):17–20, 2011..

[26] https://www.sicurezzanazionale.gov.it/sisr.nsf/archivio-notizie/nuova-edizione-del-glossario-intelligence.html.

[27] A. P. Ratna, P. D. Purnamasari, A. Shaugi, and A. Salman, "Analysis and comparison of md5 and sha-1 algorithm implementation in simpleo authentication based security system," in Proceedings of IEEE International Conference on QiR (Quality in Research), Yo.

[28] [8] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in Proceedings of Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, 2003, pp. 617–630..

[29] Hellman, M. (1980). "A cryptanalytic time-memory trade-off" (PDF). IEEETransactions on Information Theory. 26 (4): 401–406. CiteSeerX 10.1.1.120.2463.

doi:10.1109/TIT.1980.1056220. ISSN 0018-9448..

[30] K. Theocharoulis, I. Papaefstathiou, and C. Manifavas, “Implementing rainbow tables in high-end fpgas for super-fast password cracking,” in Proceedings of International Conference on Field Programmable Logic and Applications (FPL), 2010, pp. 145–150..

[31] Philippe Oechslin’s. rainbowcrack. url: https://gitlab.com/kalilinux/ packages/rainbowcrack..

[32] Markus Duermuth, Fabian Angelstorf, Claude Castelluccia, Daniele Perito, Abdelberi Chaabane. OMEN: Faster Password Guessing Using an Ordered Markov Enumerator. International Symposium on Engineering Secure Software and Systems, Mar 2015, milan, Italy. ffh.

[33] Nong Ye, Yebin Zhang and C. M. Borror, "Robustness of the Markov-chain model for cyber-attack detection," in IEEE Transactions on Reliability, vol. 53, no. 1, pp. 116-123, March 2004, doi: 10.1109/TR.2004.823851..

[34] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using timespace tradeoff. In Proc. 12th ACM conference on Computer and communications security (CCS), pages 364–372. ACM, 2005..

[35] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In 2009 30th IEEE Symposium on Security and Privacy, pages 391–405. IEEE, 2009..

[36] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pages 1242–1254, 2016..

[37] Jens Steube et al. Princeprocessor main page, 2019. Available on line..

[38] PassGAN: A Deep Learning Approach for Password Guessing. Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, Fernando Perez-Cruz.

[39] Improved Training of Wasserstein GANs. Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville.

[40] Jens Steube et al. Princeprocessor simple tutorial on hashcat’s forum, 2015. Available on line..

[41] ON THE ACCURACY OF PASSWORD STRENGTH METERS. Maximilian Golla, Markus Dürmuth. ACM Conference on Computer and Communications Security 2018 (CCS '18). Toronto, Canada, October 15-19, 2018.

[42] Improved Training of Wasserstein GANs Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville.

[43] • PassGAN: A Deep Learning Approach for Password Guessing. Briland Hitaj, Paolo Gasti,

Giuseppe Ateniese, Fernando Perez-Cruz.

[44] https://github.com/iphelix/pack/tree/622dd925bfd320d57700b1dc9b39318658adfed0.

[45] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In 2009 30th IEEE Symposium on Security and Privacy, pages 391–405. IEEE, 2009..