

|  |
| --- |
| **Fortifying Password Security: Exploring the Effectiveness of Hashing, Salting and Alternative Authentication Methods**  **Author:**  **David Dandie – 2103490@uad.ac.uk**  Introduction to Security – CMP110  BSc Ethical Hacking Year 1  2022/23 |

Abstract

In today's increasingly online world, passwords are more important than ever to keep personal and sensitive information secure. However, individuals and corporations often fail to follow safe password practices, leading to data breaches and theft. This report examines effective password practices, including password hashing and salting, and their ability to withstand dictionary attacks. The study demonstrates the effectiveness of both techniques by hashing and cracking 100,000 of the most used passwords using the SHA-1 algorithm and the popular cracking tool hashcat, and then salting them before cracking them again. The results show that using both hashing and salting together dramatically increases password security. Additionally, this report discusses other best password practices, such as password rotation and the use of password managers, and explores other methods of user authentication, including two-factor authentication and biometrics, concluding that a combinational approach of authentication variety and robust password practices is the most effective way of increasing cyber security.

Contents

[1 Introduction 1](#_Toc134699551)

[1.1 Background 1](#_Toc134699552)

[1.2 Aim 2](#_Toc134699553)

[2 Procedure 3](#_Toc134699554)

[2.1 Overview of Procedure 3](#_Toc134699555)

[2.2 Procedure part 1 4](#_Toc134699556)

[2.3 Procedure part 2 6](#_Toc134699557)

[3 Results 8](#_Toc134699558)

[3.1 Results for part 1 8](#_Toc134699559)

[3.2 Results for part 2 9](#_Toc134699560)

[4 Discussion 10](#_Toc134699561)

[4.1 General Discussion 10](#_Toc134699562)

[4.2 Limitations 10](#_Toc134699563)

[4.3 Countermeasures 10](#_Toc134699564)

[4.4 conclusion 11](#_Toc134699565)

[4.5 Future work 12](#_Toc134699566)

[References 13](#_Toc134699567)

[Appendices 14](#_Toc134699568)

[Appendix A 14](#_Toc134699569)

[Appendix B 14](#_Toc134699570)

# Introduction

## Background

Passwords are a common method used for authentication and access control to a wide range of systems, from online accounts to corporate networks. However, as the number of online services continue to grow, and the amount of personal information stored in digital form continues to grow, the importance of a strong and secure password has become more crucial than ever. However, individuals and companies still fail to incorporate safe password practices, this can lead to security breaches, data theft and financial loss.

Information provided by csoonline (Ragan.S,2023) shows that in 2013 Adobe suffered a major data breach effecting over 130 million user accounts. This breach was due to the company using Triple DES encryption rather than the industry standard hashing algorithms. The difference between encryption and hashing is that encryption does a good job of scrambling plain text into cipher text but the problem with encryption is that the cipher text can be reverted to plain text with the correct key.

A hash function is a mathematical algorithm that converts plain text into a fixed length string of characters called a hash. Hashing is useful for securely storing passwords without storing the actual passwords in plain text. When a user creates an account or changes their password the hash value of their password is stored in a database, when they log in their password is then hashed again and compared to the stored hash in the database. This is a one-way system and once hashed will never be readable in plain text again. One issue with hashing is that the output is always the same as it is a calculation, but any modification regardless of how big or small produces a completely different output.

Figure A shows the hashed value for *hello world.*

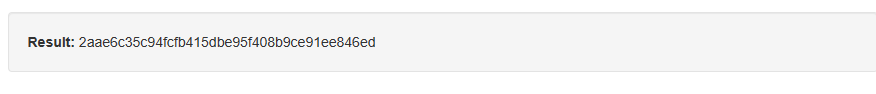
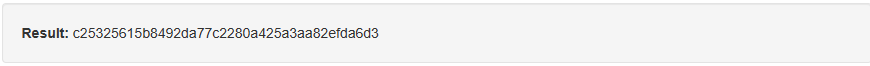


Figure B shows the hashed value for *hello world1.*



Although hashing provides a secure way to store passwords it is not impenetrable and can be cracked using a variety of attacks, most commonly being either brute force or dictionary attack. A brute force attack consists of the attacker trying all possible combinations of characters. This approach is both time and resource heavy. A dictionary attack, the attacker will use a generated list of commonly used passwords to guess the password. There is also a rainbow table attack where an attacker will use a pre generated table of hashes to quickly crack passwords. The problem is once a hash is cracked for a certain word or combination of words it is now vulnerable to attack and severely compromises password security.

Information from ICO (2023) shows that in 2012 social networking site LinkedIn suffered a massive data breach which resulted in 165 million users having their email addresses and passwords sold on the dark web. And although the company did use a hashing algorithm to store their passwords the passwords were unsalted. Salting a password is crucial to password security, A salt is a random string of characters added to the password before it is hashed, resulting in a unique output for each password even though they are the same. That means that even if an attacker obtained the hashed password, they could not use precomputed tables or easily guess the patterns using common words or patterns.

## Aim

This report will delve into the best password security practices, including hashing and salting and evaluate their effectiveness against password cracking.

* Investigate best password security practices, including hashing and salting and their effectiveness against password cracking.
* Test the effectiveness of password salting practices by hashing and cracking 10000 of the most used passwords, then salting and re-cracking them potentially demonstrating the importance of robust hashing and salting password practices.

# Procedure

## Overview of Procedure

The purpose of these procedures is to investigate the importance of salting passwords for increased security. Two procedures were conducted to compare the effectiveness of hashed passwords and salted passwords against dictionary attacks.

The following tools were used for the procedures,

1. Password text file: containing 10000 of the most used passwords (Miessler.D, 2019).
2. Rockyou.txt: A popular dictionary file containing over 14 million commonly used passwords this is used in the cracking process. (Marcel.J, 2021)
3. Hashcat: a well-known password cracking tool was used to perform dictionary attacks in both procedures. (Hashcat,2023)

The procedural setup involved the following steps,

1. Cracked password storage: Created a cracked.txt file inside the hashcat directory, this will serve as an output file to store cracked passwords.
2. Hashing Program: The second step was to create a hashing program using python. The program Reads in each of the 10000 passwords from the passwords.txt file and runs each password through the SHA-1 algorithm and stores it into hashedpasswords.txt file to be used later.
3. Cracking hashed passwords: Once the hashed passwords were generated the next step was to use hashcat to crack the hashed passwords using a dictionary attack, the rockyou.txt file was used at this point as a dictionary file to aid in the cracking process.
4. Salting program: The fourth step was to create a salting program using C++, this program works by reading each password in from the password.txt file and adding a varying length between 4-8 ASCII characters in random order and adding it as a string to the beginning of each password, it is then stored in salted.txt ready to be rehashed using the SHA-1 algorithm.
5. Cracking salted passwords: The final step was to run the salted passwords through the hashing program to rehash them and store them in hashedandsalted.txt the same process was used as before performing a dictionary attack using hashcat with the rockyou.txt as the dictionary file.

Overall, the procedural setup involved creating two programs, one for salting passwords and one for hashing passwords and using hashcat to crack them. The procedure was set up to demonstrate the difference in security between hashed and salted passwords.

## Procedure part 1

1. Execute the hashing program to convert each plain text password from password.txt into a SHA-1 hash and store it in hashedpasswords.txt, (See figure 1.1).

Text

Description automatically generated

Figure .1 displays the python program solution that will encrypt the passwords.txt file using the SHA-1 hashing algorithm and export the encrypted passwords to hashedpasswords.txt. (See appendix A for full code).

1. Insert the hashedpasswords.txt file into the hashcat directory.
2. Run hashcat from command prompt, note that due to the lack of Graphical user interface a directory address must be entered.
3. Apply syntax to allow hashcat to perform a dictionary attack using rockyou.txt as primary reference, (See figure 1.2) to display how console should display.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1.2 shows the directory address and syntax required to perform a dictionary attack using rockyou.txt. Note that the directory address will vary depending on where the hashcat file is stored.

1. Open cracked.txt file in hashcat directory and record results, (See figure 1.3).

Text

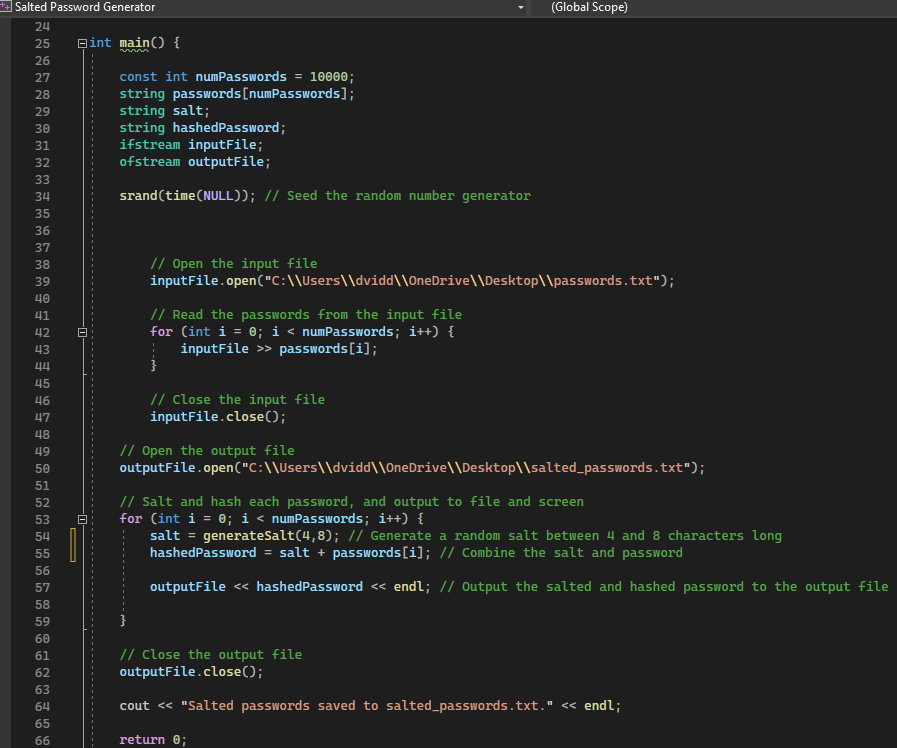
Description automatically generated

**Figure 1.3** shows the figure of passwords that were cracked inside cracked.txt.

1. Open the hashcat POTFILE inside the hash cat directory and erase its contents to allow the same passwords to be cracked again.
2. Delete the contents of cracked.txt file in the hash cat directory to remove previously cracked passwords and save.
3. Remove hashedpasswords.txt file from the hash cat directory and delete its contents for future use.

## Procedure part 2

1. Execute the salting program to import passwords from password.txt and export salted passwords to salted\_passwords.txt, (See figure 2.1).



**Figure 2.1** displays the salting program that will dynamically add between 4 and 8 random ASCII characters to the beginning of each password. (See appendix B for full code).

1. Run the hashing program again, importing from saltedpasswords.txt and exporting salted and hashed passwords to saltedandhashed.txt, (See figure 2.2).

Text

Description automatically generated

**Figure 2.2** displays the hashing algorithm with updated addresses.

1. Place the saltedandhashed.txt into the hashcat directory.
2. Run hashcat and perform a dictionary attack on saltedandhashed.txt using rockyou.txt as the dictionary file, (See figure 2.3).

A black screen with white text

Description automatically generated with low confidence

**Figure 2.3** displays the syntax required to run a dictionary attack on the new combined salted and hashed passwords.

1. Record the results of the dictionary attack on the salted and hashed passwords.

# Results

## Results for part 1

The results show that of the 10000 hashed passwords, 9671(96.71%) of the passwords were cracked using the dictionary attack method in hashcat and it took a total time of 4 seconds (See figure 3.1). Demonstrating the vulnerability of hashed only passwords to dictionary attacks and highlights the Importance of additional security measures such as password salting.

Text

Description automatically generated

**Figure 3.1** shows the success rate ‘recovered**’** of the dictionary attack in hash cat with hash only password.

## Results for part 2

The results show that from the 100000 salted and hashed passwords hashcat was unable to crack any of the combined salted and hashed passwords (See figure 3.2), the program cycled through the entire 14 million entries of the dictionary file and did not find a single match. Demonstrating the strength of a salted and hashed password and highlighting the importance of the salting process in password security.

Text

Description automatically generated

**Figure 3.2** shows the success rate ‘recovered**’** of the dictionary attack in hash cat with hashed and salted passwords.

# Discussion

## General Discussion

At the outset of this report, it was noted that hashing passwords provides a more secure method of password authentication than cypher encryption, because hashing is a one-way function. However, the results of these procedures clearly show that hashing passwords alone is not a fool proof solution, as they can still be attacked using rainbow table or dictionary type attacks. The results also demonstrate that using salt, which involves adding a random string of characters to a weak password, significantly improves its strength and resilience against these common and efficient types of attacks. Therefore, providing evidence that using both methods in conjunction with each other (hashing and salting) is best practice for password security.

## Limitations

While the procedures used in this report provide valuable insights into password security, it is important to note that there are limitations to their scope. One limitation is the effectiveness of the hashing algorithm used. The National Institute of Standards and Technology (2022) has announced plans to retire the SHA-1 algorithm by Dec 31st, 2030, due to the growing ability of modern hardware and software to crack the algorithm with relative ease. This highlights the need to replace it with more robust hashing functions such as SHA-2 and SHA-3.

Another limitation to consider is the strength of passwords used in these procedures. The passwords tested were selected from a list of 10000 of the weakest most used passwords publicly available, if more robust passwords where used the results of the procedure may have been different.

It is also important to note that only one type of attack was used in these procedures, using a relatively small dictionary file, other types of attacks, such as brute force, hybrid, and combinational attacks may have produced different results. Therefore, the results of this report should be interpreted with caution and considered within the context of these limitations.

## Countermeasures

To enhance password security and deter simple attacks, users can adopt better password practices such as selecting a strong password. According to data from the National Cyber Security Centre (2019), 23.2 million people worldwide still use “123456” as a password. Therefore, users should use a combination of characters, numbers, upper case, and lower case to ensure best password practices are followed.

password rotation can also significantly improve chances against security breaches. The Daily Mail (2019) reported that 40% of UK residents never rotate their password, which can be problematic. Changing passwords every 30 days can also allow for more robust password practices.

Using a password manager can also improve password security. One of the main reasons individuals select simple, easy to guess passwords is due to memory. With the increasing number of accounts needed for day-to-day activities online, users tend to have the same password over multiple platforms and accounts and tend to keep them simple, creating potential for security breaches. A password manager will generate and store more complex passwords for the user allowing for more robust password practices.

Furthermore, Biometric authentication is becoming increasingly popular for validating user authentication due to its numerous benefits. The NCSC (2021) shows that one of the main advantages is that it eliminates the problem of losing or forgetting passwords. Additionally biometric authentication ensures that no two authentications can be the same and the risk of a false authentication is around 1 in a million. However, it is worth noting this was a statistic for apple devices and may not be an accurate statistic for other manufactures. Overall, biometrics provides a safe and robust alternative to passwords.

The NCSC (2021) also shows that two- factor authentication is also gaining popularity as an addition in the user authentication process. This additional security feature offers multiple benefits, such as providing an extra layer of protection to a user’s account, it does this by sending a pin or a code to another device owned by the user via text message or email. Some organizations also offer independent devices in the form of key fobs or readers for that extra layer of security.

Overall, while hashing and salting passwords can significantly improve password security it’s important to acknowledge their limitations and to adopt additional measures such as using strong passwords, password rotation, password managers as well as biometric and two-factor authentication. Organizations and individuals should adopt these practices to ensure overall better cybersecurity.

## conclusion

Passwords are an essential aspect of cyber security and user authentication, and it is important to implement strong and secure password practices to protect sensitive data and prohibit unwanted access. Hashing and salting are an effective method for securely storing passwords and preventing them from being accessed in plain text, however as demonstrated in this report even with these practices in place passwords can still be vulnerable to attacks such as dictionary and rainbow table attacks, these attacks don’t need much guessing for a password cracking tool to crack as its essentially only comparing values. Therefore, it is essential to continually update and improve password practices to stay in front of potential threats.

The results of the procedures performed in this report highlight the importance of implementing robust password security practices using a combination of hashing and salting to protect against password cracking. Additionally, there are now more secure alternative methods to passwords available such as biometrical and 2FA authentication providing an even more secure user authentication process. Adopting a combination of all the discussed best password practices individuals and corporations can improve their effectiveness at overcoming cyber threats and enhancing overall cyber security.

## Future work

***Conduct a larger scale study to determine the state of password security within a population and the potential benefit of using alternative password security methods such as password managers.***

1. Assess current password security practices within a population: The initial stage of the study will involve surveying a group within the population at random to ascertain the current state of password security within the group. Methods will include creating a questionnaire with questions relating to password complexity, reuse, and knowledge of best password practices.

By analyzing the survey responses quantitative data can be produced on the percentage of people using weak password practices and give quality insight into the level of knowledge individuals have regarding best password security practices.

1. Evaluate the impact and beneficial use of password managers: following the data gathered from the initial stage of the study the second phase will involve the same group of participants from stage 1 receiving training on the importance of using password managers and the correct methods of creating and managing strong passwords. The participants will then be asked to adopt the use of a password manager to secure their accounts. Quantitative data can then be gathered on the uniqueness, complexity and overall password strength and compared to the data gathered from phase 1 of the study.

Overall, the study will aim to provide a clear understanding on the beneficial impact adopting the use of a password manager can have on overall password security, by comparing data from before and after its use research can be used to potentially provide evidence that password managers are an effective tool for both password education and security purposes.

# References

Writer, S.R. and S. (2013). *Adobe confirms stolen passwords were encrypted, not hashed*. [online] CSO Online. Available at: <https://www.csoonline.com/article/2134124/adobe-confirms-stolen-passwords-were-encrypted-not-hashed.html> [accessed : 02 May 2023]

‌*ICO (no date). Passwords and online services. [online]*. Available at: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/security/passwords-in-online-services/?q=privacy+noticeshttps%3A%2F%2Fico.org.uk%2Ffor-organisations%2Fguide-to-the-general-data-protection-regulation-gdpr%2Findividual-rights%2Fright-to-be-informed%2F%3Fq%3Dprivacy+notices> [Accessed: 02 May 2023]

danielmiessler (2019). *danielmiessler/SecLists*. [online] GitHub. Available at: <https://github.com/danielmiessler/SecLists> [accessed: 02 May 2023]

C, J.M. (2021). *rockyou.txt*. [online] GitHub. Available at: <https://github.com/josuamarcelc/common-password-list>. [Accessed: 02 May 2023]

‌Hashcat.net. (2018). *hashcat - advanced password recovery*. [online] Available at: <https://hashcat.net/hashcat/> [accessed: 02 May 2023]

NIST Retires SHA-1 Cryptographic Algorithm. (2022). *NIST*. [online] Available at: <https://www.nist.gov/news-events/news/2022/12/nist-retires-sha-1-cryptographic-algorithm> [accessed: 03 May 2023]

National Cyber Security Centre (2019). *Most hacked passwords revealed as UK cyber survey exposes gaps in online security*. [online] Ncsc.gov.uk. Available at: <https://www.ncsc.gov.uk/news/most-hacked-passwords-revealed-as-uk-cyber-survey-exposes-gaps-in-online-security> [accessed: 03 May 2023]

Morrison, R. (2019). *People are sick of passwords and don’t change them enough, study finds*. [online] Mail Online. Available at: <https://www.dailymail.co.uk/sciencetech/article-7715101/People-sick-passwords-dont-change-new-research-finds.html> [accessed: 04 May 2023]

NCSC (2021). *Using Biometrics*. [online] www.ncsc.gov.uk. Available at: <https://www.ncsc.gov.uk/collection/device-security-guidance/policies-and-settings/using-biometrics> [accessed: 04 May 2023]

www.ncsc.gov.uk. (n.d.). *Turn on 2-step verification (2SV)*. [online] Available at: <https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online/activate-2-step-verification-on-your-email> [accessed: 04 May 2023]

# Appendices

## Appendix A

# This imports the hashing library

import hashlib

# This defines what hashing algorithm will be used

hash\_algorithm = hashlib.sha1

# Open the input file and read the passwords

file = open("C:\\Users\\dvidd\\OneDrive\\Desktop\\salted\_passwords.txt", "r", encoding="utf-8")

passwords = [line.strip() for line in file]

file.close()

# Hash each password and write the hashed passwords to a new file

file = open("C:\\Users\\dvidd\\OneDrive\\Desktop\\saltedandhashed.txt", "w", encoding="utf-8")

for password in passwords:

hashed\_password = hash\_algorithm(password.encode()).hexdigest()

file.write(hashed\_password + '\n')

file.close()

## Appendix B

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <chrono>

#include <thread>

using namespace std;

// Generate a random salt of length between minLength and maxLength

string generateSalt(int minLength, int maxLength) {

const char alphanum[] = "0123456789" "ABCDEFGHIJKLMNOPQRSTUVWXYZ" "abcdefghijklmnopqrstuvwxyz" "!\"#$%&'()\*+,-./:;<=>?@[\\]^\_`{|}~";

int saltLength = rand() % (maxLength - minLength + 1) + minLength;

string salt(saltLength, ' ');

for (int i = 0; i < saltLength; i++) {

salt[i] = alphanum[rand() % (sizeof(alphanum) - 1)];

}

return salt;

}

int main() {

const int numPasswords = 10000;

string passwords[numPasswords];

string salt;

string hashedPassword;

ifstream inputFile;

ofstream outputFile;

srand(time(NULL)); // Seed the random number generator

// Open the input file

inputFile.open("C:\\Users\\dvidd\\OneDrive\\Desktop\\passwords.txt");

// Read the passwords from the input file

for (int i = 0; i < numPasswords; i++) {

inputFile >> passwords[i];

}

// Close the input file

inputFile.close();

// Open the output file

outputFile.open("C:\\Users\\dvidd\\OneDrive\\Desktop\\salted\_passwords.txt");

// Salt and hash each password, and output to file and screen

for (int i = 0; i < numPasswords; i++) {

salt = generateSalt(4,8); // Generate a random salt between 4 and 8 characters long

hashedPassword = salt + passwords[i]; // combine the salt and password

outputFile << hashedPassword << endl; // Output the salted and hashed password to the output file

}

// Close the output file

outputFile.close();

cout << "Salted passwords saved to salted\_passwords.txt." << endl;

return 0;

}