UNIVERSIDAD
DE GRANADA

Password Guessing Programs y la seguridad detrás de las contraseñas Administración y Seguridad de Sistemas

Silvio Andrés Orozco Vizquerra 299537005

Mayo de 2022

Índice

Introducción

Las contraseñas han sido el método predominante para autenticar a usuarios y acceder a recursos restringidos. Estas pueden llegar a ser un problema ya que la seguridad depende en gran medida de la calidad y fuerza de la contraseña que ha escogido el usuario y a su vez en menor medida las restricciones y condiciones que tiene el sistema en general para aceptar una contraseña como válida. Aún más si la contraseña es débil, puede llegar a ser muy sencillo para una tercera persona adivinar y descifrar la contraseña pretendiendo ser el usuario en cuestión. Especialmente, cuando sabemos que los humanos somos el eslabón más flojo de la seguridad de la información y que tendemos a escoger contraseñas fáciles de recordar. Cabe mencionar también que desde la explosión del internet en los 90s se han utilizado las contraseñas junto con el proceso de hashing (convertir un string de caracteres en un código numérico que representa el string original) creado por Wilkes y mejorado por Robert Morris permitiendo el storage de las mismas en bases de datos. Hoy en día se han creado otros métodos como salting para encriptar las contraseñas pero eso no ha detenido el avance de distintos métodos para el cracking y guessing de una contraseña. Entre ellos podemos resaltar ataques por fuerza bruta, ataques basado en diccionario, utilizando ingeniería social con un usuario, buscar un fichero que contenga las claves, preguntar a un usuario simulando un login, utilizando rainbow table attacks, entre otros. En este trabajo se procederá a detallar más sobre estos métodos y se realizará un breve análisis sobre la seguridad de una contraseña.

Desarrollo

Historia de Usable Privacy and Security Research y las Contraseñas

Para hablar sobre la historia de las contraseñas debemos comenzar en el desarrollo de Usable Privacy and Security Research (UPS). Entre los años de 1975 y 1995, Saltzer y Schroeders fueron los primeros en observar que para que un sistema informático sea usable debe ser seguro. En su paper identificaron "Aceptabilidad Psicológica" como unos de los 8 principios para construir sistemas que protejan la información. Este principio indicaba que las interfaces humanas deben ser diseñadas para que sean de fácil uso y que de esta forma los usuarios apliquen métodos de protección correctamente. Todo esto extendiendo la imagen mental del usuario haciendo que sus metas de protección encajen con los mecanismos que debe usar reduciendo así al mínimo los errores que se puedan cometer (Garfinkel y Lipford, 2014).

Más tarde, en 1979 Robert Morris y Ken Thompson, que trabajaban en Bell Laboratories, publican su paper "Password Security: A Case History" que describe la historia del diseño del esquema de seguridad basado en contraseñas. Se explica que unas de las metas más claras de las contraseñas es dar seguridad a los usuarios con la mínima inconveniencia posible. Además, el sistema de contraseñas no solo debe prevenir el acceso no autorizado de usuarios a los sistemas pero también debe prevenir a los usuarios que ya han iniciado sesión de hacer cosas que no están autorizados a hacer. Cabe mencionar también que hablan de la importancia del problema matemática detrás de las contraseñas como de los algoritmos de encripción pero que en realidad un solo un pequeño pedazo de todo el

problema en donde se encuentra también la seguridad física, las comunicaciones de seguridad, las acciones de empleados y exempleados que conocer el sistema por dentro y por último, también no solo tomar en cuenta los ataques deliberados sino accesos no autorizados casuales o revelaciones accidentales (Morris y Thompson, 1979).

Los personajes anteriores desarrollaron el primer sistema de contraseñas que se basa en que nunca se guarde el texto plano de una contraseña. Se basa en encriptar la contraseña por medio de un algoritmo y guardar solo este texto encriptado, de forma que cada vez que el usuario intente iniciar sesión se encripta la contraseña utilizada de nuevo y comparar si existe en el fichero de contraseñas. Si se hace un match, el intento de login es aceptado. Lo más importante detrás de este esquema es encontrar una algoritmo de encripción difícil de invertir aún como este algoritmo sea público y esté disponible a todos. En este caso de estudio de 1979, ya se hablaba de los posibles métodos para atacar este esquema de seguridad desde ataques de fuerza bruta y ataques por diccionario que se discutirán más adelante. Añadido a esto ya se sugerían algunas mejoras al esquema tales como un algoritmo de encripción lento, mayor dificultad al usuario para que se esfuerce en su contraseña y salted passwords que consisten en agregar 12 random bits al hash de la fecha que se crea la contraseña para que sea más difícil para el atacante multiplicado por un factor de 4096 (2^{12}).

Entre 1995 y 2005, nace realmente la UPS como "User Centered Security" por parte de Zurko y Simon, que basaba la seguridad en que los usuarios entendiesen cómo funciona intentando resolver los problemas de UPS al momento de diseño con los usuarios. Desde

entonces UPS research se ha basado en mejorar la seguridad de la autenticación de usuarios buscando progreso en la fiabilidad y seguridad de las contraseñas y otros métodos de autenticación.

La historia de Privacidad y Seguridad Usable (UPS) nos dan contexto para lo que ha sucedido en la historia con las contraseñas. Los usuarios y contraseñas fueron inventados simultáneamente alrededor de 1960 por el Massachusetts Institute of Technology (MIT) como parte del proyecto Compatible Time Sharing System (CTSS), y por IBM para controlar accesos al sistema Sabre de reservaciones. En 1969, Hoffman observó por primera vez que era fácil comprometer a los sistemas por medio de un "systematic password guessing attack" y que en 1979, como hablamos anteriormente Morris y Thompson encontraron que 2831 de 3289 contraseñas podían ser descubiertas a través de un "offline dictionary attack". En 1985, el Departamento de Defensa de Estados Unidos publicó en su "Green Book" el "Password Management Guideline" en donde se establecieron los requerimientos de las contraseñas para las próximas 3 décadas. Algunas de las pautas importantes a mencionar es dar un tiempo de vida a las contraseñas (un año para la mayoría de los casos y un mes para sistemas de alta protección), permitir un máximo de 3 intentos de contraseña antes de que una alarma sea generada y proponer tamaños de entre 6 y 8 caracteres. Ahora se han agregado otras reglas tales como requerir un mix de letras minúsculas y mayúsculas, símbolos y contraseñas más largas (Garfinkel y Lipford, 2014).

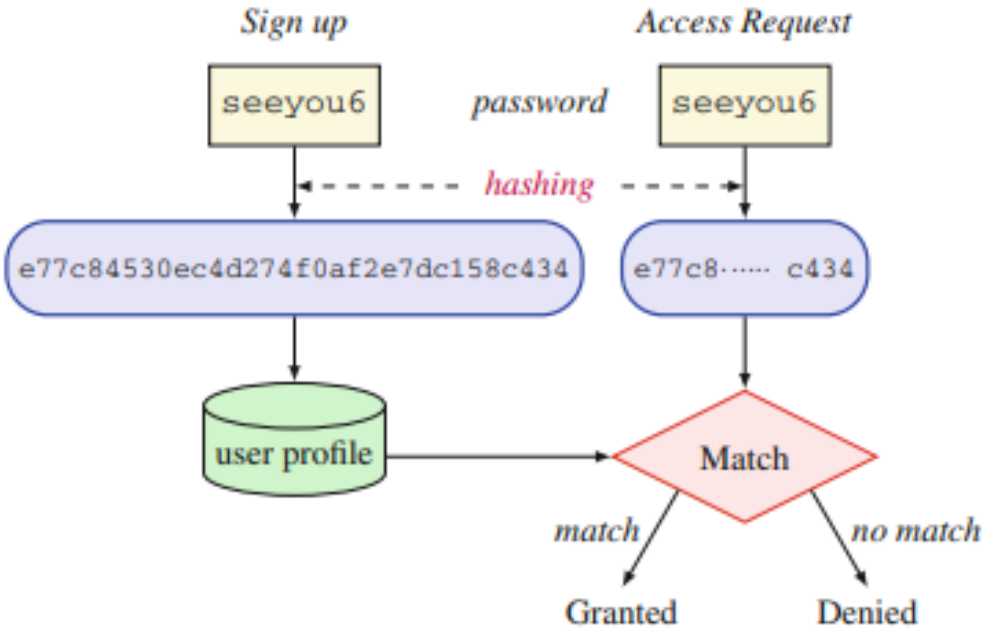
En 2013, William Cheswick publicó un artículo llamado "Rethinking Passwords", en el cual explica que varias de las reglas mencionadas anteriormente ya no son válidas. Esto lo ahondaremos después en la parte de escoger contraseñas seguras. Hoy en día, como bien sabemos las

contraseñas siguen y seguirán siendo parte de nuestras vidas por la facilidad de despliegue y simpleza, por lo que permanece importante mejorar las técnicas contra los ataques y educar a los usuarios sobre el correcto uso y elección de las mismas.

¿Cómo funciona una contraseña?

El concepto de contraseña en sistemas de seguridad y computación lo podemos definir como un string de carácteres secretos que solo el usuario conoce, de forma que al ser pasado por un algoritmo de encriptación hash y generando un hash code pueda ser guardado en en el servidor para proveer acceso a data. De forma que se utiliza tanto una contraseña como otra forma de identificación del usuario tales como nombre de usuario o un correo electrónico (Hu, 2017). Puede observarse el proceso de forma gráfica en la Figura 1.

Figura 1: Mecanismo de contraseñas para acceder a un recurso (Hu, 2017)



Es relevante indicar que si se hace un match en la base de datos con el hashcode y el user profile entonces el acceso es permitido y de lo contrario, denegado. La función es hash es una función irreversible que mapea una contraseña p a un código cifrado h de la forma $f(p) = h$ y que f^{-1} es inexistente, por lo que la única

manera de que un hacker acceda a la contraseña es intentando con diferentes strings viendo si alguno hace match con el código hash de la contraseña actual. De forma que es posible descifrar la contraseña para un ataque offline teniendo el código hash y siendo imposible para un ataque online teniendo una simple regla de 3 fallos en la contraseña bloqueando temporalmente el acceso al usuario necesitando mayor verificación (Gu, 2017). Con lo anterior, se puede intuir que el punto esencial de una contraseña es entonces que sea difícil para un hacker adivinar la contraseña desde un hash code.

Cracking una contraseña

Cracking a password es la técnica de infringir la seguridad de un sistema para encontrar la contraseña de un usuario y asumir su identidad para ingresar al sistema (Cazier y Medlin, 2006). Existen diferentes ataques para una contraseña y se puede definir el riesgo sobre los siguientes tres factores (Yannis, 2013):

1. El tiempo que toma al atacante computar y probar una contraseña
2. La cantidad de recursos que tiene el atacante
3. La probabilidad de éxito del método del atacante

De igual forma, existen 2 ataques principales:

- Online attacks: Ataques que son realizados en vivo a un host y sistema y que comúnmente no son posibles por varios mecanismo de protección tales como Capcha y máxima cantidad de intentos. Se puede utilizar herramientas como Hydra para estos ataques que soporta POP3, HTTP,IMAP, entre otros.
- Offline Attacks: Estos ataques se realizan luego de tener el o los hashes de las contraseñas y posterior a esto intentar "to crack" la contraseña en una máquina remota sin alertar al "target host". Los ataques descritos debajo están centrados en ser offline.

Los psicólogos han investigado que por lo general, las contraseñas las creamos con algo que es familiar para nosotros, por lo que exploramos diferentes técnicas para adivinar una contraseña desde un offline attack tomando en cuenta que será más fácil si un usuario le da intrascendencia a la seguridad de sus contraseñas.

Brute Force Attack

Los ataques de fuerza bruta se caracterizan por realizar una búsqueda exhaustiva calculando el hash de todas las posibles combinaciones de strings para un conjunto de caracteres y de una longitud m. En teoría, tiene un éxito del 100% aunque es un problema no computable por la cantidad de tiempo y cómo crece exponencialmente a medida que la contraseña es más larga. Finalmente, es ineficiente porque se prueba grandes cantidades de combinaciones poco probables como contraseña (Hu,2017).

Dictionary Attack

Los ataques de diccionario son útiles al pensar que los usuarios utilizan contraseñas humanamente memorizables por lo que existen palabras y variaciones comunes que un hacker puede intentar en lugar de usar combinaciones random como en el ataque de fuerza bruta. Los problemas con esta técnica es que el diccionario debe contener una vocabulario "enorme o completo" y que los usuarios que conocen este ataque realizan pequeños cambios a la palabra lo cual lo hace más difícil para este ataque. Aunque ya bien si un usuario utiliza contraseñas simples que están en el diccionario, el cracking de la contraseña será igualmente muy sencillo con esta técnica (Hu, 2017). También existen técnicas que combinan los diccionarios sabiendo que los usuarios comúnmente combinan varias palabras para hacer más seguras sus contraseñas.

Hybrid Dictionary Attack

Estos ataques combinan los ataques por diccionarios más un poco de fuerza bruta al adjuntas pequeños strings comunes al diccionario tales como caracteres numéricos y símbolos. Este ataque muchas veces sigue

siendo inviable por la cantidad de tiempo que tomaría y depende en gran medida de la cantidad de caracteres que se adjunta (Yannis, 2013).

Rule based Dictionary Attack

Este ataque se caracteriza por ser igual que el hybrid dictionary attack pero agrega reglas sobre adjuntar carácteres o palabras en el diccionario dependiendo de contraseñas comunes. Por lo tanto, el éxito del ataque depende en gran medida de que tan buenas pueden llegar a ser las reglas que tenemos que podrían reducir los tiempos para descifrar una contraseña.

Free Public Hash Databases and Table Lookup Attack

Para este ataque puede ser simple buscar bases de datos de hashes públicos y comunes que es parecido al ataque de fuerza bruta solo que simplemente sin el tiempo de computar el hash sumado a que solo se realizan las combinaciones muy sencillas. Este método es efectivo si el usuario tiene una contraseña muy sencilla de una palabra común pero no garantiza el éxito (Yannis, 2013). Al no realizar el cálculo del hash es mucho más rápido que un ataque de diccionario.

Rainbow Table Attack

Rainbow Table Attack es una variación del Table Lookup Attack. Se basa en la técnica de Time-Memory Trade-Off (TMTO) que busca guardar una cantidad mucho menor de hash codes que aún representando una cantidad enorme de contraseñas. La idea es crear una cadena de password-hash de longitud k que cubre k potenciales contraseñas y sus hash codes (Hu, 2017). Para optimizar los hashes se guardan únicamente el primera y última en cada cadena a memoria. Utilizando el mismo espacio de memoria, cubre k más palabras utilizando el mismo

diccionario pero k veces más lento que un ataque de diccionario. El mayor problema de este método es las colisiones cuando dos diferentes hash son reducidos a la misma contraseña y que la función reducción se comporte correctamente. También que necesitamos una base de datos tan grande que no llega a ser realista y para combatir este método también se creó el proceso de salting lo cual convierte esto totalmente inusable porque contraseñas idénticas resultados con un diferente hash cada vez pero es útil para sistemas que no han incorporado salting y contraseñas menores de 10 dígitos (Yannis, 2013). Algunos ejemplos de software que utilizan esta técnica son Crackstation, Hashkiller y Hashcat.

Markov Model Attack

Al reconocer que los usuarios prefieren contraseñas fáciles de recordar, se puede pensar en diccionarios inteligentes con contraseñas que son probables que los usuarios puedan generar. Narayanan y Shmatikov introdujeron este método utilizando el modelo de Markov para generar este diccionario inteligente. Se crea el diccionario basado en la probabilidad de un carácter en una secuencia dado el lenguaje nativo del usuario. Estos modelos reducen drásticamente el espacio de búsqueda y se crean diccionarios útiles y probables para descifrar contraseñas con efectividades mayores al 60% (Hu, 2017).

Attack with Probabilistic Context-Free Grammar

Estos ataques se basan en estimar la probabilidad de la contraseña de un usuario a partir de un training set de contraseñas reales filtradas y crear un context-free grammar que pueda estimar la formación de un string (Hu, 2017).

John the Ripper Attack

John the Ripper es un software abierto para descifrar contraseñas que combina ataques de diccionarios, rainbow tables y ataques de fuerza bruta con una variedad de reglas para la composición de palabras (Hu, 2017).

Artificial Intelligence Attack

Este acercamiento es el más novedoso, dado que se ha probado que con inteligencia artificial y un set de contraseñas filtradas de más de 43 millones de perfiles de LinkedIn se ha logrado descifrar más de un cuarto de las contraseñas (Hutson, 2017). PassGan usa Generative Adversarial Network GAN (deep learning) para que de forma autónoma aprenda la distribución real de contraseñas de password leaks y generar adivinanzas de contraseñas de alta calidad logrando mejorar en gran medida los métodos originales de descifrado de contraseña (Hitaj, Gatsi, Ateniese y Perez-Cruz, 2019). Este proyecto se basa en utilizar deep learning de forma que se puedan generar muestras de la misma distribución del training set.

¿Cómo escoger contraseñas seguras?

Para analizar si una contraseña es segura o no, Hu propone medidas de la calidad de una contraseña. Estas medidas pueden ser las siguientes (2017):

- Entropía: Introducida por Claude Shannon y definida como el número promedio de dígitos binarios requeridos por cada letra del lenguaje original. El National Institute of Standards and Technology (NIST) proveyó un valor para esta métrica expresando la incertidumbre de una contraseña expresada en número de bits indicando que una mayor incertidumbre se tiene una contraseña más difícil de adivinar. Para una contraseña w de tamaño m de un conjunto de caracteres de tamaño N, la entropía se define como:

  $$H(w) = log_2(N^m) = m * log_2(N)$$

  Estimando entropías de la siguiente forma:

  - +4 Por el primer carácter
  - +2 Por cada carácter entre el 2ndo y el 8vo
  - +1.5 Por cada carácter entre el 9no y el 20vo
  - +6 Por usar mayúsculas y caracteres no alfabéticos
  - +6 No en diccionarios

- Password Quality Indicator PQI: Aún estando consciente de los ataques de diccionario, las personas tienden a utilizar palabras y realizar pequeños cambios para recordarlo. Por ello, no es solo importante considerar que las contraseñas no estén en diccionarios sino también qué tan fáciles o difíciles son estos "pequeños cambios". La métrica

que se utiliza es la distancia Levenshtein que es el mínimo número de operaciones para transferir una palabra a otra. El PQI se basa en la idea anterior, consistiendo en una contraseña w con un par D (distancia Levenshtein) y L que la longitud efectiva de p. Se define con la siguiente fórmula dónde m es el tamaño de la contraseña y N el tamaño del conjunto de caracteres posibles.

$$L = m * log_{10}(N)$$

Se considera una contraseña segura si D es mayor o igual a 3 y si L es mayor o igual a 14.

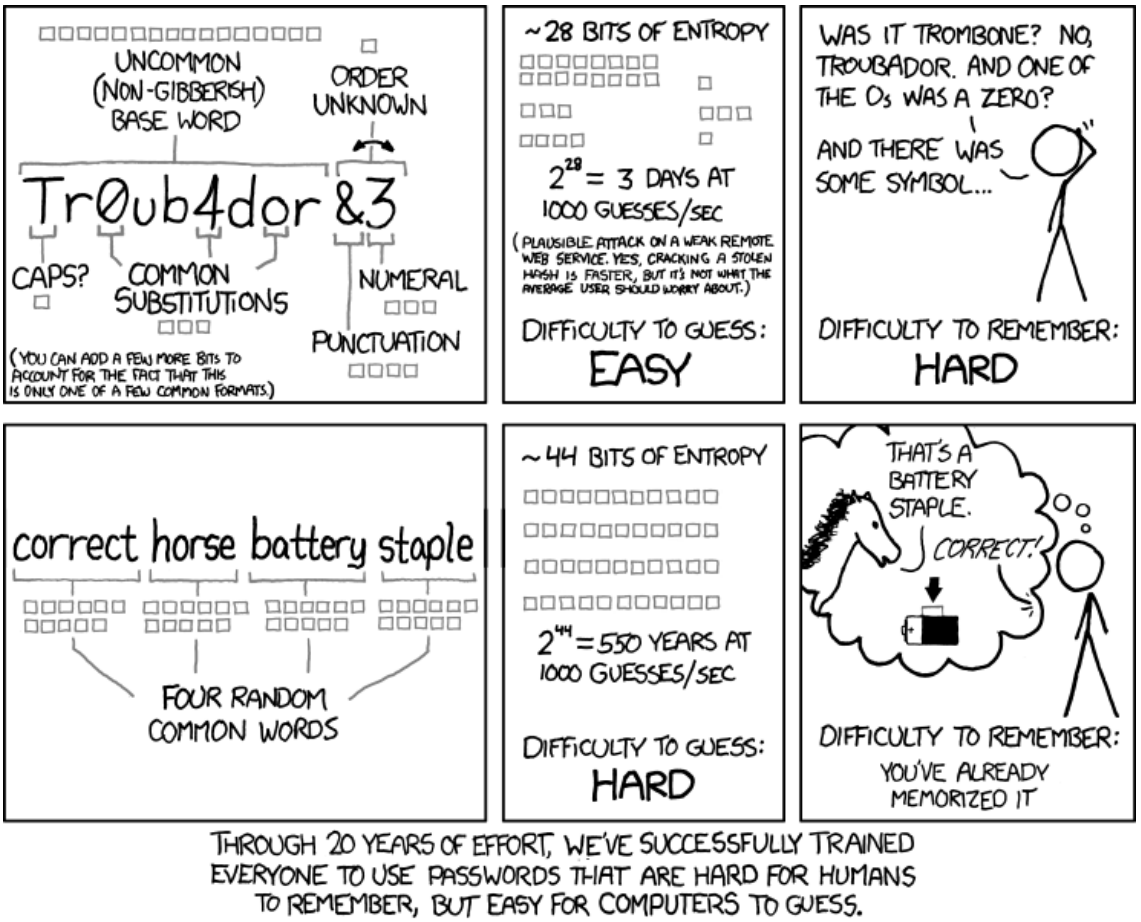
- Otras métricas por vendedores de servicio: Algoritmos que utilizan empresas tales como Dropbox, que utiliza un "zxcbn" que estima la seguridad de la contraseña buscando que tan común es la contraseña por ejemplo en conjuntos de contraseñas filtrados o palabras comunes en wikipedia.
- Cazier y Medlin definen en la figura 2 un índice y evaluación basado en las mejores prácticas sobre las contraseñas de los usuarios.

Figura 2: Clasificación de escala basado en la contraseña del usuario

Level	Description	Criteria
1	Obvious name or number.	1. Password contains all or part of a username, email, name, etc. 2. Or is somehow related to known information, such as a spouse or a child's name. We may not have enough information to always assess this. 3. It is easily guessable by someone that knows a little about the individual, based on publicly available knowledge.
2	Common word or number	1. Password is a repeating number (i.e., 99999) 2. Or it is a common word recognizable as either a name or word likely to be found in a dictionary?
3	Words with number at start or end or non-repeating numbers	1. Example, John52 or 73horses, combining recognizable words or phrases with a number. 2. Or having a nonrepeating number, such as 345820, that does not appear to be a social security number, phone number, or street address. 3. Or adding a compound word, such as lovejohn or toomuchfun. 4. Or unrecognizable alpha characters that did not form a common word or name (i.e., mtlyfll).
4	Unrecognizable alpha and number combinations	1. Has a mixture of unrecognizable alpha characters with numbers thrown in (i.e.jds932). 2. May include compound words with numbers lovejohn99.
5	Special characters	1. Numbers and letters mixed with the inclusion of special characters like #, %, and, @ etc. 2. Spaces and dashes are included in this category.

Conociendo lo anterior, debemos considerar que no solo es importante los requisitos y las métricas que puedan brindar las plataformas y sistemas que necesiten contraseñas sino también se necesitan usuarios conscientes de lo que es una buena contraseña. Como se ilustra en la figura 3, de Randal Munroe que en los últimos 20 años nos hemos enfocado en utilizar contraseñas que son difíciles de recordar para los humanos y contrariamente fáciles para las computadoras de adivinar.

Figura 3: Randall Munroe's XKCD 936



Por último, junto con las indicaciones del Green Book de 1985 y los pensamientos de Cheswick (2013) sobre "Rethinking Password", podemos establecer las siguientes recomendaciones como válidas para contraseñas seguras:

- No utilizar la misma contraseña en múltiples servicios (Que una se filtre compromete todas tus demás plataformas) .
- No escribir tus contraseñas o guardarlas en archivos (Los ataques pueden venir de coempleados o personas cercanas o ya bien si tienen acceso a tus dispositivos podrían encontrar tus contraseñas).

Password wallets pueden ser un problema si no son utilizadas correctamente.

- Utilizar multiple factor authentication
- Eye-of-newt password rules: One time passwords y “Don’t be a Moron Rule” (No escoger contraseñas fáciles que un amigo o un familiar puedan adivinar con unos cuantos intentos)
- Utilizar como contraseñas una mezcla de palabras y varios caracteres diferentes como se muestra en la figura 3.

Caso Práctico

En 2009, la red social RockYou sufrió una filtración de contraseñas que afectó a 32 millones de usuarios, especialmente porque RockYou guardaba todos los datos de los usuarios incluidas las contraseñas como texto plano exponiendo toda esta información a los hackers. Fue un total caos ya que no solo tenían políticas malas para el manejo de contraseñas sino también manejaron la situación terriblemente con sus usuarios.

En 2021, en un foro de hackers se filtró el archivo rockyou2021.txt haciendo referencia al incidente anterior. Este archivo de 100GB contenía más de 8.4 billones de entradas de contraseñas combinadas de diferentes filtraciones.

Como parte del caso práctico se desarrolló un programa en python que pudiese medir la seguridad de una contraseña y sobre una base de datos de un poco más de 90 GB de palabras recopiladas de diferentes listas de contraseñas comunes para realizar un ataque por diccionario, resultando en 82 billones de entradas únicas encontradas en el repositorio de github de OhMyBahGosh de RockYou2021.txt (2022).

El programa passwordCheck.py contiene lo siguiente:

```
# Program for Checking you Password against Password Leaks

from getpass import getpass
import time
import math
import progressbar

# First we ask user password to evaluate
while True:
    password = getpass("Enter Password to evaluate: ", "*")
    password_confirm = getpass("Confirm Password to evaluate: ", "*")
    if(password!=password_confirm):
        print("Passwords to check didn't match. Please try again.")
        continue
    break



print("We start the evaluation of your password. Please wait.")
found_password_in_dict = 0

#  We  take  the  time  that  it  takes  to  check  all  password  in  the
dictionary assuming
start = time.time()
filePasswords = open("rockyou2021.txt", "r")

password_found = ""
#We do the dictionary attack
#We assume in this case we have the plain text password already and not
the hashes
i=0
bar = progressbar.ProgressBar(max_value=progressbar.UnknownLength)
BUF_SIZE = 1024
TOTAL_SIZE = 0
tmp_lines = filePasswords.readlines(BUF_SIZE)
TOTAL_SIZE = TOTAL_SIZE + BUF_SIZE
while tmp_lines:
    for line in filePasswords.read(1024):
        passw = line.rstrip()
        if(passw==password):
            found_password_in_dict = found_password_in_dict + 1
```

```
            password_found = x
        time.sleep(0.1)
        i=i+1
        bar.update(i)
    TOTAL_SIZE = TOTAL_SIZE + BUF_SIZE
    tmp_lines = filePasswords.readlines(BUF_SIZE)
print("\n")
if(found_password_in_dict>0):
          print("Password has ben found in leaked dictionary.
{}".format(found_password_in_dict))
                  print("Password Found was {} in {}
entries".format(password_found,i))
else:
    print("Password was not found in dict and it will most likely pass
a dictionary attack in a total of {} entries checked.".format(i))

end = time.time()
time_in_s = end - start
print("Total time in seconds to check {}".format(time_in_s))

# Characters available for a password
# Numbers (10 different: 0-9)
# Letters (52 different: A-Z and a-z)
# Special characters (32 different).
# Total 94

# m size of possible caracters 94
N = 94
# m size of password
m = len(password)

# Entropy
entropy_of_password = math.log(N,2)*m
print("Entropy of password {}".format(entropy_of_password))
# Nist Guidelines entropy
entropy_nist = 0
entropy_nist = entropy_nist + 6 if m>0 else entropy_nist
entropy_nist= entropy_nist + (m-1)*2 if m>1 and m<=8 else (entropy_nist
+ 7*2 if m>8 else entropy_nist)
entropy_nist= entropy_nist + (m-8)*1.5 if m>8 else entropy_nist
ntropy_nist= entropy_nist + 3 if any(character.isupper() for character
in password) else entropy_nist
```

```
entropy_nist= entropy_nist + 3 if not password.isalnum() else
entropy_nist
entropy_nist= entropy_nist + 6 if not found_password_in_dict else
entropy_nist
print("Entropy Nist of Password {}".format(entropy_nist))

# Calculate Effective L of Password, L is Good if its more than 14
L = m * math.log(N,10)
print("L value of password (safe if more than 14) {}".format(L))
```

El programa como vemos utiliza rockyou2021.txt para evaluar la calidad de nuestra contraseña junto con los indicadores mencionados anteriormente. Puedo llegar a ser muy tardado por evaluar más de 8 billones de registros y también hay que considerar que no hacemos el hash de las contraseñas dado que ya están dadas en plain text y esto agregaría un factor más grande de tiempo por cada contraseña revisada.

El código puede encontrarse en el siguiente repositorio:

https://github.com/sorozcov/PASSWORDCHECK

Referencias

Cazier, J. A., & Medlin, B. D. (2006). *Password Security: An Empirical Investigation into E-Commerce Passwords and Their Crack Times.* Information Systems Security, 15(6), 45–55. doi:10.1080/10658980601051318

Cheswick, William (2013). *Rethinking passwords.* Communications of the ACM, 56(2), 40–. doi:10.1145/2408776.2408790

Garfinkel, S., & Lipford, H. R. (2014). *Usable Security: History, Themes, and Challenges. Synthesis Lectures on Information Security, Privacy, and Trust*, 5(2), 1–124. doi:10.2200/s00594ed1v01y201408spt011

Hitaj, B., Gasti, P., Ateniese, G., Perez-Cruz, F. (2019). *PassGAN: A Deep Learning Approach for Password Guessing.* In: Deng, R., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds) Applied Cryptography and Network Security. ACNS 2019. Lecture Notes in Computer Science(), vol 11464. Springer, Cham. https://doi.org/10.1007/978-3-030-21568-2_11

Hu, G. (2017). *On password strength: a survey and analysis.* In International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (pp. 165-186). Springer, Cham.

Hutson, M. (2017). *Artificial intelligence just made guessing your password a whole lot easier.* Science. Recuperado de https://www.science.org/content/article/artificial-int

elligence-just-made-guessing-your-password-whole-lot-easier

Morris, R., & Thompson, K. (1979). *Password Security: A Case History.* Commun. ACM, 22(11), 594–597. doi:10.1145/359168.359172

Lee, R. (2018). *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing.* doi:10.1007/978-3-030-26428-4

OhMyBahGosh. (2022). RockYou2021.txt. Github Repository. https://github.com/ohmybahgosh/RockYou2021.txt#:~:text=About-,RockYou2021.,compiled%20of%20various%20other%20wordlists.

Orozco, S. (2022). *PasswordCheck*. Github Repository. https://github.com/sorozcov/PASSWORDCHECK

WeLiveSecurity. (2017). *A short history of the computer password.* Recuperado de https://www.welivesecurity.com/2017/05/04/short-history-computer-password/

Yannis, C. (2013). *Modern Password Cracking: A hands-on approach to creating an optimized and versatile attack* (Tesis de Master). University of London.