Use of a One - Way Hash without a Salt or Predictable Salt

CS463 - Network Security Project Report

Submitted By

Awanit Ranjan
181CO161

Submitted To

Dr. Mahendra Pratap Singh
AssistantProfessor

Department of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal

April 2022

CONTENTS

CHAPTER 1

Introduction

World comprehensive web security is a set of procedures, policies, and technologies for ensuring the web servers, browsers, and other programs that communicate with the web servers and the internet infrastructure's reliability and predictability. Password and data transfer are secured using a variety of encryption algorithms, cryptographic protocols, and cryptology algorithms. With the emerging threat of losing privacy and data-stealing activity, securing the passwords stored in the database is required.

1.1 CWE-759 & 760's description

Common Weakness Enumeration (CWE) is a community-developed list of common software and hardware weakness types that have security ramifications. "Weaknesses" are flaws, faults, bugs, or other errors in software or hardware implementation, code, design, or architecture that if left unaddressed could result in systems, networks, or hardware being vulnerable to attack. The CWE List and associated classification taxonomy serve as a language that can be used to identify and describe these weaknesses in terms of CWEs as defined in [4].

Passwords were previously saved in plain text in the database. In today's world, storing the password in plain text is extremely dangerous. Storing passwords in plaintext puts them at danger of being stolen if the database is hacked. It could lead to a forged login and a breach of privacy. After that, it became common practise to store passwords as hashes. The practise employs a one-way cryptographic hash against a non-reversible input, such as a password. Nonetheless, as with **CWE-759** [1], the software does not use salt as part of the input. As a result, it makes the scenario prone to a Rainbow Table attack.

Computer and network security experts proposed using salt with the passwords to get around the rainbow table. **Salt** is a term used in cryptography to describe

the addition of some random data to input before hashing to make dictionary attacks more difficult [1][2]. As a result, the modified software employs a one-way cryptographic hash against non-reversible input data, such as a password. Despite this, the software uses a predetermined salt as part of the input. As a result, attackers can use dictionary attack techniques like rainbow tables to pre-compute the hash value, reducing the resilience that an unpredictable salt would give, as specified in **CWE - 760** [2]. However, the software uses salting to build a hash for a password. Nonetheless, it employs a technique that, according to **CWE-916** [3] , does not provide a sufficient amount of computational effort to render password cracking assaults infeasible or expensive. Assume that an attacker has access to the hashes. Due to the absence of significant computational effort, brute force attacks employing rainbow tables or specialized hardware like GPUs, which can compute hashes considerably quicker than general-purpose CPUs, will be easier.

1.2 Threat

A **threat** is a potential negative action or event aided by a vulnerability that results in an unwanted impact on a computer system or application in the context of computer security.

The Threat posed by CWE-759 and CWE-760 is related to Access Control. The compromise of password can expose users' secrecy, authorization, and authentication to serious threats. By stealing a password, an attacker can get over the security system and obtain access to sensitive information or assume another person's identity.

1.3 Organization of the report

The rest of the chapter is structured as follows. The chapter-2 presents a literature survey stating the latest status of the threat and different types of attacks performed, followed by some strategies proposed by the researchers to mitigate or detect the threat. Chapter 3 offers a method to tackle this common weakness, enumerations 759 and 760. Chapter 4 presents a light analysis of the proposed approach, followed by chapter 5, which concludes the work

CHAPTER 2

Literature Survey

This chapter comprises two main sections that present the current latest status, relevant cyber-attack performed for exploring the stated threat, and review research work related to strategies and techniques performed by users to mitigate the danger.

2.1 Latest status & Attacks

This section throws light on the current status of the threat. The typical authentication mechanism is via username and password. It is usual to keep hash values of passwords together with their related username in a database. Otherwise, an attacker can access the database and use the passwords to access the systems. Storing hash values would make it impossible for an attacker to perform this simple "password lookup." However, saving password hash values in a database does not provide security against the following attacks:

- A brute-force attack is possible where an attacker guesses all possible combinations of characters from a particular character set and passwords up to a certain length. For big character sets, however, this approach is unfeasible.
- An attacker can easily create a database of hash values for multiple randomly selected passwords, sort them, and compare them to the hash values in the password file. The generated database of hash values for passwords can then be used for comparing the hash values of multiple password files.
- A dictionary attack is a type of birthday attack in which passwords that are likely to be successful, such as dictionary words, are hashed as part of the preparation of the attacks. Dictionary attacks have a generic complexity similar to that of birthday attacks. When there are many entries in the password files and a large number of such files, the precomputed birthday and dictionary attacks to find passwords may become effective.

- Birthday type attacks will not work if the goal is to find the password of a specific user; instead, preimage attacks with a generic complexity of 2t are needed to find the password of a particular t-bit hash value.
- A Rainbow Table is a pre-computed table used to store the output of hash functions, which is commonly used to crack password hashes. Tables are generally used to recover a key derivation function with a limited number of characters up to a certain length.
- SQL injection attack occur when data from an untrusted source enters a program.
  - Due to the sensitive nature of SQL databases, loss of privacy is a common issue when SQL Injection vulnerabilities are exploited.
  - Bad SQL statements check user names and passwords; it may be easy to join a system as a different user who has never used a password before, thus attacking authentication.
  - If authorization information is stored in a SQL database, a successful attack of a SQL Injection vulnerability may be able to modify it.
  - A SQL Injection attack can change or remove sensitive information just as quickly as it can read it, thus hurting data integrity.

Secret passwords are used in many authentication mechanisms. Unfortunately, the length and randomness of passwords chosen by users do not change over time. On the other hand, hardware advancements provide attackers with ever-increasing computational capability. As a result, passwords are becoming obsolete. Nowadays, for securing the system against password security mechanisms, either powerful hash functions, e.g., SHA-512, is used, or such hash algorithms are adaptable to evolving trends of hardware improvements e.g., bycrypt. I have also attempted to proposed a one way strong hash function which is adaptable in nature by looking ahead in future.

2.2 Related works associated to threat

The researchers of computer, network security, and cryptographic domains have proposed various ways to tackle the issue of password protections. Mark Julius P et al [16]. present an alternative technique of encrypting data using a transposition and substitution cipher with salting, a prevalent hashing method. The goal of this cipher is to retain all of the original letters while mixing them and maintaining the overall amount of characters. The new algorithm has been shown to eliminate

the repetitious appearance of letters compared to the existing ones through a series of simulations. Furthermore, this strategy will be safer when combined with another algorithm to form a hybrid approach. Salting was used as an added layer of protection in this investigation, where the text was put at the end of the data to generate separate hashed data.

Rathod et al. [13] demonstrates how passwords affect the random creation of salt and how salt affects hash values. The study also includes an investigation of the behavior of passwords and hash values using various tools such as Wireshark, Ettercap, and Hydra. The researchers also addressed how random salt production could improve password security. They discussed various scenarios that demonstrate the difficulty an adversary must face cracking a secure password and the different random salt-generation mechanisms that can be used in conjunction with constraints to increase the computational complexity of password cracking.

Justin Holmgren et al. [5] developed CRHFs from exponentially secure one-way functions that were mildly strengthened. The distinct flavors of one-way product functions have been introduced as a new family of computational assumptions in this study (OWPFs). One advantage of this method is that it may result in non-black-box constructs that are just non-black-box in the security proof, allowing for practical efficiency. Online services generate and store a tremendous amount of data. This guide is for System end-users, System architects, System developers, and Software Testers. This project aims to employ the Salted Password Hashing Technique to secure user data.

The user information is maintained as plain text in their database, shopping online can be highly insecure. Hashing is used to overcome this issue. Instead of saving user data as plain text, P. Sriramya et al. [15] focus on saving encrypted user data to the database. The Bcrypt algorithm is used to increase the security of user data. The Bcrypt algorithm may encrypt data up to 512 bits, resulting in a longer encryption key and a hashed value for user data.

Blind decoding systems are offered as instruments for protecting consumers' privacy in the online buying of electronic documents. The company has no way of knowing which papers the customers have purchased, according to Yu-Chi Chen et al. [17] The oracle problem affects the majority of blind decoding systems. Schemes based on the transformability of digital signatures were developed to assure the validity of consumer requests. Pritesh N et al. [12] describes how to design systems that maintain password security up to date with hardware speeds.

According to Thulasimani Lakshmanan et al., [10] Hash functions are the most widely used cryptographic primitives and are currently employed in a variety of cryptographic systems and security protocols. The output length of SHA-192 is designed to be 192 by default. The SHA-192 was created to meet various increased security levels and to withstand advanced SHA attacks. When the SHA-192 security study is compared to the old one provided by NIST, it shows that the SHA-192 offers higher security. The SHA-192 algorithm can be used in a variety of applications, including public-key cryptography, digital signature encryption, message authentication code, random generator, and security architecture for forthcoming wireless devices such as software-defined radio.

Message authentication, as defined by Janaka Deepakumara at al. [7], is a critical approach for ensuring that received messages came from the alleged source and have not been tampered with. The input message can be any size, and it is processed in 512-bit blocks using 64 steps requiring 128-bit block manipulation.Alexander D. et al. [9] propose five novel contributions to ad-hoc communications and security that do the following: one-way cryptographic hashing as a mechanism for securely communicating in an environment where preexisting shared knowledge exists; hashed shared knowledge messages as a basis for the secure formation of self-selecting subgroups and trust-building; adding salt to shared knowledge hashes to remove the static nature of common messages and defend a system.

Volodymyr Shevchenko et al. [14] the authors presented a new cellular automata system that may be utilized to generate Diffe-Helmen ALgorithm one-way hash functions. For the updated Diffie-Hellman algorithm, an automated design system for the structure of one-way functions was devised. A one-way function can be employed in the modified Diffie-Hellman algorithm employing cellular automata with parameters set using a CAD system. We created a method for statistically evaluating cellular automata, which allows us to select the best automata to utilize as a one-way function. The work has designed and tested cellular automata rules in the direction of changing the basic rule of cell state change, as well as defining the cell proximity.

Praveen Gauravaram et al. [11] had; presented the first security analysis of the salt password hashing application. They showed that salt prefixed with password hashes can be precomputed offline birthday attacks when hash algorithms based on compression functions with easily discovered fixed points are employed to compute them. This attack, for example, is applicable to salt prefixed password

hashes produced using typical hash functions based on the popular Davies-Meyer compression function, such as MD5, SHA-1, SHA-256, and SHA-512. This attack reveals a subtle aspect of this application: salts prefixed to passwords do not prevent an attacker from using a precomputed birthday attack to fabricate an unknown password, despite the fact that salts prevent an attacker from obtaining passwords. Zubair Zaland et al. [18] develop a method that provides security through obscurity. The probability of a break-in is reduced exponentially by applying numerous layers of extra security protocols before storing the data. Authors attempt to protect password storage by salting, encrypting, and lastly, hashing the passwords such that no patterns are revealed.

Jeong et al. [8] proposed access-based salts that are constantly changing, thus, making passwords more secure while also concealing the salt's identity. When users log in, the IP address and access date are updated in the access log. Without breaking personal information security policy, using access log as Salt alters the hash value of the password frequently, not as flawlessly randomly as One-Time Pad.

CHAPTER 3

Proposed Approach

Any algorithm that accepts a user-supplied password as input should be protected against password guessing. Any public or stable output in form of hashed should be of limited help in reassembling the password. Although hash values are not easily broken, they are incredibly vulnerable to rainbow table and dictionary table assaults. A random text string is appended to the user's provided password, and a hash is generated to address this issue. It should be emphasized that, contrary to popular belief, using a good salt with a hash does not significantly increase the effort for an attacker who is targeting a single password or who has access to a vast number of computational resources, such as cloud-based services or specialized, low-cost hardware. Offline password cracking can still be effective if the hash function is not computationally expensive; many cryptographic algorithms are designed to be cheap and vulnerable to assaults requiring ample computing resources, even if the hash is cryptographically secure. Compared to alternative techniques such as adaptive hash functions, adding salt only minimally increases an attacker's computing requirements.

Thus, I have proposed a hashing mechanism Figure 3.1 some constraints rules to safeguard the stored passwords from offline and online attacks by considering these limitations. I have proposed a flexible block cipher 3.2 , making it harder and more time-consuming to yield the hash. It generates the key from the information from the password and salt entered. There are 18 subkeys denoted by an array K, stored in a list named the state. The proposed cipher is a 64-bit block cipher, structured as a Feistel network consisting of 16 rounds. This encrypts by splitting the entered 64 bits into left and right mini-blocks of 32-bits each.Left mini bock is xorred with subkey 1 i.e K_1.The result is passed into the confusion diffusion block (C-D Block, in short). The result of the C-D Block is xorred with subkey-2, i.e., K_2 and finally, left and right mini blocks are swapped. This ends round one. In the same fashion, the next 14 rounds are performed. Finally, in the last round, we xorred the left and right mini-block of 32 bits with subkeys 17 and 18, i.e., K_{17}, K_{18} respectively. The result of xor operations is xorred to yield a 64-bit value.

```
Function HashingPhase(round_pumped, salt, password)
    state  <-- keyGenerationAndSetupPhase(round_pumped,salt,password)
    # AIM IS TO MAKE THIS STEP SLOWER
    print('KeyConfigurationSetup Done')
    cipherText <-- "Everything is Uncertain in this World"
    repeat(64)
        cipherText  <-- getHash(state,cipherText)
    print('HASH IS OBTAINED Done')
    return = ConcatenatedHashValue(round_pumped,salt,cipherText)
```

Figure 3.1: Figure detailing pseudocode for Hashing Phase

The Confusion - Diffusion Block (C-D Block) is a simple function that takes the input of 32 bits and splits the input into two sub inputs of 16 bits each, say l_{sub_input} and r_{sub_input}. It calculates the summation(ignoring the carry for the most significant bit) and xor of l_{sub_input} and r_{sub_input} and finally concatenates the result in some fashion to produce a 32-bit block output.Thus, this all comprised of hashing phase.

My proposed approach includes one more phase: the Key generation and configuration phase 3.3. I have introduced this phase as a necessity so that the process of producing hash consumes some amount of time, thus restricting attackers from cracking the passwords quickly. This Key Key generation and configuration phase starts with initializing all 18 32-bits key with some constant value. Afterward, each key is xor with a password. Finally, the randomly generated 128-bit salt's first 64 bits are passed to the proposed block cipher using the current state of subkeys. The resultant ciphertext obtained is replaced with subkeys 1 and 2, i.e., K_1, K_2 respectively, and also this is xor with the next 64 bits of salt. Subsequently, the next 64 bits of salt are passed to the block cipher. The result is used to replace subkeys 3 and 4, i.e., K_3, K_4 respectively, and also this is xor with the first half, i.e., 64 bits of salt. After that, this process is similar until all the subkeys are replaced.

I am using arc4random CSPRNG for the salt generation, which generates random salt each time called. Some other features which I have added to my approach

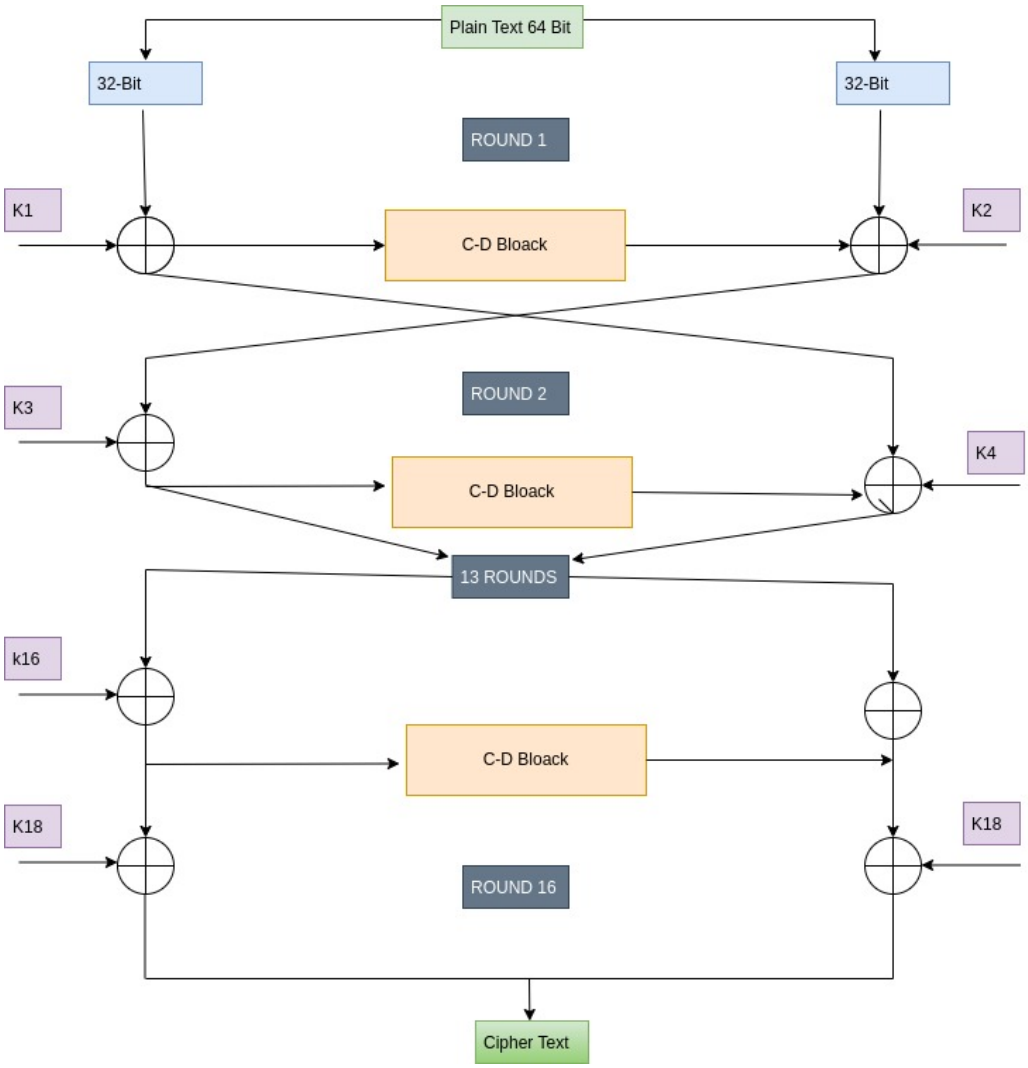

Figure 3.2: Figure Demonstrating Proposed Hashing Fiestel Network

include the following:

- The user will not be able to enter into the system if four unsuccessful authentications occur for a day. Thus, restricting the attacker from the online password guessing attack.
- The salt will automatically change after a certain period, thus, in turn, changing the corresponding hash and adding protection against offline attacks.
- The password length should be at least five characters and 20 characters.
- I am storing password's hash and salt in the database.
- Along with the username and user email, I am also storing the First time when the account is created and the current successful login time. The former

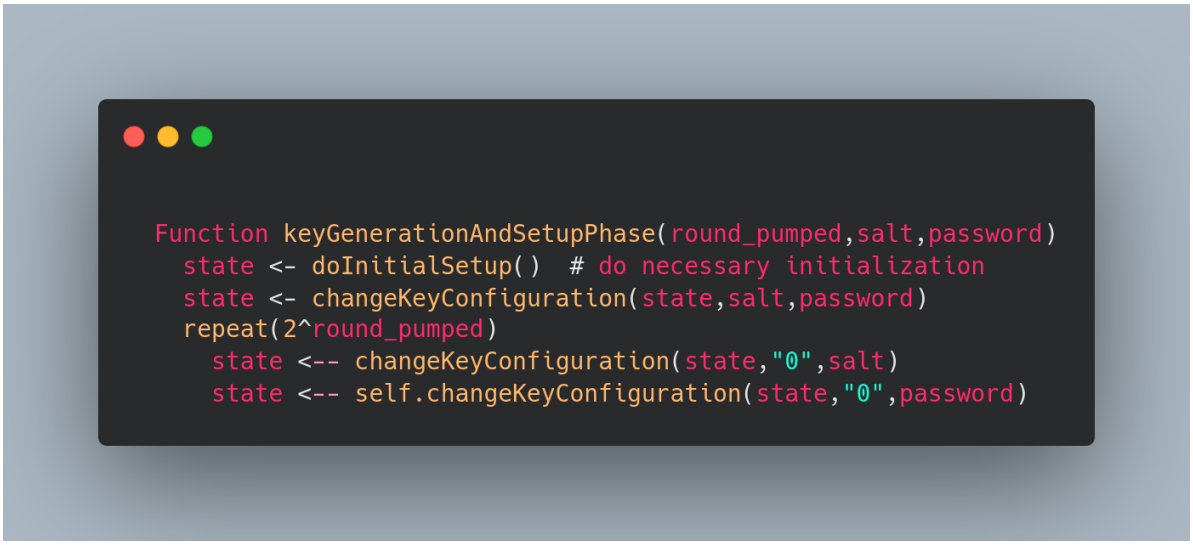

Figure 3.3: Figure Demonstrating key Generation & Setup Phase

helps in the change of salt after a certain period, while the latter helps figure out the number of unsuccessful attempts made in a day.

- For tracking unsuccessful attempts made in a day, I am storing variable columns named loginCountFailure. Once the count of this variable exceeds three on a certain day, the user will be blocked for the day and won't be allowed to enter the system.

The approached listed two functions which are required for the implementation of my proposed approach which is listed in figure below. The first one is used for key generation and configuration and second one is encryption i.e generation of hash for given salt and password.

CHAPTER 4

Result and Analysis

4.1 Libraries Used

The Flask application develpoed for simulation purpose is deployed locally. A virtual environment is created and some following libraries are installed .

- Flask
- Flask-sqlalchemy
- Flask-login
- werkzeug
- pandas
- bcrypt
- chacha20poly1305

4.2 System Specification

The setup is performed on local desktop with 8GB RAM, 1 TB Hard disk, i5 processor. The result and analysis are performed on google coolab with following specifications :

- **Google Coolab Platform** : Colab, or 'Colaboratory', it allows user to write and execute Python in the browser, with
  - 1. Zero configuration required,
  - 2. Free access to GPUs and
  - 3. Easy sharing

  It's specifications includes :

- Disk : 107.72 GB
- RAM : 12.69 GB
- CPU : Intel(R) Xeon(R) CPU @ 2.20GHz (1 core, 2 threads)
- GPU : 1xTesla K80 , compute 3.7, having 2496 CUDA cores , 12GB GDDR5 VRAM

4.3 Dataset Description

Lists of the top 100,000 and 1,000,000 passwords are available from the OWASP project. This list are also available on SecLists [6]. SecLists is the security tester's companion. It's a collection of multiple types of lists used during security assessments, collected in one place. List types include usernames, passwords, URLs, sensitive data patterns, fuzzing payloads, web shells, and many more. The goal is to enable a security tester to pull this repository onto a new testing box and have access to every type of list that may be needed. Data breaches and cyber attacks are becoming more and more common. In order to keep the online identity and private information safe, taking care of youthe passwords is as essential as ever. One of the key elements of a strong password is its uniqueness. But some passwords are anything but that. This this repository [6] contains the most commonly used passwords phrases used in passwords by people around the world.

4.4 Analysis

I have compared my algorithms against the state of art technique for the stated CWE. I have drawn a graph depicting the average time taken by the proposed algorithm for each length of password. This analysis is compared with the bcrypt algorithm. It is found that the proposed algorithms performs a lot similar to the bcrypt. Also, its additional features of change in salt after certain duration of time passed, and blocking of user account after 4 unsuccessful authentications makes the proposed algorithm better than the bcrypt. The Figure **??** shows the performance of proposed algorithm with the bcrypt.

4.5 Simulation Results

These are some of the screenshots attached for simulation results and sqlite3 results databse.

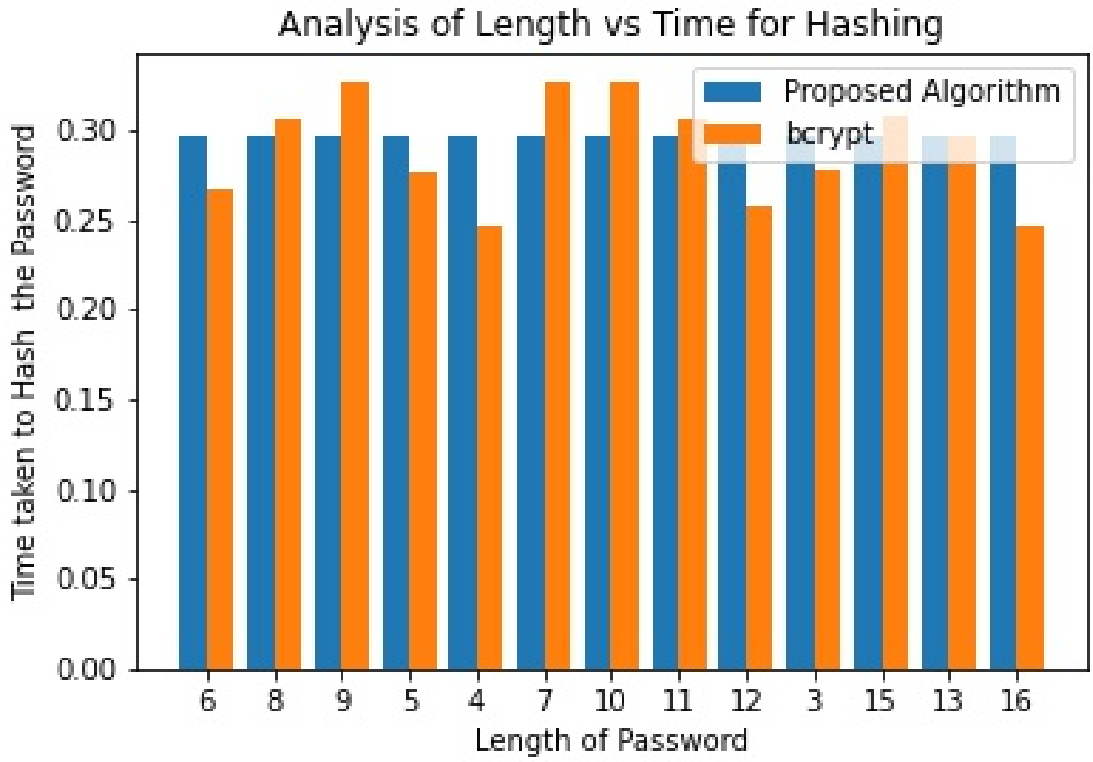

Figure 4.1: Figure showing performance analysis of proposed algorithm with the bcrypt.

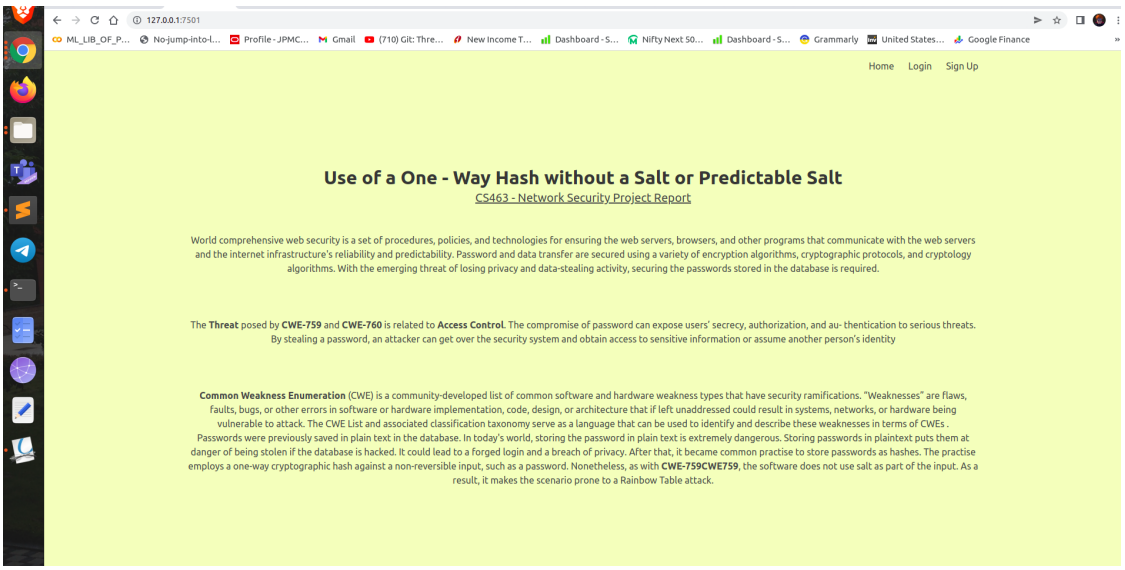

Figure 4.2: Figure showing Home Page.

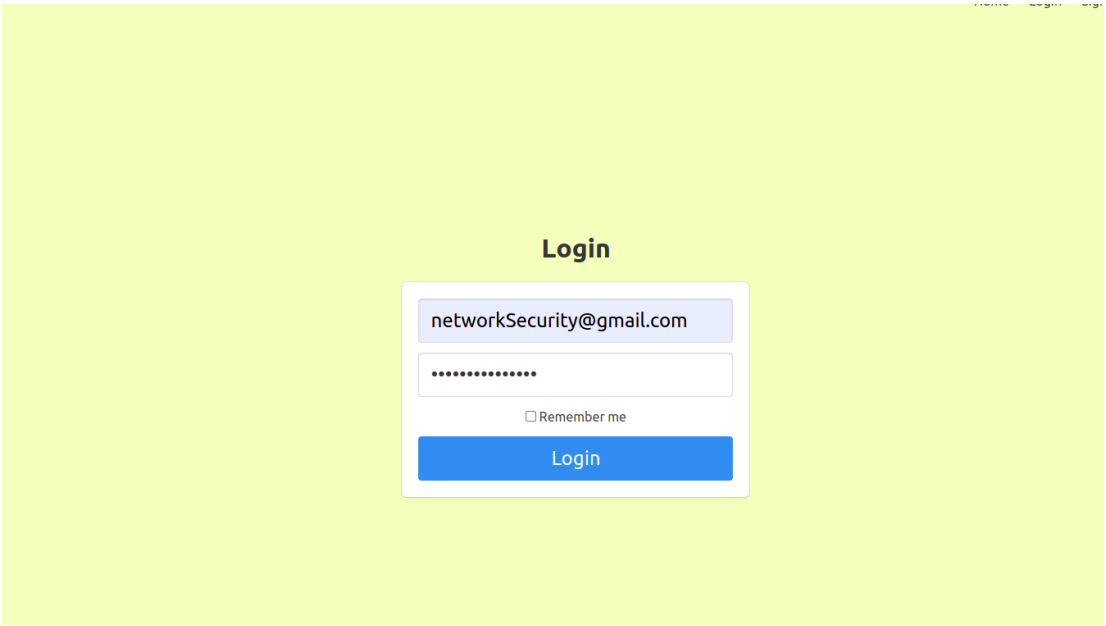

Figure 4.3: Figure showing Login Page.

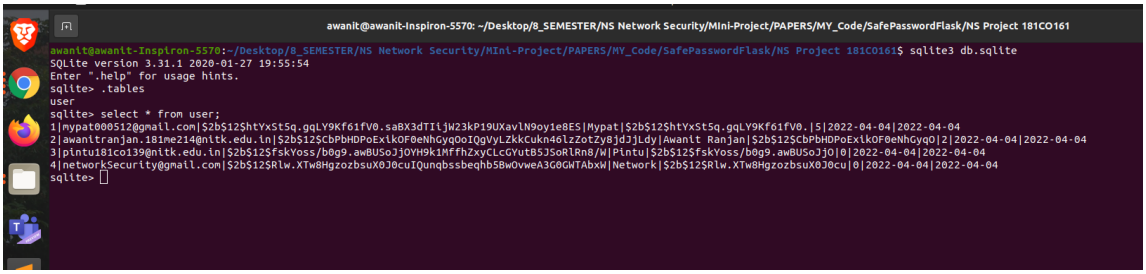

Figure 4.4: Figure showing results of user table stored in database sqlite3.

CHAPTER 5

Conclusion and Future Work

Many authentication techniques rely on passwords that are kept secret. Unfortunately, the length and entropy of passwords chosen by users remain constant over time. In contrast, hardware constantly improves, providing attackers with more computational capacity. As a result, password schemes are no longer resistant to online password guessing attacks. This project report includes I have formalized the notion of a password scheme and showed that the computational cost of such a scheme must necessarily increase with the speed of the hardware. I have proposed key Generation/Configuration Setup and Hashing Phase. The former generates the key and takes a sufficient time while the latter does the hash generation from obtained keys. I have performed the analysis with bcrypt, and my proposed algorithm performs similarly to bcrypt in taking time to produce the hash. N. As hardware speeds increase, my proposed algorithm allows preserving the cost of offline password cracking by tuning a simple configuration The algorithm is not 100 percent secure against the mentioned attacks; besides, there is a need to implement an automatic salt change as proposed. Thus, this gives a future scope of the said project.

Bibliography

[1] Common Weakness Enumerations (CWE) 759. Use of a one-way hash without a salt. *https://cwe.mitre.org/data/definitions/759.html*.

[2] Common Weakness Enumerations (CWE) 760. Use of a one-way hash with a predictable salt. *https://cwe.mitre.org/data/definitions/760.html*.

[3] Common Weakness Enumerations (CWE) 916. Use of password hash with insufficient computational effort. *https://cwe.mitre.org/data/definitions/916.html*.

[4] A community developed list of Software and Hardware weakness Types. Common weakness enumerations. *https://cwe.mitre.org/about/index.html*.

[5] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions. *IEEE*, 2018.

[6] https://github.com/danielmiessler/SecLists. Seclists.

[7] Howard M. Heys Janaka Deepakumara and R. Venkatesan. Fpga implementation of md5 hash algorithm. *IEEE*, 2020.

[8] Jinho Jeong, Dongheon Woo, and Youngwook Cha. Enhancement of website password security by using access log-based salt. *IEEE*, 2019.

[9] Alexander D Kent and Lorie M Liebrock. Secure communication via shared knowledge and a salted hash in ad-hoc environments. *IEEE*, 2011.

[10] Thulasimani Lakshmanan and Madheswaran Muthusamy. A novel secure hash algorithm for public key digital signature schemes. *Arab Journal of Information Technology, Vol. 9, No. 3, May, pp. 262-267.*, 2012.

[11] Hyderabad India Praveen Gauravaram, Tata Consultancy Services Innovation Labs. Security analysis of saltpassword hashes. *International Conference on Advanced Computer Science Applications and Technologies*, 2012.

[12] Jigisha K Pritesh N and Paresh V. Password protection using cryptographic hash.

[13] Urvesh Rathod, Meghna Sonkar, and BR Chandavarkar. An experimental evaluation on the dependency between one-way hash functions and salt. *IEEE*, 2020.

[14] Volodymyr Shevchenko, Denis Berestov, Igor Sinitsyn, Viktor Shevchenko, and Georgi Dimitrov. Automated design of the structure of one-way functions for the modified diffie-hellman algorithm. *IEEE*, 2021.

[15] P Sriramya and RA Karthika. Providing password security by salted password hashing using bcrypt algorithm. *ARPN journal of engineering and applied sciences*, 2015.

[16] Mark Julius P Tiozon, Jessie R Paragas, and Neil M Pascual. Implementation of traditional transposition cipher with salting principle 2021 ieee. *IEEE*, 2021.

[17] G. Horng Y.C. Chen and C.C. Huang. Privacy protection in on-line shopping for electronic documents. *5th International Conference on Information Assurance and Security, Vol. 2, pp. 105– 108.*, 2009.

[18] Zubair Zaland, Sibghat Ullah Bazai, Shah Marjan, and Muhammad Ashraf. Three-tier password security algorithm for online databases. *IEEE-IISEC*, 2021.