Security Database Model

Daniel Hinz, Gregory Johannes, Mohamed Ali

November, 17TH, 2021

Table of Contents

Password Storage Techniques

- Plain-Text
- Encryption
- Hashing
- Hash/Salt/Pepper
- Slow Hash

Plain-Text

Storing passwords as they are typed into the data table.

Pros:

- Very easy to implement.
- Virtually no overhead.

Cons:

- Trivial to steal.
- No protection against attacker.

Encryption

- Using an encryption algorithm to obscure the true value of the password.
  - This is done by passing the plain-text though a encryption algorithm (Such as AES, or 3DES). The Plain text would be the password itself.
- The key for then encryption algorithm would be held by the system.
  - Not on the table itself, but elsewhere in the architecture. For example in the code.

Encryption Pros/Cons

Pro:

- Better than plain-text.

Con:

- If the attacker gets the key, then the entire table is as good as plain-text.
    - There are several types of attacks that could make this less difficult then you would think.

Hashing

- Using a hashing function to change the password into a hash.
- The plain-text goes into a hashing function and the output is the hash value.
- Whenever a password is entered, it would go through the hash function and the result would be compared to what is in the table.
    - Examples of these functions are MD5, SHA-1, SHA-256.

Hashing

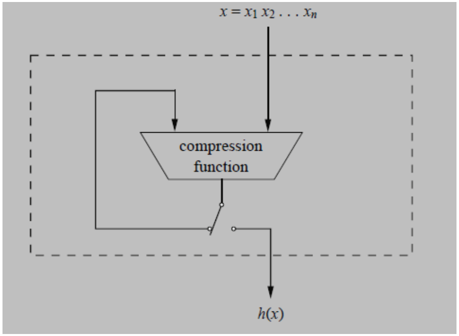

Figure: Hash Function

- Here we can see just one of the irreversible techniques the hashing function uses to get its output: Compression
- Compression takes two bit-ranges and preforms logical operation on them to produce outputs.

- Because of this loss of data you cannot get the original data, back after it goes through the hash function.

Hashing Pros/Cons

Pros:

- You do not have to store a key
- Slightly better than encryption

Cons:

- Vulnerable to different types of attacks:
  - Analysis Attacks
  - Dictionary Attacks
  - Rainbow Table Attacks
  - Hash requirements
- Pre-image resistance
  - Second pre-image resistance
  - Collision resistance

Hashing Attacks

- Analysis attacks - observe patterns and similarities between all of the hashes to get clues what common passwords are being used.
- Dictionary attacks – Run through a dictionary of common passwords and hash them as they go until a match to the password hash is found.
- Rainbow table attacks - A table of pre-computed hashes and their plain-text counterparts. Useful for passwords that are common.

Difference Between Dictionary and Rainbow Attacks

- Rainbow tables are pre-made tables of pre-computed password/hash pairs (This will become important later).
- In a Dictionary attack, the attacker manually and one by one computes the hashes and compares them to the table they are breaking. (Also important later).

Hashing Requirements

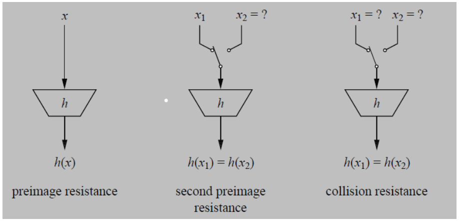

Figure: Hash Requirements

Pre-image Resistance (One-Wayness)

- Given a output of hash function (Hash value), it must be impossible to find the input.

Hashing Requirements

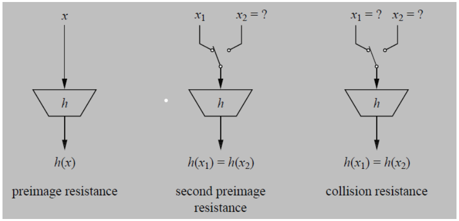

Figure: Hash Requirements

Second Pre-image Resistance (Weak Collision Resistance)

- Given a output of hash function (Hash value), it must be computationally infeasible to manufacture an identical hash.

Hashing Requirements

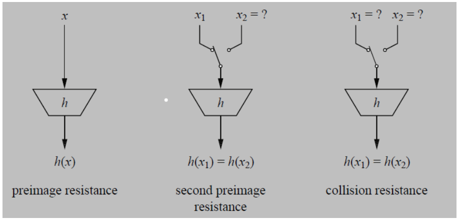

Figure: Hash Requirements

(Strong) Collision Resistance

- It must be mathematically infeasible to find two hashes that are the same given two different inputs.
- The difference here is that the goal is not to just make it hard to find any specific hash pair that is identical, but any hash pair.

The difference between Encryption and Hashing

- Both Encryption and Hashing take input (plain-text) and preforms a series of techniques to produce a string of seemingly random characters.
  - This was demonstrated by a previous group with AES.

- The difference is that an encryption can be reversed which is know as decrypting, while hashes are designed to make it impossible to retrieve the original data from the output (hash value).

Hash/Salt

- Salt - An appended string of random characters to a password before it is put into the hashing function to be hashed.
- This has several advantages that make the overhead of maintaining unique salts for each user worth the effort.

Hash/Salt Pros/Cons

Pros:

- Random string are unique for each password and stored next to their hash value.
    - Allows system to take the input from the user at login and add the salt to get the correct hash value.
- Protected from rainbow table attacks.
    - Since hash value for password: PW1234 will be different than hash value for the password PW1234 #sleh.n
- Rainbow tables do not have the value for the second hash so the pre-computed values are unusable.

Cons:

- Dictionary attacks still work since an attacker can take the salt and append it to the attempts.
- Still an tedious process, but doable with a fast enough computer.

Hash/Salt/Pepper

- Pepper can be though of as a second salt or a secret salt.
- This string of characters is not held on the table so the attacker cannot just take and use it like the salt.
- Two methods for applying a pepper:
    - Hard Code: That is to put the random string into the code so that before any value is put into the hash function, the pepper will be added.
    - Encrypt: Take the salted hash value and encrypt it. The key to the encryption would be considered the pepper instead of a random string.

Hash/Salt/Pepper Pros/Cons

Pros:

- Extra layer of protection, in many cases a salt and pepper hash is indistinguishable from a salt hash.

Cons:

- Many hashing algorithms that are commonly used do not support peppered hashing.
- The process can add extra run-time, making the user wait longer before allowing access.

Slow Hash

- Using a hashing algorithm that is intentionally slow.
  - Normally, you would want a hashing algorithm to be fast so that the user does not have to wait too long to gain access to the information they are wanting.
- Algorithms like bcrypt, scrypt and Argon 2 preform this function.
  - As a bonus they also salt the hash automatically.
- The algorithms have what is called a work factor. This is basically like dial to control how slow you want the algorithm to go.

Work Factor

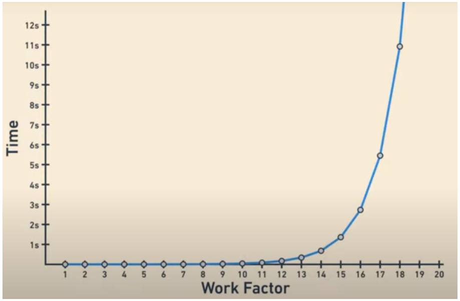

Figure: Work Factor

- For perspective, with today's computing power, a work factor of 6 can hash a input in 5ms.
- A work factor of 15 would take over 1 second.

- This is a big deal because attackers can no longer preform 100,000+ computations per second.
- The name of the game is not impossible. Just computationally infeasible.

Data Anonymity: K-Anonymity, L-Diversity

- To achieve data anonymity, it is important to manipulate the data in a way that you can still use it, but also protect the privacy of the people who's data you collect.
- To do this we have to hide and manipulate the data in specific ways.

Types of Data Attributes

- Explicit Identifiers – Name, Username, ID
    - Things that can be used to directly tie the data to a person.
- Quasi-Identifiers – Where someone lives/works, gender, age
    - Things that can assist in identifying someone with additional information.
- Sensitive attributes – Credit history, health information, job titles
    - Information that people would consider private and not want shared without their permission.

Techniques used to achieve K-anonymity and L-diversity

- K-anonymity 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.
- To achieve this, you either suppress data or generalize it.
  - Suppression - Removing or replacing data so that it is either false or invalid.
  - Generalization - Taking data and putting into ranges. Such as age or salary.
- These techniques help create the equivalence class and protect data privacy, but there is a step further you can go.

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.

L-Diversity: Making it Work

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

Improvements for the Future

- Implementing Hash peppering
- Hide columns (SQL alchemy does not have this functionality)
- Dynamically calculate T-Closeness (.1684)
  - T-Closeness was calculated for initial table (by hand), but was not implemented in code.