The State of Password Storage in the Wild

Teodor Dobre
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, U.K.
psytd8@nottingham.ac.uk

Xavier Carpent
School of Computer Science
University of Nottingham
Nottingham, NG8 1BB, U.K.
pszxc1@exmail.nottingham.ac.uk

ABSTRACT
Passwords are the most widely used form of authentication on the internet. The main way of storing these passwords is by using hashing algorithms, which make passwords unreadable to humans in plain text. However, there are ways to find out what the password behind a hash is and depending on the strength of the hashing algorithm used, different levels of difficulty in doing so. In this work, we present the state of server-side password storage in the wild in recent years based on discovered data breaches and analyze the passwords in some of these breaches. The results of our analysis will show what the usage of different hashing algorithms is throughout recent years and overall, and the distribution of the strength in passwords in different breaches, correlating it with the type of service where they were hosted and the quality of password storage used, while also emphasizing the difficulties in doing such a study reliably.

CCS Concepts
• Security and privacy→Security services→Password storage.

Keywords
password storage; passwords; leaks; data breaches; hashing

1. INTRODUCTION

1.1 Background Information

1.1.1 Password Storage
Passwords are used to authenticate users before granting them access to the assets of a website. They are normally created at the registration stage when a user wants to create an account for a service. After deciding on a password and inputting it on a website, a hashing function is run on it, which takes in the password and creates a string indistinguishable from random noise from it. The hash is then stored inside a database on the website's server. This way, not even the admins of the website will be able to tell what a user's password is. These databases are, however, susceptible to hacking, along with other data stored besides the passwords, by hackers that exploit vulnerabilities on the website, such as SQL injection attacks [1], LFI/RFI [2], public-facing databases, etc. Since hashes of passwords cannot be reversed to reveal the passwords behind them, hackers only have the option to run the same hashing function that was used when storing the passwords on strings that they think may be passwords inside the database, using different attack patterns which we discuss in the next subsection. This is known as offline cracking [3].

The worst storage method is storing the passwords in plain text, as once the database is leaked, the hacker can see the exact passwords users have for their accounts.

The optimal way to store passwords is password hashing with a password based key derivation hashing function. The main idea is that hash functions take a message of any length and return a pseudorandom hash of fixed length. Because of this, the passwords are unintelligible to any bad actor that is responsible for a security breach. Hash functions also have the attribute of being one-way functions, meaning there is no way to reverse the hash back to the original password. So, when a user types in their password to log in, the password they input is hashed and the resulting hash is compared to the one in the database. If the hashes match, the user is allowed to log in. However, not all hash functions have the same security level.

There are hash functions that are less secure or are not appropriate for use in password storage, such as MD5, SHA-1, SHA-256, etc. These hash functions are either outdated (MD5 was created in 1991 and is proven to be broken [4]) and are no longer suitable for use in modern systems, or are useful in other applications and not password storage, at least by themselves (SHA-256 is used to verify Bitcoin transactions [5] and SHA-256/512 are used as the hashing functions for the PBKDF2 algorithm where they are used for a large number of iterations). It is very easy for modern hardware to go through a large number of hashes of these functions in a small amount of time in order to crack them.

The next step up from this would be salting the passwords before hashing them [6]. This is the process of adding a random string of characters to the user's password before hashing and storing it. The salt that was added is also stored alongside the hash, and whenever the user wants to log in, the salt is added to the input, hashed, and compared against the database. This method makes dictionary and rainbow table attacks ineffective, leaving brute force as the only option. It also helps in protecting users who choose weak passwords.

There is also peppering, which works in a similar way to salting, however the pepper is not stored with the hash, *"but rather the pepper is kept separate in some other medium, such as a Hardware Security Module"* [7].

The current academic standard in password storage are password based key derivation functions, such as bcrypt, scrypt, PBKDF2 and argon2. These functions have one of the conventional hashing functions as a component. However, the difference is that they run that function in a loop for a high number of iterations on the password before storing it. For example, for PBKDF2, in the year 2005 the recommended number of iterations was 4096 [8]. However, as hardware speeds got faster over time, these numbers got larger and larger, to the point where in 2023 a number of 210,000 iterations is recommended for PBKDF2 using SHA-512 [8]. This results in a big increase in the time it takes hackers to brute force a large number of passwords, however, normal use by single users trying to log in isn't affected on the same scale, as the time it takes to run these many iterations on a single password is insignificant, while for a hacker it takes much longer to try and

crack millions of passwords hashed like this. Salting is also part of these functions, so dictionary and rainbow table attacks are also deterred.

Other password based key derivation functions like scrypt use a memory intensive algorithm, which makes it costly to perform large-scale hardware attacks.

1.1.2 Attack Patterns

The three main types of attacks that hackers use when cracking passwords are brute force, dictionary, and rainbow table attacks. It is important we talk about these, as the type of storage used for passwords dictates what attacks are most likely to succeed.

Brute force attacks simply mean trying every possible combination of characters up to a maximum length, hashing them, and seeing if the hashes match the database. Although not overly complex, these types of attacks are still very effective against conventional hashing functions, especially if there is no salt, and are often the only option if salt is used.

Next are precomputed attacks, such as dictionary attacks [9]. These are a way of cracking passwords by systematically trying every word or combination of words in a dictionary. These dictionaries are usually predefined by hackers and are made up of common words, or more likely passwords that have been leaked before and are known to be commonly used, such as "password123". Other functionalities can be added to these types of attacks based on known user behavior when setting passwords, such as l33t speak [10], changing the 'o' character to '0' in the password for example, or combining multiple words, such as "password" and "1234".

Finally, the last, most common type of attack is rainbow tables [11], which are a space-time tradeoff. This type of attack takes less time but takes up more storage than a brute force attack. Essentially, hash chains are used, which are smaller than the actual hashes, and are stored in a large list called a "rainbow table". Then, when trying to crack the hashes in a database, these hashes are compared to the hash chains inside the table, and if a match is found, the hash is retrieved from the table, as well as the actual password it came from.

There are multiple other types of attacks that hackers can use to find out a user's password, such as phishing attacks, man-in-the-middle attacks, or keyloggers, however we are not interested in those in this project.

1.1.3 HaveIBeenPwned

HaveIBeenPwned [12] (abbreviated HIBP) is a website dedicated to spreading awareness to users about the number and frequency of security breaches. Its creator, Troy Hunt is a cybersecurity expert, with a vast amount of experience in the field, with an award for Most Valuable Professional for Developer Security [13] while working as a Microsoft Regional Director. The website runs on a massive database of leaked emails and passwords dating back to 2007 and to the present. The database has more than 840 million unique leaked passwords, and, if we count every instance in which a password was leaked, that number goes up to over 5 billion passwords. The site is also being used by multiple organizations to check user's passwords at registration, login, or password change against the database of HIBP, instructing them to choose another if theirs is present in it. Also, December 2021 was the date on which Troy Hunt announced the collaboration of the website with the NCA and FBI [14], which help add more compromised passwords to the site's database. Because of all this, we decided to use HaveIBeenPwned as our main information source about past leaks.

1.2 Key Objectives

The main aim of this project is to assess the current/recent state of affairs of password storage in the wild based on data on password leaks from HaveIBeenPwned, and also see what the state of the actual passwords is based on some available databases.

Key objectives:

· Analyze past password database leaks (how they happened, what was the storage method, what attack was used if known, etc.)

· Explore previous research on these leaks and extract key information (research on the hash function used, attack used, etc.)

· Show the evolution in the usage of hashing functions, as well as the overall picture with data from HaveIBeenPwned

· Gather dehashed/plain text password databases from leaks

· Show the distribution in password strength of these databases using ZXCVBN [15]

· Draw conclusions based on the results of the analysis with regards to the different storage methods

2. MOTIVATION

There is obviously a disconnect between what security experts know to be the correct way of handling password storage and what is done in practice, as evidenced by data observed in the large number of discovered leaks and studies done on these issues. This project attempts to assess this disconnect, see how large it is and how it evolved over time, as well as trying to come up with hypotheses for changes in this evolution. The reason why we chose to tailor our research around data from leaks is because they represent concrete evidence of how a company handled their password storage at the time of the leak and cannot be combatted.

There has been a lot of research and testing done on hashing functions over the years. All this research has proven why PBKDF functions are much more secure than the conventional hashing functions. However, despite all this, recent leaks show that there is still widespread use of functions such as MD5 or SHA-1 in password databases. This can be because developers are not aware of the importance of password security, or their companies did not instruct them to implement the appropriate security measures.

In the example of the 8tracks leak from June 2017 [16], the company states in their press release that *"Passwords on 8tracks are hashed and salted, meaning that even we can't tell you what your password is by looking at the database. Although the decryption of one user's password through brute-force techniques is unlikely, we recommend that users change their password on 8tracks and any sites on which they may have used the same password to ensure their personal security."* This is more of a "saving face" statement, which has the goal of making their users think that the company did nothing wrong when storing their passwords. However, the hash function they used was SHA-1, which, even if salted, is not up to the security standard.

We can see that not enough importance is given by companies to password storage. In 2021 alone, an estimated 2 billion passwords were leaked by hackers [17]. These leaks usually result in reputational and financial damage on the organization's end, and, depending on what type of account was breached, multiple other types of inconveniences for the user. This project attempts to use data on leaks available on HIBP, with the aim of showing how the situation is "in the field". By also analyzing password databases that were leaked, I am hoping to show companies the importance of the storage methods of these passwords based on real-world cases.

3. RELATED WORK

To the best of our knowledge, there hasn't been a formal attempt to show how the usage of hashing functions has evolved over the years. We will attempt to do this using information on discovered data breaches from HaveIBeenPwned. We are aware that this might bias our results, as one could argue that websites that use inadequate passwords storage methods are more likely to have their data stolen. However, because companies do not willfully share information about their password storage, and to the presence of well-known names such as LinkedIn, NetEase and Adobe in the leaked databases list on HIBP, we would argue that this is the closest we can get to getting an actual picture of the state of password storage on the internet. The other option would have been to go directly to companies and ask them what they use for password storage, however we doubt most companies would comply with this request considering it concerns the security of user details, and you could argue that the majority of the companies that would reply will be ones that are using appropriate hashing for their passwords, which introduces bias as well.

There has been a lot of work done over the years researching the performance of hashing algorithms [18], and the effectiveness of different attack patterns on these algorithms [19]. Enough to know which hashing algorithms are worse than others and by how much, and the effectiveness of custom attack patterns against specific databases, which is why this study will not focus on these aspects. Instead, this project will attempt to compare data from different leaks, and see if there are any differences between passwords used by users based on different characteristics, such as website functionality, region, etc.

The study by The Imperva Application Defense Center (ADC) [20] analyses the complexity of passwords in the RockYou database leak from 2009. It finds that about 30% of the users in the leak did not have passwords longer than 6 characters, and about 60% of users use a character set of only letters and numbers. Both these aspects are used in the calculation of password entropy, which we calculate for the databases we found using the ZXCVBN library [15].

The most relevant paper related to our work was by Ntantogian et. al, "Evaluation of password hashing schemes in open-source web platforms" [21]. This paper attempts to go to the source of the problem by analyzing the hashing algorithms used by CMS platforms on top of which most websites are built on. One of the talking points of the paper is that the default settings for most of these platforms are not up to a reasonable security standard, not specifying a minimum password length, salt used in hashing, or a low number of iterations when hashing functions. Since these CMS' are meant to allow users with little to no technical knowledge to be able to create websites, we can see why these default settings are important. Some users may have little to no knowledge about password storage security to know to change these settings, and so this provides insight into one of the reasons why this is still a problem to this day.

It also presents a detailed cost analysis on the time cost it takes attackers to crack passwords based on brute-force and dictionary attacks on passwords hashed with different algorithms, which is one of the reasons why this study doesn't go into that.

This work also used only datasets of passwords that came from leaks where they were stored in plain text to not bias the result. We decided to also use dehashed datasets, as one of the points we wanted to study was if the strength of passwords changes with the password storage method. Even if we won't have the full database of passwords available, we have a reference total number available from HIBP, and the missing passwords will also give an idea of the strength of the hashing used.

Another relevant work is the one by Naiakshina et. al [22], which paints a picture on how developers do not think of security as a primary concern, which was one of the points I touched on in the Motivation section.

4. DESCRIPTION OF WORK

The first part of our work was to build a database of all the leaks on HIBP's "Who's been pwned" page that contained passwords. It was quite tricky getting the initial data to its final state, as we had to decide how much time we wanted to dedicate to the web scraping script we used on HIBP, and how much work we could do manually, as it was not worth the time and effort to try and get the script to write all of our data perfectly for us. As a result, there was a great deal of semi-manual data collection we had to conduct. Data which we needed and could not find on HIBP we had to search for from other sources, and some leaks needed manual intervention as well, as our script could not account for the discrepancies in them, such as leaks using multiple hashing algorithms, leaks that were discovered in plain text but were not originally stored this way and leaks for which whether they used salt or not was not mentioned. This database served as the basis for our analysis on the usage of hashing functions.

For the next part of our project, we found leaked databases of passwords in plain text. We could not pick and choose the password databases we wanted to analyze and had to rely on whatever we could find posted on public hacking forums. This is because we did not want to use databases that were used in many studies before (RockYou). Because of this, our databases are not the largest or most recent, as the reason we could find them was because they were no longer that valuable to hackers, which is why they decided to post them in public. More recent databases could also be found, however they usually are for sale, and due to ethical concerns we did not want to spend any money on the databases we analyzed.

Even though the databases we analyzed were all in plain text, not all of them had their passwords stored this way originally, as some of them were cracked from their previously hashed forms (reasoning for this in Related Work section). We analyzed these using the ZXCVBN library [15]. ZXCVBN is a password strength measuring tool developed by Dropbox which uses multiple factors like entropy, l33t speak, words and other factors for coming up with a score for a password. We used this score to analyze the passwords inside these leaks, which gave us insight on how they change based on the type of website they came from.

5. METHODOLOGIES

5.1 Data Gathering

For creating our database of leaks we used a web scraping [23] script which created a database with the following columns, based on data about each leak from HIPB: the name of the website that had their data leaked, the date the leak happened, the date the leak was added to HIBP, the number of passwords leaked, the hashing algorithm, if salt was used, and three binary columns which each were used to signify if the scraping script was not able to find the hashing used for the leak, if there were multiple hashing methods used in the database, and for plain text leaks, if it is questionable whether the passwords were actually stored in plain text. If the value was 1 for any of these three columns, it required some manual input from us.

When the hashing could not be found for a leak, we had to go through the description of the leak on HIBP ourselves to see if it

was mentioned. If it was not, we tried searching the articles linked on HIBP for the leak that maybe mentioned it. If it was not there either, we used breaches.net [24] as our second source, which provided data to HIBP for some of the leaks. If after some time we were still not able to find the hashing used, we set it as 'Unknown'. We also had to set the salting for some of our leaks as 'Unknown', as we could not find any information on whether salt was used for them or not. Initially we thought about setting the value for these to 0, however for some of the leaks that did not have this information mentioned on HIBP we did manage to find articles that mentioned them having salt, so we decided to keep them as 'Unknown'.

Next were the leaks that had multiple hashing methods used. An example of a leak like this is the MyFitnessPal leak, which used SHA-1 hashing for older accounts and bcrypt for newer ones. When there were multiple hashing methods present in a leak, we added a new line with the same information, except the hash function, for each different hashing function used. We also left the number of passwords for the new lines blank. We then tried finding the distribution of passwords between the different hashing functions and set it accordingly. If we were not able to find it, the number of passwords for the new lines was left blank, and as a result these leaks were not used in some of our later analyses on the number of passwords leaked.

Finally, there were the plain text leaks that were questionable whether they were actually stored this way, or they were cracked from their hashed forms and only leaked in their plain text form. To decide what leaks were like this, we manually went through all the plain text leaks in our database and checked the time between the leak happening and it being discovered/added to HIBP. For example, the Lookbook leak happened in August 2012, however the data was only put up for sale in June 2016 and contained plain text passwords. That is almost 4 years of time for the hacker to crack a large majority of the passwords. For reference, 16 out of the 70 plain text leaks in our database are questionable. As a result, we had to remove these leaks from most of our analysis, as most of it was concerned with the hashing of the passwords, and it was not reliable to include them.

We also had to remove entries of credential stuffing lists or of collections of leaks, like the Anti Public Combo List, as these entries are likely to contain other leaks from our database due to their size and would have skewed our analysis.

All the manual changes we made to our database after running the web scraping script are kept in a log with motivation for them for easy future reference.

5.2 Data Analysis

Our analysis of the database of leaks was split into three parts: the overall picture, the evolution over time and the salt usage.

When analyzing the data we gathered, we had to make sure that we did not use inappropriate data for the analysis we wanted to conduct. For example, when we were concerned with the number of leaks for each specific hashing method, we included the leaks that contained multiple hashing methods and for which we could not find the password distribution. However, we did not include these leaks when analyzing the number of passwords leaked per hashing method, as we did not know the actual numbers for the different methods. For this, we created a new data frame which did not contain these leaks and that we used whenever our analysis was concerned with the password numbers.

When making graphs of our data, we could not represent each hashing method separately, as there are 30 different ones in our database. Instead, we only represented the more common ones: bcrypt, MD5, plain text, SHA-1, SHA-256, SHA-512 and Unknown. The rest were combined and represented under the 'Other' name. They are presented in the following table, along with their frequencies:

Table 1. 'Other' hashing functions

Hash Function	Frequency	Hash Function	Frequency
vB	18	CRC32	1
IPB	9	Magento	1
PBKDF2	7	md5crypt	1
MyBB	4	NSLDAPS	1
phpBB	4	NTLM	1
phpBB3	4	SHA-1bcrypt	1
		CUSTOM	
scrypt	3	SHA2-384	1
argon2	2	sha256crypt	1
base64	2	SMF	1
DEScrypt	2	WHMCS	1
Wordpress	2	XF	1

There was one outlier we had to remove when analyzing the evolution of the average number of passwords leaked over the years per hashing function, which was the MySpace leak from 2008 [25]. It was the only SHA-1 leak from that year, and due to its size of just under 320 million passwords, it made our graph harder to read, and so we decided to remove it.

When analyzing the salt usage, we did not include plain text leaks, as it is not possible for them to have a salt.

5.3 Password Databases Analysis

The password databases we analyzed were all found on the internet on different forums for leaked data by hackers. Due to the scarcity of the data, we could not come up with a methodology for gathering it, as we had to just get whatever we could find.

The next step was to remove all other data from the files besides the passwords. Because all the databases came from different sources, each one required a different preprocessing script to extract the passwords from them. We made sure not to affect the passwords in any way when doing this (one of the mistakes we made at the beginning was using the strip() function in python, which for some leaks removed the whitespaces at the end of passwords). After getting the passwords we saved them in text files, which we used to have a look at to make sure they were correct, as well as pickle [26] files, which we used when reading them to calculate the ZXCVBN score, as this was faster than reading them from the text files. For the plain text leaks we found, we updated the 'Number_of_Passwords' column in our database with the number of passwords we found for them, if the storage method was plain text. If the passwords we found were cracked from their hashed forms, we kept the number from HIBP as our number of passwords.

For calculating the ZXCVBN scores we had to use the multiprocessing library [27] in python, as this significantly reduced the time it took to calculate the scores for our larger databases. To make sure our script worked correctly, we also coded a more bare-bones one which we used on a smaller sample of one of our databases. We then used our multiprocessing script on the same sample and then made sure the results were the same. We then made histograms for all the separate leaks' scores and added the percentage out of the total on top of every bin.

6. RESULTS

All figures are available in the Appendix for better visibility, including some which could not be added due to the page limit.

6.1 Database of Leaks

The last time we ran our web scraping script on HIBP before starting our analysis was the 17th of January 2023. At that point we had 524 leaks in our database. We then re-ran our script on the 8th of April 2023 and added 13 new leaks to our database, which brought our total up to 537. That gives us an idea of how password leaks are happening right now, with 13 leaks in just under 4 months and around 3.2 leaks per month. All the 13 new leaks are relatively recent, with 2 happening in 2019, 1 in 2020, 1 in 2021, 4 in 2022 and 5 in 2023. This supports our hypothesis that the lack of leaks in recent years is due to them not being discovered yet, not them not happening. This is reinforced by the average time between leaks being discovered and added to HIBP of 1.4 years, with the earliest being added the same day they were discovered, and the latest being 12 years and 5 months. The total number of passwords leaked within the breaches in our database is just over 5 billion.

Table 2. Top 10 password leaks

Leak	Date	Number of passwords	Hashing	Salt
MySpace	01-Jul-08	359,420,698	SHA-1	0
Wattpad	29-Jun-20	268,765,495	bcrypt	1
NetEase	19-Oct-15	234,842,089	plain text (q)	0
Zynga	01-Sep-19	172,869,660	SHA-1	1
Adult FriendFinder (2016)	16-Oct-16	169,746,810	SHA-1	?
Linkedin	05-May-12	164,611,595	SHA-1	0
Dubsmash	01-Dec-18	161,749,950	PBKDF2	1
Adobe	04-Oct-13	152,445,165	?	?
MyFitnessPal	01-Feb-18	143,606,147	SHA-1 & bcrypt	? & 1
Canva	24-May-19	137,272,116	bcrypt	1

There are 16 questionable plain text leaks that we had to exclude from most of our analysis, as we were not sure if the passwords were stored in plain text for them or they were cracked and subsequently leaked in plain text. They total to over 467 million passwords.

There are 33 leaks which had passwords hashed using multiple methods in their data. Only one of them had three different methods, the others all had two.

There were 98 leaks for which our algorithm was not able to find the hashing algorithm used in the descriptions on HIBP and we had to manually find it ourselves, most of which we got from breaches.net [24], which we used as our secondary source for hashing methods.

Table 2 represents the top 10 leaks in terms of number of passwords leaked.

What we can see from this table is that half of the leaks in the top 10 used SHA-1 as their hashing function for their passwords (MyFitnessPal only for older accounts). This would suggest that, for some reason, bigger companies tend to use SHA-1 as their hashing function. This hypothesis will be reinforced when we look at the average number of passwords leaked per hashing function in the next section.

One other hypothesis we came up with that would explain this finding is that the reason why these companies used SHA-1 is because that is what they used at the start, and as they grew larger and other, better security standards came up, it would have been too expensive for them to migrate hundreds of millions of passwords from SHA-1 to another hashing function. This is supported by the fact that all of these companies are relatively old, the latest one being founded in 2007, which would give them time to grow to a large user base (MySpace grew this large in only 5 years between its foundation and the leak happening). The MyFitnessPal leak is also evidence of this, and it is evident that they did a switch from SHA-1 to bcrypt at some point, however they didn't change the passwords hashed with SHA-1 to bcrypt (or those are for people that stopped using their service and didn't log in for a long time, and as a result they didn't have a chance to change their hash to bcrypt, hard to say for certain). As for why SHA-1 specifically, all these companies are based in the United States, where SHA-1 was designed by the NSA and was a *"U.S. Federal Information Processing Standard"* [28], which explains how these companies are linked. By 2011 when SHA-1 was deprecated [28], all of these companies would have grown their number of users and could have decided it was not worth it to change their hashing. This lays the groundwork for future work to be done on this, specifically on how the region companies are based in influences the hashing used, and which regions use better/worse hashing overall.

6.2 Overall analysis

6.2.1 Frequency of Hashing Functions

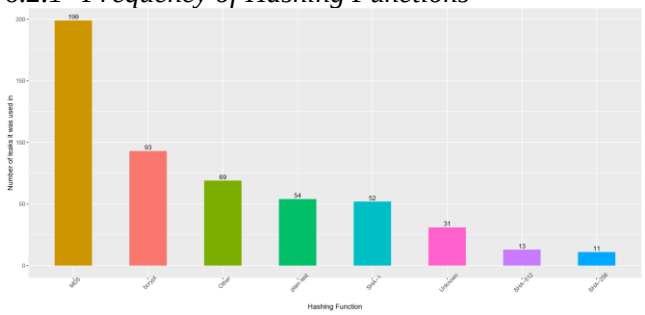

Figure 1. Usage of hashing Functions

Figure 1 shows the total usage of each hashing function in our database. We can clearly see that MD5 is the dominant hashing function with more than double the amount of bcrypt, the second one out of the total. This can be explained by the paper by Ntantogian et. al [21] we discussed in the 'Related Work' section, that showed that some CMS platform, including the most popular one, Wordpress, still used MD5 as their default hashing method. Our analysis may indicate that most people did not bother changing this default hashing method to a better one. In the centre of the figure, we have plain text and SHA-1, both not ideal when storing passwords. It is not a good sign that bcrypt is the only good hashing function present here, with functions like scrypt, argon2 or PBKDF2 only appearing in single digits. If we were to add them all up, only 105 out of 537 leaks used appropriate hashing for their passwords, which is just under 20%.

6.2.2 *Passwords per Hashing Function*

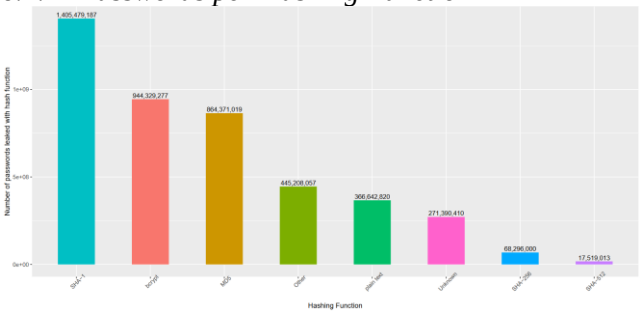

Figure 2. Passwords per Hashing Function

Figure 2 shows how many passwords were leaked for each hashing function. For this figure we did not include leaks which used multiple hashing functions that we did not know the distribution of passwords of. Here we see that, even with almost 4 times as less as many leaks as MD5, SHA-1 is the leader by a wide margin, almost 460 million more passwords than number 2, bcrypt. This supports the observation we made based on the top 10 leaks in our database, that larger companies tend to use SHA-1, specifically from the US. We can also see that bcrypt overtakes MD5 with less than half the leaks, which indicates that bcrypt is used by larger companies as well, and that MD5 is used by smaller ones. This might be as a result of bigger companies employing security experts and valuing security more than websites with much smaller, or even single developers would.

6.2.3 *Average Passwords per Hashing Function*

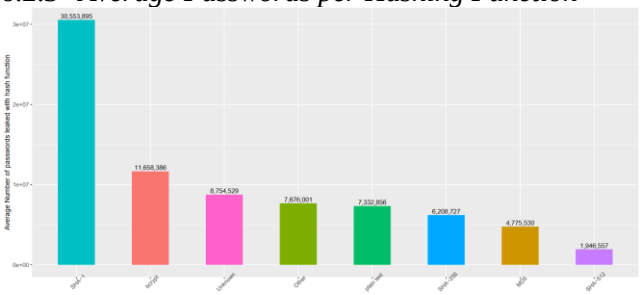

Figure 3. Average Passwords per Hashing Function

Figure 3 shows the average number of passwords leaked for each hashing function. For this, we simply divided the number of passwords by the number of leaks, without including the leaks with multiple functions, as explained earlier. Here it becomes even more apparent how SHA-1 is used by much larger companies than any other hashing method, while MD5 goes even farther back, showing that even if it had the largest frequency, most of it was from smaller leaks.

6.3 Analysis over time

This part of the analysis had the purpose of putting our previous results on the time axis, to see how they evolved over time. Of note is that HaveIBeenPwned was created in 2013, and so the rise in leaks before that year should not be taken as a matter of fact. Likewise, the lower number of leaks in recent years is not actually representative of reality, but instead leaks in recent have not been discovered yet, as we stated earlier in the 'Leaks Database' subsection of the 'Results' section.

6.3.1 *Number of Leaks Each Year*

When looking at the number of leaks available each year we saw that the highest rise was from 2014 to 2015, which led into the highest peak in 2016 with 74 leaks in one year. After this we saw the number start to stabilize around 45-60 leaks per year.

6.3.2 *Hashing Functions Used Each Year*

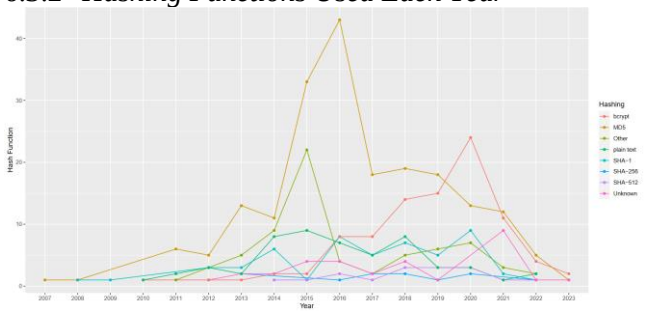

Figure 4. Hashing Functions Used Each Year

Figure 4 shows the same information, but this time separated by hashing functions. We can see that in 2016 when the highest peak happened, the function that had the highest part of it was MD5. We can also see a sharp decline in the use of MD5 after this year, and as time goes on it gets overtaken by bcrypt, which is a good sign. All the other functions are on the same level throughout the years.

It is hard to say exactly what caused the peak of MD5 in 2016 and sudden decrease in 2017. Since we've seen how important the default hashing functions used by CMS platforms are when developers are building a website [21], it could be that one such popular platform decided to change their default hashing to a different one from MD5, however we have found no evidence to support this claim. Such work can take up a lot of time when we factor in that each region can have different regulations on security and different factors, which are hard to pinpoint now, might have influenced these changes in the past. It does, however, create opportunities for future work on this subject.

6.3.3 *Number of Passwords Leaked by Year and Hashing Function*

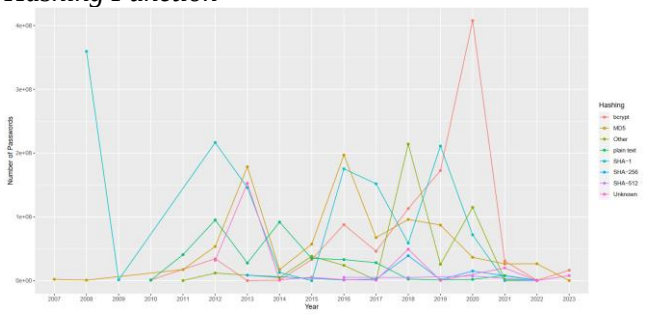

Figure 5. Number of Passwords Leaked by Year and Hashing Function

Figure 5 shows the evolution in the number of passwords leaked by hash function. Here, the second highest peak is the one in 2008, which represents the MySpace leak, the biggest leak in our database. The only peak that surpasses it is the bcrypt one in 2020, which took 24 different leaks to do so. We can see that MD5 doesn't have the same presence in this graph as the previous one, showing again that the websites that use it tend to be smaller in size. To contrast this, bcrypt is much more visible here, especially in more recent years. Throughout the years we can see the different spikes in SHA-1, showing how the larger leaks in the top 10 happened.

6.3.4 Average Number of Passwords Leaked by Year and Hashing Function

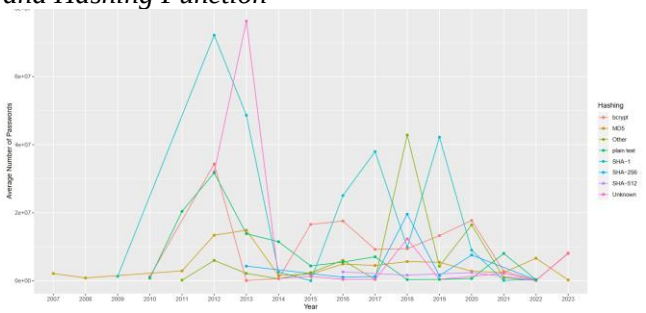

Figure 6. Average Number of Passwords Leaked by Year and Hashing Function

Figure 6 shows the evolution in the average number of passwords leaked each year. For this graph, we had to remove the MySpace leak, as it was very much an outlier. Being the only SHA-1 leak in 2008 and also our largest one, it made our graph much harder to read due to it being very high up. Even so, we can still see the larges spike in SHA-1, showing again how the few websites that do use it tend to be of larger sizes. The 'Unknown' peak in 2013 is due to the Adobe leak , as there were only two leaks of this type in that year, and Adobe is our 8th largest one. This graph also shows how much lower MD5 is when looking at the average perspective, with bcrypt clearly above it from 2015 to 2020.

6.4 Salt analysis

When it comes to salting, we noticed that the state of salting is not looking bad, as more than half the leaks in our database used a salt when hashing their password, specifically 61.3% out of 468 leaks, as we did not include plain text leaks in this analysis. Even if all the leaks for which we did not find information on salting for did not actually use salt, the majority still did. Of note is that the second most used hashing function, bcrypt, uses salting by default, and as we've seen default settings do have an influence on the outcome when it comes to password storage. Table 3 shows the distribution of salting for the hashing functions where programmers could choose whether to use salt or not. It is only for functions that we have a sample size larger than 10 of.

Table 3. Salting in hash functions

Hashing	Total	Salt	No salt	Unknown
MD5	199	99 (50%)	52 (26%)	48 (24%)
SHA-1	52	23 (44%)	8 (15%)	21 (40%)
vB	18	13 (72%)	0	5 (28%)
SHA-512	13	10 (77%)	0	3 (23%)
SHA-256	11	5 (45%)	1 (9%)	5 (45%)

What we can see from this table is that when developers have a choice on whether to use salt or not, the percentage of websites that do use it goes down compared to the overall analysis. The highest loss in proportion is for SHA-1, which goes down by around 17%, however the proportion of leaks with 'Unknown' salt is at a high 40%, so it is hard to say for certain this is the worst case, since there are still many leaks that we do not know if they used salt. We can also see a decrease of 11% for MD5, this time with only 24% Unknown and a much larger sample size, which is a much better indicator that when salt is not default to the hashing function, a considerable number of developers won't use it. The other three hashing functions do not have a large enough sample size for us to draw any solid conclusions. It is also worth mentioning that we did not have other very important information about the salts used, which would have made this analysis much more substantial, like the salt size, which has a big impact on how much harder it makes the passwords to crack.

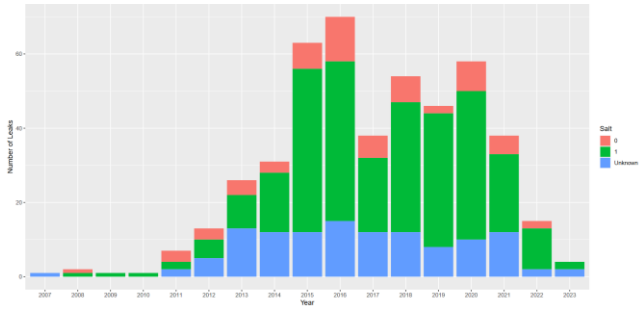

Figure 7. Salt Usage by Years

Moving in the same direction as our previous analysis, figure 7 shows the evolution of salting over time. We can see that in all years the majority of leaks did use salting, with the minority not using any salt. However, considering that we've known about salting since the 80s, the fact that it makes dictionary and rainbow attacks impossible and it also isn't expensive to compute for companies, there shouldn't be any reason for any website not to use salting when storing their passwords.

6.5 Password Databases

In this section we will be analyzing the password databases we found using the ZXCVBN library and looking at the scores returned by it, as well as the number of guesses in log10 form for the passwords. ZXCVBN scores are represented from 0 to 4, with 0 being the weakest and 4 being the strongest. Table 4 shows the leaks we managed to find the databases for and Table 5 shows their distribution of ZXCVBN scores.

Table 4. Leaked Databases Analyzed

Leak	Date	Number of Passwords	Hashing	Salt
ClearVoice Surveys	23-Aug-15	14,082,399	plain text	0
Unico Campania	19-Aug-20	166,173	plain text	0
Pokémon Creed	08-Aug-14	139,631	plain text	0
Gamigo	01-Mar-12	8,237,166*	MD5	0
Warmane	01-Dec-16	842,965*	MD5	1

6.5.1 Pokémon Creed

Pokémon Creed is an RPG website based on Pokémon. We can see that it has by far the worst distribution of scores. In addition to this, it is also the smallest of the databases we analyzed. Considering what the website was used for, it is a safe assumption that its users would be younger in age, which could be one of the explanations for the bad scores. This database also shows how effective dictionary attacks can be against websites that have a very specific function. By simply looking at the passwords in the database, we can see many passwords containing the word "pokemon" or names of other Pokémon, which makes a custom dictionary extremely effective when trying to crack such passwords. Of course, these passwords were stored in plain text, so in this case there would be no need for it.

6.5.2 Unico Campania

Unico Campania is an Italian public transport website for the Naples region. This database revealed to us one of the problems with using ZXCVBN for this analysis, which has to do with the language the passwords are in. The ZXCVBN does a lot of other analysis before coming up with a score for a password, such as the different sequences in the passwords, what they represent (names, years, etc.), comparing them against other commonly used passwords, detecting l33t speak, etc. However, all this functionality is based around the English language. The fact that most of the passwords in this database were in the Italian language meant that the algorithm would get its analysis wrong. For example, if we run ZXCVBN on the password "motorcycle", it gives it a score of 1. However, if we run it on the password "motocicletta", which is the Italian word for "motorcycle", and is also a password present in our database, it gives it a score of 4. This is because the library does not inherently support the Italian language, and so it splits the word up into three sequences: 'mot', 'ocicl' and 'etta', instead of keeping it one word, like with "motorcycle". This brings up one of the issues we talked about before in this paper, which is that of changing default settings. Besides changing the default password hashing used by a CMS when developing a website, adding salt to your hashing (this shouldn't be an issue if using an appropriate hashing algorithm), we now also have the issue of making sure that if your website uses a password strength estimator like ZXCVBN, it is built around the user-base of the website. This way, users won't get positive feedback for passwords that are much less secure than what the algorithm shows.

Table 5. ZXCVBN Scores

Leak	0	1	2	3	4
ClearVoice Surveys	4.7%	21.4%	17.1%	54.1%	2.4%
Unico Campania	1.9%	27.9%	23.7%	25.8%	20.7%
Pokémon Creed	15.2%	46.6%	21.7%	12.8%	3.6%
Gamigo	6.0%	27.2%	21.3%	42.6%	2.9%
Warmane	5.0%	33.7%	32.4%	22.4%	6.4%

6.5.3 ClearVoice Surveys

ClearVoice Surveys is another database which provided us with interesting results for our analysis. It is a company that says it provides users with money in return for completing surveys. We can see that it has a good distribution of passwords according to our ZXCVBN analysis, 54.3% having a score of 3. However, when looking at the passwords in the database, we noticed large sections of passwords that had a length of 8 characters and contained random upper-case letters and numbers, or passwords that had a length of 10 characters containing either upper or lower-case letters in combination with numbers in a random sequence. We came up with two possibilities that could explain this. The first, less likely one, is that the website gave the users these passwords when registering, which they never changed afterwards. Presently, the website does not use such a system, however the database that was leaked was a backup from 2015 that was later leaked in 2021, so we cannot know what type of system was in use in the past. We came up with this hypothesis when we forgot the password we used on one of the hacking forums we were searching for databases, and when trying to change it the website gave us a "temporary" password to log in with, which was similar in format with the ones we observed in the database. The other, more likely option is that the accounts using these passwords are bots and not real people. Since the website promises monetary compensation, we believe it is not that unlikely that a large number of accounts would be bots used by people to gather money at a much faster rate than with a single account. Bots would also not have as much trouble with remembering complex, random passwords than humans, which would explain the high percent of passwords with a score of 3.

6.5.4 Gamigo & Warmane

These final two databases were not stored in plain text to begin with, but instead were hashed using MD5 and later cracked by hackers. As a result, not all the passwords are present in the databases. Gamigo is a German online game publisher. Despite this, we do not think the language problem is a as big as with Unico Campania, as their main website is in the English language and accessible by anyone. Unico Campania represented a much more specific case, as it was a website for public transportation inside of Italy, which results in the vast majority of passwords being in the Italian language. Warmane is an online service for World of Warcraft private servers. The main comparison between these two databases is salting. Gamigo is described on HIBP as using *"weak MD5 hashes with no salt",* and as a result almost all of the passwords were present in the database we found, only missing just under 6.5 thousand passwords from the number listed on HIBP. That is less than 0.08% of the passwords. By contrast, Warmane, which used salting when hashing the passwords, had just over 273 thousand passwords missing out of the total from HIBP in the database we found. That is 24.5% of passwords which were not able to be cracked as a result of the salt. This comparison helps us draw to conclusions. For one, it shows the importance of salt, as a much larger proportion of passwords was not able to be cracked as a result of it. The fact that both websites are related to videogames and both databases were obtained from the same source on the same forum only strengthens this conclusion. The other is showing how unsecure MD5 is. For Gamigo, almost all of the passwords were cracked from their MD5 hashes, making it almost as secure as storing them in plain text, while for Warmane, even with salt, 75.5% of the passwords were still cracked.

We can see that the score distribution for Gamigo is very similar to the one for ClearVoice Surveys. If we go with our conclusion for ClearVoice, that such a shape is representative of bots being present on the website, we could say that there are less present on Gamigo, as the percent of passwords with a score of 3 is less than on ClearVoice, with a higher percentage in the 1 and 2 scores.

We can also see a similarity between Warmane and Pokémon Creed, both websites being related to video games. The distribution for Warmane is better than that of Pokémon Creed, with higher percentages in the 2 and 3 scores and lower in the 1 score, this with 273 thousand passwords still missing from its database, which should be the most secure ones from it. Since these passwords were hashed and salted and the ones for Creed were stored in plain text, this could be the start of an argument that as password storage improves, user passwords also improve, as this would be representative of the website and its user base being more security-conscious and more "mature".

7. SUMMARY AND REFLECTIONS

7.1 Conclusions

In this paper, we have analyzed the overall picture and evolution of hashing functions used in password storage, as well as analyzing the passwords themselves and contrasting their strength with the aim of the website they came from. We have also highlighted the

difficulties in doing such a study, such as the bias in our database, the scarcity/lack of information (salt size, number of iterations for a hash function) which would have added a lot more depth to this study, ZXCVBN being unreliable in languages other than English, and also the amount of time and extra digging required to collect up-to-date, correct information about data breaches. Overall, we have seen that the situation is not great, with bcrypt being the only good hashing function with a large presence, and with MD5 having almost double the amount. However, further analysis on the number of passwords leaked with each function and the evolution over time revealed that things aren't as bad as they seem and look to be taking a turn for the better, with bcrypt being used by much larger websites than MD5 and, as of 2020, overtaking it in usage. The analysis of the password databases showed how developers need to account for what their main user demographic is and use appropriate systems, while also showing how the password storage method and website purpose can be used to get an idea of how the password strength distribution looks like. It also showed how websites with a specific theme, such as Pokémon, are likely to be targeted with highly specific dictionary attacks based around that theme.

7.2 Future Work

Other relevant work than the ones highlighted in sections 6.1 and 6.3.2 could be automating our analysis. Having a website that automatically adds password leaks to our database as they get added to HIBP and updates and displays our graphs according to the new data added can be useful in keeping up to date with changes in the space of password security. We do think that even this would still require some manual input, depending on how much information is published on HIBP about specific leaks.

7.3 Project Management

The scope of my project was a lot larger when I started. Initially, I also intended to use Hashcat to try and crack hashed password databases myself and do an analysis on the different attack patterns that can be used. However, as time went on during the second semester, I realized that I wouldn't have time for this and opted to use databases that were dehashed by someone else instead.

The first semester was reserved for research analysis and data gathering, which I managed to complete, but only late during the Winter break. This was because I was initially going to gather data on leaks manually and much more in-depth, including information on how the databases were leaked, for example. As the semester went on, I got better with text processing due to one of my other modules, and as a result decided to switch from this to developing the web scraping script I used in the end. All the work I did before wasn't in vain, as I learned a lot about how companies treat leaks and how the situation is like in the field, however I didn't have much to show in terms of results.

During the second semester I worked on analyzing the database I put together and the leaks I found. This took longer than I first expected, as I was not proficient in R, the language I used to put together the graphs on my database. I never managed to gray out the 'Unknown' bar/line in my graphs without affecting the other functions in any way. I also had trouble finding databases for the second part of my analysis, as some of the forums I went on required me to reply to a thread before being able to access the leak, with a limit of 2 replies per day, and others required me to pay in order to download the leaks.

Throughout the project I had weekly meetings with my supervisor where I would showcase my work and set the plan till the week after. There were some skipped meetings when I didn't have enough to show due to other courseworks needing to be done.

As for the external aspect, I intend to submit this paper to different journals focused on cybersecurity, such as the Computers & Security journal by Science Direct. The final paper will have the missing figures.

7.4 Contributions and Reflections

I will start by saying that I am disappointed that I did not accomplish everything I set out to do at first. There were multiple times where I would panic, thinking I would not have enough interesting work done by the end of the project. I would have liked to try my hand at cracking passwords and seeing what impact the different hashing functions have.

However, writing this dissertation has made me realize that things might not be as bad as I first thought. To the best of my knowledge, there are no other studies going this in depth on the usage of hashing functions. My project may not use any novel analysis methods, however the information it presents is relevant and can be used to educate people on password storage. We also analyzed password databases which we did not find used in studies before.

In terms of LSEPI, my work did not use any personal information that could identify people, as I removed all the other information from the files I found except the passwords. Also, I am aware of the gray area surrounding web scraping, however I had a brief interaction with Troy Hunt where I showcased my work and mentioned the use of web scraping, and he did not have any issues with it. I did not involve any human participants.

8. ACKNOWLEDGMENTS

Our thanks to Troy Hunt for building and maintaining HaveIBeenPwned and making the data publicly available.

9. REFERENCES

[1] "SQL (Structured query language) Injection," Imperva, 2022. [Online]. Available: https://www.imperva.com/learn/application-security/sql- injection-sqli/. [Accessed 14 April 2023].

[2] S. Oza, "File Inclusion Vulnerabilities — Web-based Application Security, Part 9," Spanning Backup, 2023. [Online]. Available: https://spanning.com/blog/file-inclusion- vulnerabilities-lfi-rfi-web-based-application-security-part-9/. [Accessed 14 April 2023].

[3] D. Sewell, "Offline Password Cracking: The Attack and the Best Defense," Alpine Security, 2021. [Online]. Available: https://www.alpinesecurity.com/blog/offline-password- cracking-the-attack-and-the-best-defense-against-it/. [Accessed 20 April 2023].

[4] G. Ramirez, "MD5: The broken algorithm," Avira, 28 July 2015. [Online]. Available: https://www.avira.com/en/blog/md5-the-broken-algorithm. [Accessed 6 December 2022].

[5] S. Outten, "Bitcoin Transaction Validation, What Exactly Goes on Under the Hood?," Deltec Bank, 5 October 2021. [Online]. Available: https://www.deltecbank.com/2021/10/05/bitcoin-transaction- validation-what-exactly-goes-on-under-the-hood/?locale=en. [Accessed 15 April 2023].

[6] "Salt (cryptography)," Wikipedia, 22 November 2022. [Online]. Available: https://en.wikipedia.org/wiki/Salt_(cryptography). [Accessed 6 December 2022].

[7] "Pepper (cryptography)," Wikipedia, 29 November 2022. [Online]. Available: https://en.wikipedia.org/wiki/Pepper_(cryptography). [Accessed 6 December 2022].

[8] "PBKDF2," Wikipedia, 29 March 2023. [Online]. Available: https://en.wikipedia.org/wiki/PBKDF2#cite_note- RFC3962-1. [Accessed 18 April 2023].

[9] "Dictionary attack," Wikipedia, 23 October 2022. [Online]. Available: https://en.wikipedia.org/wiki/Dictionary_attack. [Accessed 6 December 2022].

[10] "Leet," Wikipedia, 30 March 2023. [Online]. Available: https://en.wikipedia.org/wiki/Leet. [Accessed 20 April 2023].

[11] "Rainbow Table," Wikipedia, 30 October 2022. [Online]. Available: https://en.wikipedia.org/wiki/Rainbow_table. [Accessed 6 December 2022].

[12] T. Hunt, "HaveIBeenPwned," 2022. [Online]. Available: https://haveibeenpwned.com/. [Accessed 21 November 2022].

[13] "Most Valuable Professional," Microsoft, 2022. [Online]. Available: https://mvp.microsoft.com/en-us/overview. [Accessed 6 December 2022].

[14] T. Hunt, "Open Source Pwned Passwords with FBI Feed and 225M New NCA Passwords is Now Live!," 20 December 2021. [Online]. Available: https://www.troyhunt.com/open-source-pwned-passwords-with-fbi-feed-and-225m-new-nca-passwords-is-now-live/. [Accessed 21 November 2022].

[15] D. L. Wheeler, "zxcvbn: Low-Budget Password Strength Estimation," *USENIX Security*, 2016.

[16] D. Porter, "Password security alert," 8tracks Blog, 27 June 2017. [Online]. Available: https://blog.8tracks.com/2017/06/27/password-security- alert/. [Accessed 6 December 2022].

[17] G. Sevilla, "Hackers leaked 2 billion passwords in 2021," Insider Intelligence, 21 July 2022. [Online]. Available: https://www.insiderintelligence.com/content/hackers-leaked-2-billion-passwords-2021. [Accessed 6 December 202].

[18] L. Ertaul, M. Kaur and V. A. K. R. Gudise, "Implementation and Performance Analysis of PBKDF2, Bcrypt, Scrypt Algorithms," Proceedings of the International Conference on Wireless Networks (ICWN). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), CSU East Bay, Hayward, CA, USA, 2016.

[19] P. Kamal, "A Study on the Security of Password HashingBased on GPU Based, Password Cracking usingHigh-Performance Cloud Computing," 2017.

[20] "Consumer Password Worst Practices," The Imperva Application Defense Center (ADC), 2014. [Online]. Available: https://www.imperva.com/docs/gated/WP_Consumer_ Password_Worst_Practices.pdf.

[21] C. Ntantogian, S. Malliaros and C. Xenakis, "Evaluation of password hashing schemes in open source web platforms," Computers & Security, Piraeus, 2019.

[22] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand and M. Smith, "Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study," in *ACM*, Dallas, Texas, USA, 2017.

[23] "Web Scraping," Wikipedia, 1 November 2022. [Online]. Available: https://en.wikipedia.org/wiki/Web_scraping. [Accessed 2 December 2022].

[24] "breaches.net," breaches.net, 2022. [Online]. Available: https://breaches.net/. [Accessed 10 January 2023].

[25] T. Hunt, "Who's been pwned (MySpace," HaveIBeenPwned, 2023. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites#MySpace. [Accessed 16 April 2023].

[26] "pickle — Python object serialization," Python, 2023. [Online]. Available: https://docs.python.org/3/library/pickle.html. [Accessed 16 April 2023].

[27] "multiprocessing — Process-based parallelism," Python, 2023. [Online]. Available: https://docs.python.org/3/library/multiprocessing.html. [Accessed 16 April 2023].

[28] "SHA-1," Wikipedia, 2023. [Online]. Available: https://en.wikipedia.org/wiki/SHA-1. [Accessed 2023 April 22].

[29] "Submitted Breach Notification Sample," State of California Department of Justice, [Online]. Available: https://oag.ca.gov/ecrime/databreach/reports/sb24-138932. [Accessed 6 December 2022].

[30] T. Hunt, "Who's been pwned (Lookbook)," HaveIBeenPwned, 2023. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites#Lookbook. [Accessed 16 April 2023].

[31] T. Hunt, "Who's been pwned," HaveIBennPwned, 2023. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites#AntiPublic. [Accessed 16 April 2023].

[32] T. Hunt, "Who's been pwned (DatPiff)," HaveIBeenPwned, 2023. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites#DatPiff. [Accessed 16 April 2023].

[33] F. Grossi, "Low-Budget Password Strength Estimation," github, 11 January 2019. [Online]. Available: https://github.com/flaviogrossi/zxcvbn-it. [Accessed 19 February 2023].

[34] T. Hunt, "Handling Chinese data breaches in Have I been pwned," 8 October 2016. [Online]. Available: https://www.troyhunt.com/handling-chinese-data-breaches- in-have-i-been-pwned/. [Accessed 21 November 2022].

[35] Z. Whittaker, "Animoto hack exposes personal information, location data," TechCrunch, 20 August 2018. [Online]. Available: https://techcrunch.com/2018/08/20/animoto- hack-exposes-personal-information-geolocation-data/. [Accessed 6 December 2022].

[36] T. Hunt, "Who's been pwned (MyFitnessPal)," HaveIBeenPwned, 2023. [Online]. Available: https://haveibeenpwned.com/PwnedWebsites# MyFitnessPal. [Accessed 16 April 2023].