HIGH SCHOOL EDITION
Journal of Student Research
Volume 10 Issue 3 (2021)

Cryptography: A Quantitative Analysis of the Effectiveness of Various Password Storage Techniques

Rohan Patra[1] and Sandip Patra[#]

[1]Dougherty Valley High School, San Ramon, CA, USA
[#]Advisor

ABSTRACT

Recently, there has been a rise in impactful data breaches releasing billions of people's online accounts and financial data into the public domain. The result is an increased importance of effective cybersecurity measures, especially regarding the storage of user passwords. Strong password storage security means that an actor cannot use the passwords in vectors such as credential-stuffing attacks despite having access to breached data. It will also limit user exposure to threats such as unauthorized account charges or account takeovers. This research evaluates the effectiveness of different password storage techniques. The storage techniques to be tested are: BCRYPT Hashing, SHA-256 Hashing, SHA-256 with Salt, and SHA-256 with MD5 Chaining. Following the National Institute of Standards and Technology (NIST) guidelines on password strength, both a weak and robust password will be passed through the stated techniques. Reversal of each of the results will be attempted using Rainbow Tables and dictionary attacks. The study results show that pairing a strong password that has not been exposed in a data breach with the BCRYPT hashing algorithm results in the most robust password security. However, SHA-256 hashing with a salt results in a very similar level of security while maintaining better performance. While plain SHA-256 hashing or chaining multiple hashing algorithms together is theoretically as secure, in practice, they are easily susceptible to simple attacks and thus should not be used in a production environment. Requiring strong password which have not been exposed in previous data breaches was also found to greatly increase security.

Introduction

Authentication is one of the most important areas in computer security. Many forms of authentication exist including single sign-on, authentication by IP address, and hardware keys. However, due to its simplicity and universal nature, passwords remain the most common and popular form of authentication (Bonneau et al., 2012). As passwords are so widely used, focus in the security space has shifted to finding better ways to implement the use of passwords as opposed to new authentication mechanisms.

Unfortunately, the fact that passwords are so widely used is also a drawback since if a password is used in multiple places and it is leaked from one source, no security measure can prevent an attacker from using that password to log in to other places where it has been reused. According to Drexel University professor Susan Wiedenbeck, a good password is both "easy to remember and hard to guess" (Wiedenbeck et al., 2005). Therefore, simply adding a few numbers and symbols to a common word such as a pet's name is inadvisable as an attacker can easily predict it. A password which is both memorable and secure could be two random and unrelated words such as cake and bell paired with a series of special characters and numbers.

Risks in Insufficient Password Security

In the event of an attack, insufficient password security can lead to user exposure despite strong password storage techniques. According to the guidelines set forth by the NIST, it is the responsibility of the application accepting passwords to make sure that users' passwords are strong and do not contain the following criteria: the password appears in a previous data breach, the password is a common dictionary word, the password contains repetitive or sequential characters, or the password contains contextual words such as names (Grassi et al., 2017). If the password can be easily guessed, or appears in public data, it is a simple task for attackers to attempt to authenticate using every password in a list until an attempt is successful, which negates the pros of good password storage security.

Drawbacks to Encrypting Passwords

"Encryption is a way of scrambling data so that only authorized parties can understand the information. In technical terms, it is the process of converting human-readable plaintext to incomprehensible text, also known as ciphertext. In simpler terms, encryption takes readable data and alters it so that it appears random. Encryption requires the use of a cryptographic key: a set of mathematical values that both the sender and the recipient of an encrypted message agree on. Although encrypted data appears random, encryption proceeds in a logical, predictable way, allowing a party that receives the encrypted data and possesses the right key to decrypt the data, turning it back into plaintext," (Cloudflare, Inc., n.d.). Although technically secure, encryption algorithms are reversible, meaning that an attacker can find out the original text, the password in this case, without knowing the password beforehand. The only information required to reverse encryption is a decryption key or a string of characters which can be fed to an algorithm to determine the original text. When using encryption to store passwords, the decryption key must be stored somewhere on the server, and if an attacker gains access to the server and the decryption key, they can bypass the passwords' security.

Password Hashing

A hashing algorithm is a function which accepts any data and maps it to a string of a fixed size (Arias & Auth0, 2019). The output of a hashing algorithm is referred to as a hash or a digest.

What Makes a Good Hashing Algorithm?

A good hashing algorithm must match the following guidelines: it is easy and practical to compute the hash, but difficult or impossible to regenerate the original input if only the hash value is known; and it is difficult to create an initial input that would match a specific desired output ("Guide to Cryptography - OWASP," 2018). A good hashing algorithm only goes one way, meaning that once some data has been hashed, there should not be a way to reverse it. Obviously, this is a very desirable feature for password storage as a legitimate user will know both the original password and the application will know the hash, and the data can easily be compared to authenticate the user. However, if an attacker were to gain access to the application and the stored hashes, they would have no way of retrieving the original passwords.

Password Reversal Techniques

Rainbow Tables

Rainbow tables are one form of attack commonly used when attempting to reverse hashed passwords. Essentially, the attacker computes and stores the hashes to many commonly used or known passwords using various hashing algorithms. Then, the attacker simply checks if the digest in question matches any of the precomputed hashes. If a match is found, then the password is successfully found. That is why it is important to have strong password security in addition to secure password storage.

Dictionary Attacks

Another common method of reversing hashes is a dictionary attack. Like a Rainbow Table attack, the attacker must have a list of known or common passwords beforehand. However, in this attack, the hashes are not computed beforehand. Each password in the list is hashed using the same hashing technique used to produce the digest in question and compared to it. Since this technique requires computations of hashes one at a time during the attack, it is much slower than rainbow tables. However, since it does not rely on a known list of hashing techniques, it is more effective against uncommon hashing techniques.

Collision Attacks

The hallmark of a good hashing algorithm is its ability to not produce the same digest for two different pieces of data. However, since the digest must be of a fixed length but can represent data of any length, there must be multiple pieces of data which evaluate to the same digest. The longer the length of the digest, the more permutations available to represent data, which makes it more difficult for one to find such an occurrence. A collision attack, mainly only effective against weaker algorithms with shorter digests, is when an attacker does not know the original password but knows another password which results in the same hash, so the attacker can use that known password to bypass authentication.

Common Hashing Algorithms

MD5: MD5 is a hashing algorithm which generates 32-character long digests. Due to its short digest length and the fact that collisions have been found for hashes produced by this algorithm (Selinger, 2006), MD5 is a very insecure hashing algorithm.

SHA-256: "The SHA-256 algorithm is one flavor of SHA-2 (Secure Hash Algorithm 2), which was created by the National Security Agency in 2001 as a successor to SHA-1. SHA-256 is a patented cryptographic hash function that outputs a value that is [64 characters] long," (N-able, 2021). Since it has such a long character length and is cryptographically sound, no collisions have been found for SHA-256 and it is regarded as a secure hashing algorithm.

BCRYPT: BCRYPT is an algorithm which is of a subset of hashing algorithms known as key-stretching algorithms. Key-stretching algorithms serve the same purpose of hashing algorithms; however, they are much slower and harder to compute. This results in greater security since "even with a fast GPU or custom hardware, dictionary and brute-force attacks are too slow to be worthwhile," (CrackStation, 2019).

Password Salting

Password salting is a technique used to decrease the effectiveness of Rainbow Table attacks. A randomly generated, long string of characters, or salt, is prepended or appended to the data before hashing and store alongside the data. Each password uses its own unique salt. Thus, rainbow table attacks are rendered ineffective as precomputed hashes cannot account for modifications to the known passwords. However, salted passwords are still vulnerable to dictionary attacks.

Research Question

What is the most secure way to store passwords; and which method provides the best balance between security and performance for viability in a production environment?

Methodology

Overview

The storage techniques to be tested are: BCRYPT Hashing, SHA-256 Hashing, SHA-256 with Salt, and SHA-256 with MD5 Chaining. Although not proven to be secure, a commonly used superset of password hashing is hash chaining. In the case of this experiment, we are assuming that the application stores the SHA-256 hash of the MD5 hash of the original password. Following the National Institute of Standards and Technology (NIST) guidelines on password strength, both a weak and robust password will be passed through the stated techniques, and the execution time for each technique will be recorded. Then, reversal of each of the resulting strings will be attempted using rainbow tables and dictionary attacks (hashcat). The data recorded will be the time taken to reverse the hash or whether the attack was successful. A performance index will be created for each algorithm to properly compare the varying storage techniques, considering both execution time and reversal time. The performance index will be the average time taken to completely execute each technique rounded to two places of decimal.

Passwords to Test

The insecure password to be tested will be Jeff123. It breaks all the NIST's guidelines on strong passwords as it is a common name and appears in several known data breaches. See Figure 1. The strong password to be tested will be JeffPurpleCats76! which follows all the NIST's guidelines on password security. It adds on to the weak password by appending unrelated but memorable words as well as number and special characters. This results in a strong but memorable password. The password also has not been seen in any public data breaches as seen in Figure 2.

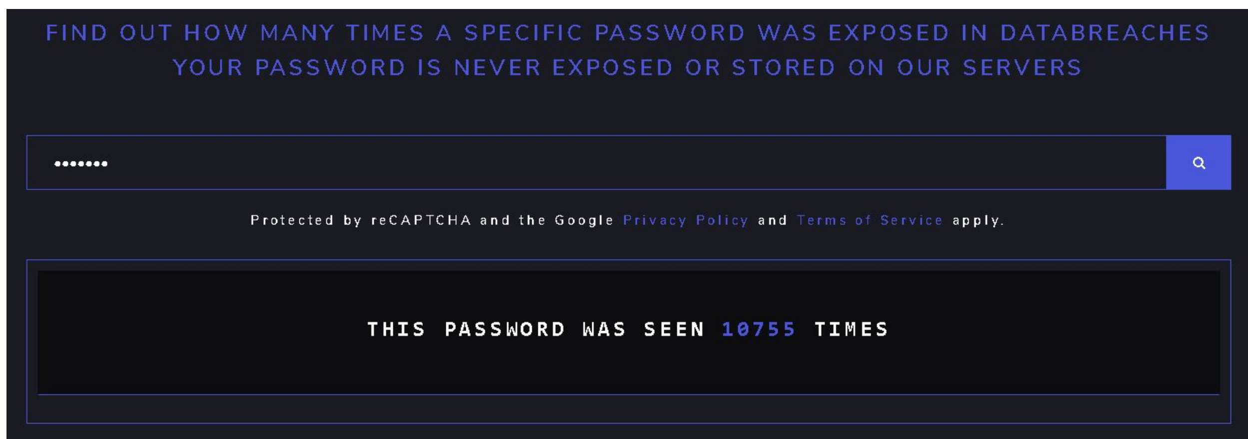

Figure 1. Checking whether the password Jeff123 has been exposed in a data breach via BreachDirectory (Patra, n.d.).

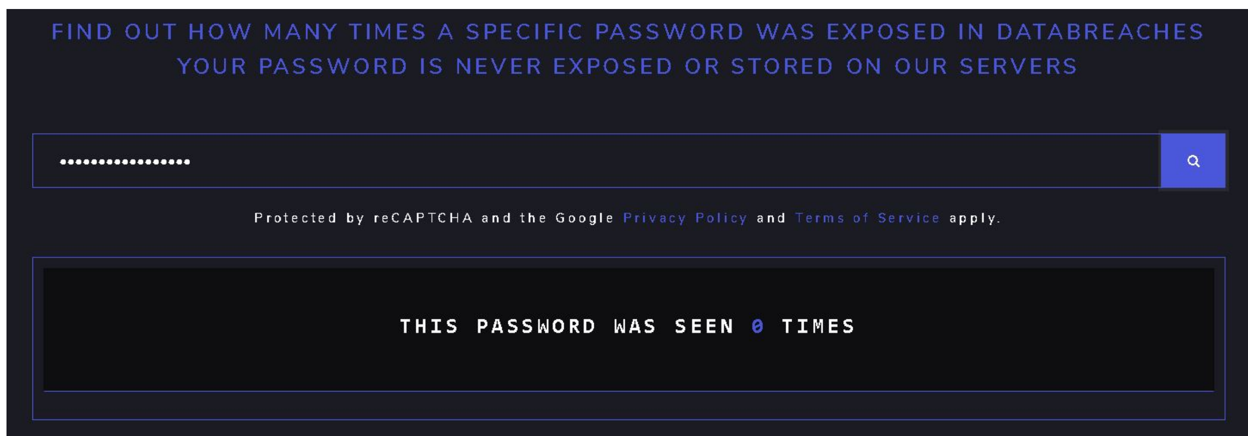

Figure 2. Checking whether the password JeffPurpleCats76! has been exposed in a data breach via BreachDirectory (Patra, n.d.).

Testing Storage Technique Computational Efficiency

Each storage technique will be implemented and tested using the programming language Python (Python Software Foundation, 2021). See Figure 3, Figure 4, Figure 5, and Figure 6 for the code used to test BCRYPT hashing, SHA-256 hashing, Salted SHA-256 hashing, and SHA-256 with MD5 hash chaining respectively. Each technique will be tested 200 times to get a more accurate average representation of computational efficiency.

```
import bcrypt
import time
def hash_it():
    salt = bcrypt.gensalt()
    print(bcrypt.hashpw(b'Jeff123', salt) + salt)
start_time = time.time()
for i in range(200):
    hash_it()
end_time = time.time()
print("Total time taken was %f milliseconds." % ((end_time - start_time) * 1000.0))
```

Figure 3. Python code to find the BCRYPT hash of a given password 200 times and give the time taken in milliseconds. The code prints the result of hashing and the salt from each round separated by dollar sign.

```
from hashlib import sha256
import time
def hash_it():
    print(sha256(b'Jeff123').hexdigest())
start_time = time.time()
for i in range(200):
    hash_it()
end_time = time.time()
print("Total time taken was %f milliseconds." % ((end_time - start_time) * 1000.0))
```

Figure 4. Python code to find the SHA-256 hash of a given password 200 times and give the time taken in milliseconds. The code prints the result of hashing from each round.

```
import secrets
import time
from hashlib import sha256
def hash_it():
    salt = secrets.token_hex(8)
    print(sha256(b'Jeff123'+bytes(salt, 'utf-8')).hexdigest()+":"+salt)
start_time = time.time()
for i in range(200):
    hash_it()
end_time = time.time()
print("Total time taken was %f milliseconds." % ((end_time - start_time) * 1000.0))
```

Figure 5. Python code to find the salted SHA-256 hash, using a randomly generated 16-character salt of a given password 200 times and give the time taken in milliseconds. The code prints the result of hashing and the salt from each round separated by a colon.

```python
import time
from hashlib import sha256, md5
def hash_it():
    print(sha256(md5(b'Jeff123').digest()).hexdigest())
start_time = time.time()
for i in range(200):
    hash_it()
end_time = time.time()
print("Total time taken was %f milliseconds." % ((end_time - start_time) * 1000.0))
```

Figure 6. Python code to find the SHA-256 hash of the MD5 hash of a given password 200 times and give the time taken in milliseconds. The code prints the result of hashing from each round.

Testing Rainbow Table Attack

The Rainbow Table attacks will be simulated using the BlueCode Hash Finder software (BlueCode Team, n.d.). See Figure 7 for a screenshot of the software. The software includes a pre-built database of various types of hashes for a comprehensive list of known or dictionary-based passwords. The software will be loaded with a list of the hash to be reversed repeated 200 times and it will be run using the default settings. The total time for execution will be divided by 200 to find the average time taken to reverse the hash.

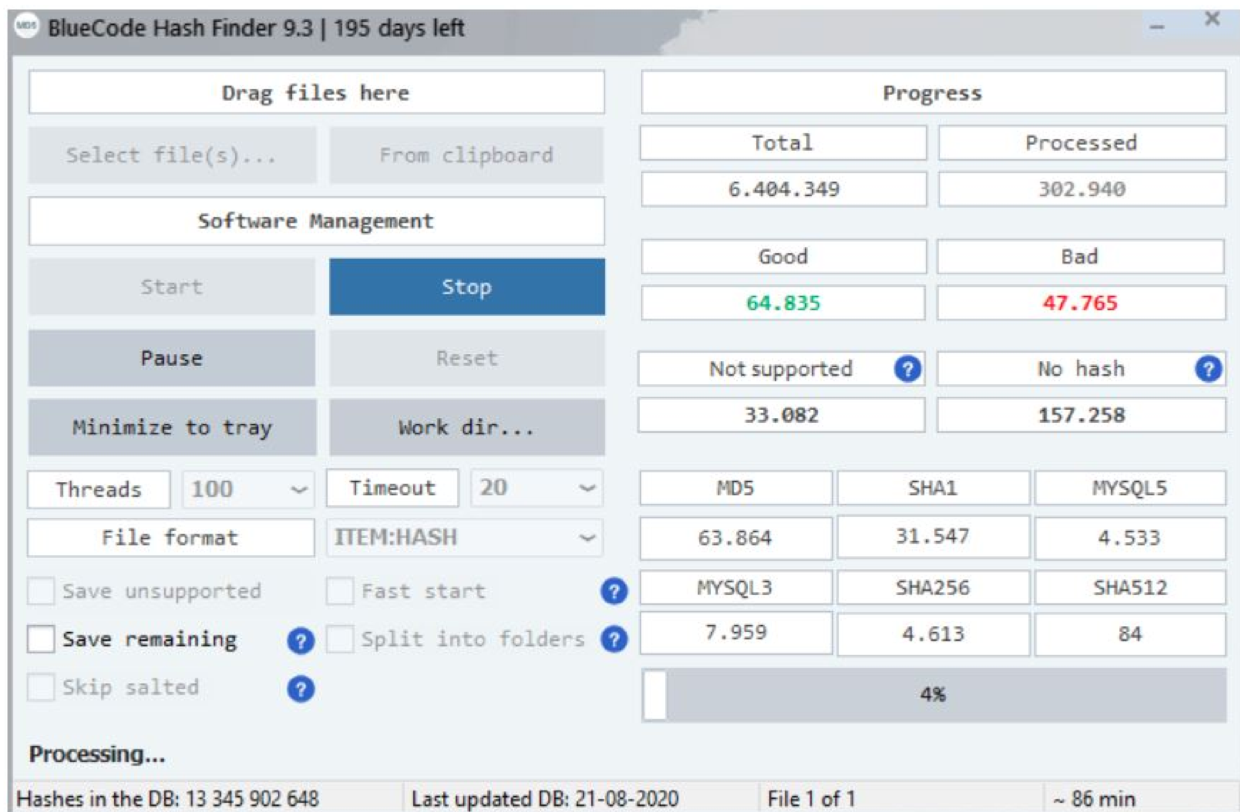

Figure 7. Screenshot of the BlueCode Hash Finder software (BlueCode Hash Finder, 2020), for testing Rainbow Table Attacks.

Testing Dictionary Attack

The dictionary attack will be tested using a Python script. Although there are various utilities which support greater speeds as a result of utilizing the computer's Graphics Processing Unit and multi-threading, those utilities cannot be used in this experiment as no single utility supports all the password storage techniques to be tested, and thus, results may become skewed. The list of known passwords to be used in the simulated dictionary attacks is the list known as "weakpass_2a" (Weakpass_2a, 2017). It is a public compilation of known passwords from various public sources.

```
import time
start_time = time.time()
for line in open("weakpass_2a.txt", "r"):
    if hash_it(line.strip()) == query_hash:
        print("Hash Found:", line)
        break
end_time = time.time()
print("Total time taken was %f milliseconds." % ((end_time - start_time) * 1000.0))
```

Figure 8. Python code to attempt to reverse the hash "query_hash" using the wordlist contained in "weakpass_2a.txt". Code assumes that function "hash_it()" provides the digest of the algorithm used to produce "query_hash" when run on each line in the wordlist.

The dictionary attack will be tested on each hashing technique and password 5 times and the average will be taken.

Hardware Specifications

To make sure that the results of this experiment can be reproduced, the hardware specifications of the device on which the test scripts will be run are shown in Figure 9.

```
Device Information
Device name     DESKTOP-3SOAEI6
Processor       Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
Installed RAM   16.0 GB
System type     64-bit operating system, x64-based processor

Windows OS Information
Edition         Windows 10 Pro
Version         20H2
Installed on    2/20/2021
OS build        19042.985
Experience      Windows Feature Experience Pack 120.2212.2020.0
```

Figure 9. Screenshot of hardware specifications of device on which test scripts will be run.

Results

After conducting a series of tests, the following results were obtained.

Storage Technique Computational Efficiency

Weak Password

```
b'$2b$12$mXUqEpoFYg4OLhlS9P7kZ.aU7iNi/WYPhZYxFAJKRTY/7727WdPbC$2b$12$mXUqEpoFYg4OLhlS9P7kZ.'
b'$2b$12$dfXuemz1MneYHG7/UnxHm.mOzvVKlBNxHhn.HdOp.bXx/1ztWcbAq$2b$12$dfXuemz1MneYHG7/UnxHm.'
b'$2b$12$yQLYHQ5JuFbRQvX9.M5CAesRHgPTS7eLCD6JxhYqPFfjB1w02Pv2G$2b$12$yQLYHQ5JuFbRQvX9.M5CAe'
b'$2b$12$IawGv1AA8NwFxMl.gYQcmOuvpz/JCUOl16Azb/jtCgQrFFXX.iy..$2b$12$IawGv1AA8NwFxMl.gYQcmO'
b'$2b$12$dIKmJWG8udWgVfDfMpkmtetdqncsnihguQ2qq/xxdX1bJLGj1g4S.$2b$12$dIKmJWG8udWgVfDfMpkmte'
b'$2b$12$FTtuh6uj27nHOhuRYE/K.eK5PIbM4xqe0kpHMbt2s5lryzOBc4jYy$2b$12$FTtuh6uj27nHOhuRYE/K.e'
b'$2b$12$RZ0CZeIVk2g3ODfTFVBh8eNb1h/4liy5oZWR2x.skVRzduBgEjq7C$2b$12$RZ0CZeIVk2g3ODfTFVBh8e'
b'$2b$12$9J2aNAAIuIwNi9Le/tzeD.UJCog/lLqRnfQd3jh82xPzYfzj7fMiW$2b$12$9J2aNAAIuIwNi9Le/tzeD.'
b'$2b$12$C8TRNvonbXX1loXBtzh8QOICPh9Eur24UZ/6mgoezQFzYyBwg9dom$2b$12$C8TRNvonbXX1loXBtzh8QO'
b'$2b$12$1tnqPknlQyIqQcue7Pa4R.34hjkM1CW4CjnBFdQ4COIk1GPKzJS7y$2b$12$1tnqPknlQyIqQcue7Pa4R.'
Total time taken was 42114.629030 milliseconds.
```

Figure 10. Output of the python script in Figure 3 when run against the password Jeff123.

```
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
'07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680'
Total time taken was 11.000156 milliseconds.
```

Figure 11. Output of the python script in Figure 4 when run against the password Jeff123.

```
'e2a820efe3b8d58e6f30bc0f2a798a5cfdf9d01750629dd58e0805bcdf41df1c:37d884c4cdc3faa5'
'7d6abf185cc95e6d9d3d0c059bb3b69301ae42f49928f9bf8d9a10e170f2193f:4acf616c998ca61e'
'2a5b094095c617152c3d0c8daacf1b3d47393ac3cd7692b5fe0d1e2877d4a461:506de8431f7c2199'
'71beb58cb28141a52969b97938beee0cd339dda193742f732081fe7ed9c89d57:d829b3526113a591'
'c9f5fcab4f67e0bbe52b879d7ccd94b3fd6443abad39caad14c3caf829f0ab0d:32d42970c6d090bb'
'0b5bfc2947d83fa6dbaac7cc5133d013562a4c221a942c792fb884060c83e637:5c4651ceec734870'
'7d58a3a3b33a403915a80242f0e7e374012e08ad9848c7b53652a3424eded671:96da0fe4a1babd4b'
'7843e6bf4f6c37ecc7574d0ea5ce59315b876ff56f1ae0e448375965b2b27326:74d7ef0b16be6ca2'
'0376ba20656c0b2c5359815ddd2f84d9a2250f21bbe706a3c38af587db5e4e99:ee9699c7504493fd'
'23153f3a168b2681ff7877b1499716d2e94abf420ad185f7c7d72d75cbfee058:253e1f7d3eec7bca'
Total time taken was 14.998913 milliseconds.
```

Figure 12. Output of the python script in Figure 5 when run against the password Jeff123.

```
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
'39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf'
Total time taken was 18.360376 milliseconds.
```

Figure 13. Output of the python script in Figure 6 when run against the password Jeff123.

Table 1. Table detailing the computational efficiency and outcomes of storing the weak password Jeff123 using five different techniques. The data includes the total time to store it 200 times using each technique, the average time to store the password once using each technique, and each technique's performance index. The lower the performance index, the more efficient the technique is.

Technique	Total Time (ms)	Avg. Time (ms)	Index
BCRYPT	42114.63	210.57315	210.57
SHA-256	11.00	0.055	0.06
SALTED	14.10	0.0705	0.07
CHAINING	18.63	0.0918	0.09

As an additional note, in both BCRYPT hashing and Salted SHA-256 hashing, each resulting digest is different even though the original data is the same. This results in additional security in cases where the different users are using the same password as if one password is exposed, the other one is still secure.

Strong Password

```
b'$2b$12$mXaOUwZ.oy9WOaGSo.MjReFkM/Nr6NbZuRFS7xxTxMpOG7tdzaz82$2b$12$mXaOUwZ.oy9WOaGSo.MjRe'
b'$2b$12$SXexwNeTSbd9r45PmoMlRuKdY3Zc2o6W51IoqnZzOMhSlLw0dqifa$2b$12$SXexwNeTSbd9r45PmoMlRu'
b'$2b$12$QmT/OhnICdOIxL9xrdauDu8Kddk8mFvi/tXbDjpVmhYFpkQBS2t2q$2b$12$QmT/OhnICdOIxL9xrdauDu'
b'$2b$12$3WK7fkNTlJvF20Ox2R3jbuloBnRr3Xw9sXhvEn.hY9otKqd3Lesxa$2b$12$3WK7fkNTlJvF20Ox2R3jbu'
b'$2b$12$vd9rI5rs7PgWmjS0r9L6Pe0i.YeESctSm5CLldyZc6JOAdhBmRRZC$2b$12$vd9rI5rs7PgWmjS0r9L6Pe'
b'$2b$12$xCkjlDqIJp8omGjRAqJXieGO67NLFyBvBlyWDHFyfCwHUfTzbpaC6$2b$12$xCkjlDqIJp8omGjRAqJXie'
b'$2b$12$PoJcpVEbbh/2sGA.hsMsRuA1GBZGlO.IHtQ30EU/xPmXBF3hIPTmK$2b$12$PoJcpVEbbh/2sGA.hsMsRu'
b'$2b$12$8csujdB6Ox8so9sd5jQNP.o0FCbSZJOXH.LMcCzEA3gu4AxSEz0yW$2b$12$8csujdB6Ox8so9sd5jQNP.'
b'$2b$12$aIzXfcXhtUQ.j0rAAqjqT.dws8wNRMynfXqy2aza6PDMm9Ifa9hm6$2b$12$aIzXfcXhtUQ.j0rAAqjqT.'
b'$2b$12$fSvViflKcCyNWRiKt8mv.eHh3q4q/ymxDwGAJlcCCXCY97pzLEtdC$2b$12$fSvViflKcCyNWRiKt8mv.e'
Total time taken was 42286.035538 milliseconds.
```

Figure 15. Output of the python script in Figure 3 when run against the password JeffPurpleCats76!.

```
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
'3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450'
Total time taken was 8.007050 milliseconds.
```

Figure 16. Output of the python script in Figure 4 when run against the password JeffPurpleCats76!.

```
'ce5514cbc5e673928a3d33641bee2bd379d5925e726a631b171b0fe642566a55:d1e6b1711405bbd6'
'e7d3bf78927a16877becc0686c702882a8150227054db4a9cae20bce021d75d2:aee40df0e80f88c7'
'4c22812635a8403b0f7218138e4bac5ad677b96af836526c95fea36599a9e375:4877ca09870bdf1c'
'a16a33a7814686ed42a6d57e0e04d6d944e349b403ad883bab8a66e8495afbc1:130525d90251a308'
'aad70763ff38258687be0e04207ebbf910462110908fcd96eb28a7b90596515b:4e51319564b6b6cb'
'bab7a65a3152420c20b6b7826df719e05401ce9dfc339f38b53643bf88fc9898:9113cecfbf00a59e'
'1426fe71a130683b8b489d2a72e2fd0b80e290b6a42d5321127a2a9c53e28676:5b7927066d7e0406'
'0922ad08a68c91bce18c99c403eff947fa3b14ade01906f864c2920f53fd3e4b:be550892e20c4e02'
'16dead0846ced5d971318e71f1bd1b68f74a4258d8db1611648d06d8526ede48:a81199491db239d4'
'6c71de913ad7e3d12d82cc4e508562d028e82860f0c7b010753cfec07c6b378d:70d676a71fd05361'
Total time taken was 15.014172 milliseconds.
```

Figure 17. Output of the python script in Figure 5 when run against the password JeffPurpleCats76!.

```
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
'e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884'
Total time taken was 10.015488 milliseconds.
```

Figure 18. Output of the python script in Figure 6 when run against the password JeffPurpleCats76!.

Table 2. Table detailing outcomes of storing the weak password JeffPurpleCats76! using five different techniques. The data includes the total time to store it 200 times using each technique, the average time to store the password once using each technique, and each technique's performance index. The lower the performance index, the more efficient the technique is.

Technique	Total Time (ms)	Avg. Time (ms)	Index
BCRYPT	42286.04	211.4302	211.43
SHA-256	8.01	0.04005	0.04
SALTED	15.01	0.07505	0.08
CHAINING	10.02	0.0501	0.05

Hash Reversal Techniques

Table 3. Table detailing the specific hash digests to be reversed in this section.

Technique	Original	Hash
BCRYPT	Jeff123	$2b$12$fBoKgNe.MMDA4WztLtttAO0ZyelAdJVTYwr3qtWfy5/kN9r9teEzO$2b$12$fBoKgNe.MMDA4WztLtttAO
SHA-256	Jeff123	07008d4ceca9fcf36c8022bf74dba9b81795a5fb5c5874d8080ebc426a095680
SALTED	Jeff123	fc32ee5228350a5e484e40884475856cec11e2cc72a6830f9965d8c58c997bb5:236aad22d550ad96
CHAINING	Jeff123	39bbcd5c11fee656d65479015561a8cb10e13c03430c9624def3d27b88839dcf
BCRYPT	JeffPurpleCats76!	$2b$12$CzPe/YXh77z0FQ5BdFBrMOeiPnkU68eMUeSCUg78Aurudjm2Z9zaa$2b$12$CzPe/YXh77z0FQ5BdFBrMO
SHA-256	JeffPurpleCats76!	3cfe7a9c4ba17b658a40b26f46f2ef2d8d2dd6d6390ddb7c3a7b1cab21971450

SALTED	JeffPurpleCats76!	73c3983d24b01d898c896765df64a752ca7ea86cbe449a9000a060a0490f9d00:7c188d7bb93991a7
CHAINING	JeffPurpleCats76!	e8cd09d400ae6c90c7311e12e48908d4b9013c770e12b5613541c3b5b04be884

Table 4. Table detailing outcomes of attempting to reverse the stored weak password Jeff123 using Rainbow Tables. The data includes the total time to reverse it 200 times from each technique, the average time to reverse each technique once, and whether the reversal was successful.

Technique	Total Time (ms)	Avg. Time (ms)	Successful
BCRYPT	N/A	N/A	FALSE
SHA-256	0.17	0.00085	TRUE
SALTED	N/A	N/A	FALSE
CHAINING	0.12	0.0006	TRUE

Table 5. Table detailing outcomes of attempting to reverse the stored strong password JeffPurpleCats76! using Rainbow Tables. The data includes the total time to reverse it 200 times from each technique, the average time to reverse each technique once, and whether the reversal was successful.

Technique	Total Time (ms)	Avg. Time (ms)	Successful
BCRYPT	N/A	N/A	FALSE
SHA-256	0.15	0.00075	TRUE
SALTED	N/A	N/A	FALSE
CHAINING	0.14	0.0007	TRUE

Table 6. Table detailing outcomes of attempting to reverse the stored weak password Jeff123 using dictionary attacks. The data includes the total time to reverse it 5 times from each technique, the average time to reverse each technique once, and whether the reversal was successful.

Technique	Total Time (hours)	Avg. Time (hours)	Successful
BCRYPT	65.52	13.104	TRUE
SHA-256	0.02	0.004	TRUE
SALTED	0.05	0.01	TRUE
CHAINING	0.05	0.01	TRUE

Table 7. Table detailing outcomes of attempting to reverse the stored strong password JeffPurpleCats76! using dictionary attacks. The data includes the total time to reverse it 5 times from each technique, the average time to reverse each technique once, and whether the reversal was successful.

Technique	Total Time (hours)	Avg. Time (hours)	Successful
BCRYPT	584.88	116.976	TRUE
SHA-256	0.15	0.03	TRUE
SALTED	0.17	0.03	TRUE
CHAINING	0.16	0.03	TRUE

Overall Analysis

To better compare all obtained data, the data is plotted on a graph as coordinate points with the x-coordinate being the storage technique's average performance index and the y-coordinate being the number of reversal techniques which were able to successfully reverse it. See Figure 19. The closer the point is to the origin, the better the efficiency and reversal difficulty of the storage technique.

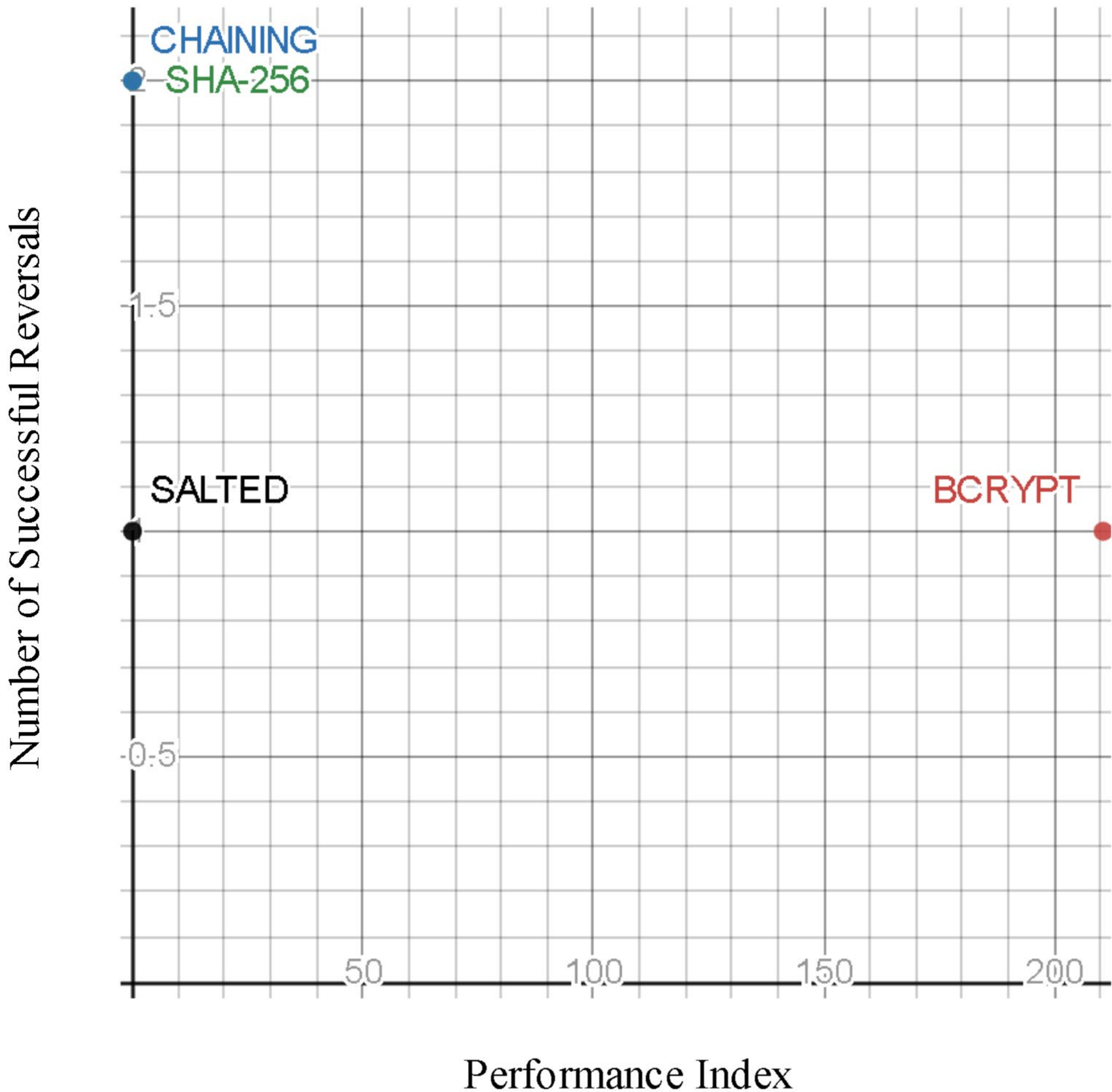

Figure 19. Overall data graphed as coordinate points with the x-coordinate being the storage technique's average performance index and the y-coordinate being the number of reversal techniques which were able to successfully reverse it. Both chaining and SHA-256 have near identical values.

Discussion

The data as a whole showed strong security when using the Salted SHA-256 and BCRYPT password storage techniques. Since plain SHA-256 and hash chaining are commonly used techniques for storing passwords, and they fail to generate unique digests given the same password, they commonly appear in both Rainbow Tables and dictionaries. Although they are extremely efficient compared to alternatives, the efficiency is increased at the cost of security and reversal difficulty. Based on the graph in Figure 19, both BCRYPT and Salted SHA-256 appear to have similar levels of security as they are both resistant to Rainbow Table attacks. However, when looking at the timings in Table 6 and Table 7, BRYPT takes a vastly larger amount of time to reverse. This is due to the fact that it is a key-stretching algorithm. However, BCRYPT's security and reversal difficulty come with tradeoffs as well since it is extremely inefficient even when compared with Salted SHA-256. Despite being much slower to reverse than CHAINING and SHA-256, Salted SHA-256 has a very similar level of efficiency. Thus, it is the ideal choice for those who require the best combination of security and efficiency. Between the four techniques examined in this experiment, the ultimate decision when it comes to choosing a password storage technique is essentially a choice between Salted SHA-256 and BCRYPT. If efficiency is not an issue in an application, BCRYPT's security would make it the ideal choice.

In addition to a proper password storage technique, requiring strong passwords is highly beneficial to overall security. Especially in the instance of BCRYPT, using a strong password greatly increases the difficulty of reversal.

Conclusion

The study results show that pairing a strong password that has not been exposed in a data breach with the BCRYPT hashing algorithm results in the most robust password security. However, SHA-256 hashing with a salt results in a very similar level of security while maintaining solid performance. While plain SHA-256 hashing or chaining multiple hashing algorithms together is theoretically as secure, in practice, they are easily susceptible to simple attacks and thus should not be used in a production environment. Although choosing a good password storage technique can play a big role in user security, much of the responsibility falls on the shoulders of the users themselves to use strong and unique passwords that have not been exposed in past data breaches. In the end, the goal of implementing proper password storage is to use a technique secure enough, and difficult and time-consuming enough to reverse, to deter possible attackers.

Acknowledgments

I would not have been able to complete this project without the aid of many people. First, I would like to thank my father and mentor, Mr. Sandip Patra, who lent me his vast technical knowledge and experience. Secondly, I would like to thank the rest of my family and my friends for their love and support. Finally, I would like to thank the Journal of Student Research, which made this entire project possible by making the world of research available to high school students.

References

Arias, D. & Auth0. (2019, September 30). Hashing Passwords: One-Way Road to Security. Auth0 - Blog. https://auth0.com/blog/hashing-passwords-one-way-road-to-security/

BlueCode Hash Finder (9.3). (2020). [Computer software]. BlueCode Team. https://bluecode.info/

Bonneau, J., Herley, C., Oorschot, P. C. V., & Stajano, F. (2012). The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. 2012 IEEE Symposium on Security and Privacy. Published. https://doi.org/10.1109/sp.2012.44

Cloudflare, Inc. (n.d.). What is encryption? Cloudflare. Retrieved May 15, 2021, from https://www.cloudflare.com/learning/ssl/what-is-encryption/

CrackStation. (2019, June 5). Secure Salted Password Hashing - How to do it Properly. https://crackstation.net/hashing-security.htm

Grassi, P. A., Fenton, J. L., Newton, E. M., Perlner, R. A., Regenscheid, A. R., Burr, W. E., Richer, J. P., Lefkovitz, N. B., Danker, J. M., Choong, Y. Y., Greene, K. K., & Theofanos, M. F. (2017). Digital identity guidelines: authentication and lifecycle management. Digital Identity Guidelines. Published. https://doi.org/10.6028/nist.sp.800-63b

Guide to Cryptography - OWASP. (2018, June 13). In Open Web Application Security Project. https://wiki.owasp.org/index.php/Guide_to_Cryptography

N-able. (2021, April 1). SHA-256 Algorithm Overview. https://www.n-able.com/blog/sha-256-encryption

Patra, R. (n.d.). BreachDirectory - Check If Your Email or Username was Compromised. BreachDirectory - PASSCHECK. Retrieved May 29, 2021, from https://breachdirectory.tk/passwords

Python Software Foundation. (2021, May 24). Welcome to Python.org. Python.Org. https://www.python.org/
Selinger, P. (2006, February). MD5 Collision Demo. Dalhousie University. https://www.mscs.dal.ca/~selinger/md5collision/

weakpass_2a. (2017). Weakpass. https://weakpass.com/wordlist/1919

Wiedenbeck, S., Waters, J., Birget, J. C., Brodskiy, A., & Memon, N. (2005). Authentication using graphical passwords. Proceedings of the 2005 Symposium on Usable Privacy and Security - SOUPS ’05. Published. https://doi.org/10.1145/1073001.1073002