Security Database Model

Daniel Hinz, Gregory Johannes, Mohamed Ali

November 2021

Abstract

We created a database to securely store the information of our users. The methods that we implemented for securing the information was hashing the password protected database. In the paper we compare the different methods of password storing and provide reasons why the type of hashing we selected for our database is the best option in terms of security.

For further security of the information of our users we implemented data privacy using K-Anonymity and L-Diversity in the database. This protected the users from being their identities being discovered in the unlikely event that our security measures failed to protect the database in the first place.

Using these methods we are able to insure that our database will function practically as we believe it does theoretically. However, given the time frame in which we completed the database and its accessing web application, we were not able to implement all the features that we wished too. These features will be mentioned at the end of the paper.

Table of Contents

1 Password Storage Overview

Here we will introduce the different methods of storing passwords on the database. Since some of the sections can be rather heavy, we though it best to ease into them. The methods of password storing is given in degree of effectiveness. For example the first method of storing passwords is in plain-text. This method is not as effective as using a slow hash to store them.

1.1 Plain-Text

This method is the most strait forward. The passwords are stored as they are typed into the database.

1.2 Encryption

Using this method the password is put through an encryption algorithm to obscure it.

1.3 Hashing (Flavored)

Hashing is the most complicated of the password storing methods which is probably why it is the most effective. Hashing has some similarities to encrypting with the difference begin that the original password cannot be obtained from the hash value. There are techniques used with hashing to make it more effective that I call flavoring. The flavorings of the hash is salt and pepper which will be elaborated on in later sections.

1.4 Slow Hashing

Like flavored hashing, slow hashing is a variant of hashing, but the speed in which it operates sets it apart. Apart from this difference, its functionality is the same.

2 Plain-text

While this is the easiest of the methods it is also the most ineffective since if the database were to be hacked into and the data table were to viewed, there would be nothing protecting the account passwords from being seen. From a security standpoint this is quite literally no security at all. As aforementioned, this method is the worse of all the options and was not considered for our database. However, there are many companies that still use this method of password storing.

The way you can identify if a website is storing your password in plain-text, or at most encrypted (This will be covered next) is by attempting to recover your password. If the website is able to display you your password in anyway, that means that your password is not protected since it was easily retrieved for you.

3 Encryption

The encryption method is a step above plain-text, but not by much. The encryption algorithm takes in plain-text and a key. The plain-text is the password that you put in, and the key is usually a string of random characters generated for the purpose of being used for the algorithm. This encryption algorithm has as output cipher-text. This text would appear to be a random assortment of characters.
However, in many cases in information security, encrypted text is still considered personal information since the information is still there, it is just obscured. With the key, the password can be seen which lies the problem with this method. If a hacker can get into the database, its feasible, that they can gain access to the data table the key is located in.
Another large reason encryption is an insufficient means of password storage is because if that key were to be obtained, then the entire table would be reduced to plain-text. Not just one user.

4 Hashing

Hashing uses a function called a hashing function (very creative) to create a hash value out of the password that is put in by the user. The password enters the hashing function as input and the output which is the hash value is a string of characters appearing to be random. When a password is entered, it is put into this hashing function and the resulting hash value is compared to the hash value generated when the password was created. If the hash values are the same, then access is granted.

4.1 Pros/Cons

While you do not have to store a key like you do for encryption, there are still some drawbacks to using a hash function.

There are a variety of attacks that can be preformed on a hash function that will allow an attacker to retrieve the password that is hashed. There are also requirements that a hash function must have to provide basic hash protection.

4.2 Hashing Attacks

There are several types of attacks that hashing is vulnerable to. These attacks do not break the hashing function or reverse the hash value. As stated earlier, there is no way to reverse the hash function to retrieve the original password.

The first type of attack is called an Analysis attack. This type of attack observes the patterns and similarities between all of the hashes on a data table in a database and gets clues that common passwords are being used. That is to say, that if two hash values are the same then the attacker knows that the passwords are the same. So if the attacker finds the password to one of the users, then they have the password to both accounts.

The second type of attack is called a Dictionary Attack. This type of attack runs through a dictionary of command passwords and puts them through the hash function to get the hash value of each of the common passwords. The attacker than compares the hash value that they just created to the entire data table to see if any of the hash values match. If they do, then the password that the attacker just hashed is the password to that persons account.

The third type of attack is called a Rainbow Table Attack. This attack uses a table that contains pairs of commonly used passwords and their hash value. With this table, since the hash values are pre-computed, all the attacker needs to do is compare the data table to the rainbow table and whatever hash values match, belong to the associated password paired to the hash value.

5 Slow Hashing

Slow hashing is very similar to hashing with a big difference. It is SLOW. Generally, you want a hash value to be quickly executed in order for the user to gain access to the account quickly. The speed a typical hash function executes is in the milliseconds. A slow hash algorithms intentionally slows down the computation type in order to deter attackers. Algorithms like bcrypt, and Argon2 use what is called a work factor to adjust the speed in which it completes the hashing of an input.

5.1 Work Factor

The work factor can be visualized as a dial that can be turned up the increase the time it takes to complete the hash, or turned down to speed up.

For For perspective, with today's computing power, a work factor of 6 can hash a input in 5ms. This was the recommended setting for the work factor when the slow hash algorithm bcrypt was published in the 90's. At that time, computers took just over 1 second to complete a hash with this work factor setting.

A work factor of 15 would take over 1 second with today's computers. With a larger work factor attackers can no longer preform 100,000+ computations per second. When an attacker is performing thousands of hash computations just for one password, this can be a strong deterrent.

6 Comparisons

Before moving on there is something that should be clarified. Dictionary Attacks and Rainbow Table Attacks are very similar and are easily mistaken for each other. The same can be said with encryption and hashing. It is understandable if these pairs of subjects can be confusing since they are so similar.

6.1 Dictionary Attacks vs Rainbow Table Attacks

Rainbow tables have two columns. One column has a password, and the other has the hash value that is generated from it.

A dictionary attack does not use this table. The attacker instead takes the password and generates the hash value manually. This has a advantage over rainbow tables because dictionary attacks work on flavored hashes (hashes that are salted and/or peppered). This does come at a high computational cost, so if the tables are not flavored, then a rainbow table would be more efficient.

6.2 Encryption vs Hashing

Encryption and hashes are both algorithms that take plain-text and make them into a string of incoherent characters. The difference is that an encryption can be decrypted. By using the key associated with the cipher-text, you can generate the original text. This is not the case with the hash algorithm (or hash function). This is because the process of creating the hash value loses the original meaning of the plain-text used to generate it.

7 Hashing (Flavored)

With that background information, we can move onto what I call flavored hashing. Again, this is taking a hash and applying a salt and/or a pepper to it.

7.1 Salted Hash

A salted hash is when a random string of characters are appended to plain-text before it is put into the hash function. When the random string (or salt) is different for each user, then the result is a unique hash for every user regardless of the password the user uses. So for example the password PW123 will have a different hash value than password PW123 with a salt #sleh.n.

The benefit of using a salted hash is that rainbow tables do not work. Those tables do not include our unique salts for each hash value in our data table. Dictionary attacks are still possible, but they are a lot slower than using a rainbow table.

The drawback to a salt is that the random string is on the data table the hashed password is on. That way when a password is entered, the salt for that user is looked up on the table, appended to the password the user typed in, and that resulting string is the input to the hash function. A attacker knows that a salted hash is used because the salt is right there on the table.

7.2 Salt/Pepper Hash

A pepper can be thought of as a second salt. It is another string of random characters that is appended to a hash value (usually one that is already hashed). The benefit of using a pepper is that an attacker has no way of knowing that the password has been peppered just by looking at it. A salted hash value, and a salt and peppered hash value look equally random.

The pepper is not stored on the data table like the salt is. There are two methods a pepper is stored.

Either the pepper is hard coded into the hash function, or the salted hash is encrypted, and the key to that encryption is what is considered the pepper.

The encryption method is usually the pepper of choice, but since peppers are relatively new to the hashing space, many pre-made hashing functions do not support peppering. This will change in the future, but until then, it is one of the draw backs of using a peppered hash.

8 Data Privacy

To achieve data privacy, the data is manipulated in such a way that you can still use it, but also protect the privacy of the people who's data you collect. This is called data anonymity. The process helps prevent attackers from identifying the person who's data is on in the database.

To do this the data must be hidden and manipulated in specific ways. Depending on what type of data attributes are on the table, and what the purpose of the data is, determines on what actions we take on it to anonymous that data. There are three types of attributes that we need to label.

Explicit Identifiers. This attribute contains data that can explicitly identify the person who's data is on the table. Information like a name or ID can help an attacker figure out with little effort who the person is.

Quasi-Identifiers. This attribute contains data that can assist in identifying someone with additional information. While this information does not explicitly expose a person, with some additional information such as information gleaned from a background information attack can lead to the identification of the subject of the data. Quasi-Identifiers are locations of someones place of residence or place of work, age, gender, and other such closely related information.

Sensitive attributes. This attribute contains information that people would consider private and not want shared without their permission. This could be credit score, job titles, or medical history. Since many people find many things to be

sensitive, this attribute can potentially cover a lot of information.

With these attributes defined, some data privacy techniques can be applied to the data to provide the anonymity that is desired.

8.1 K-Anonymity

K-anonymity is on of two techniques we have chosen to implement in our project. It is achieved when you have a set of tuples that are not distinguishable from each other. This group of tuples is called an equivalence class and to achieve this, you either suppress data or generalize it depending on your goals for the information.

Suppression removes or replaces the data so that it is either false or invalid. An example of this would be if you have a name of a person, you would completely remove the name and replace it with a star (*).

Generalization takes data and puts it into ranges. Using the previous example, instead of replacing all of the names with stars (*), we would make every male have one name, and female have a different name. If you wanted to generalize it further, you could give everyone the unisex name in order to hide their gender as well.

These techniques help create the equivalence class and protect data privacy, but there is a step further you can go. After creating equivalence classes for K-Anonymity, you can preform a data privacy technique called L-Diversity.

8.2 L-Diversity

After creating equivalence classes the sensitive data that the table is displaying can still be used to identify someone. Examples of this are the homogeneous pattern attacks and background knowledge attacks we covered in class.

By taking this sensitive data and applying a rule to the groups that k-anonymity created, we can provide further privacy. This rule is as follows: *For each group of quasi-identifiers, there can me at most 1/L of its tuples that can have an identical sensitive attribute.*

To clarify, this means if you look at any tuple in a group, there will be no more than a certain number of tuples that are the same. That number is the L in L-diversity.

9 Implementation

10 Future Improvements

For improvements to be made in the future, there are several things that would like to do that would make this database both more efficient and function better.

First of which would be implementing Hash peppering proved to be rather difficult since the native salted hash algorithms did not support it. Since our custom hash algorithm which preformed the flavored hash was not complete, we regrettably had to omit it.

Next is to hide columns that had suppressed data attributes. The database architecture that was used was SQL alchemy which was later discovered lacked functionality to preform this task.

Lastly to implement a dynamic calculation of T-Closeness. This is a additional step that can be done to provide data privacy.