**PASSWORD**

A strong password is one that is hard to guess and cannot easily be found on leaked password lists. It is also one that is resistant to common attack methods like:

* Brute-force
* Dictionary attacks
* Social engineering

Characteristics of s Strong Password

|  |  |
| --- | --- |
| Length >= 12 characters | Longer passwords require substantially more time to brute force. |
| Includes Uppercase | Increases complexity—more character combinations |
| Includes Lowercase | Standard for natural language (still required). |
| Includes numbers | Improves numerical diversity — 1234 alone is weak, but mixed is better |
| Includes Special Chars | Makes patterns more difficult to predict (e.g., @, #,!) |

A weak password, such as password123, fits length and digit requirements but is easily guessed. A powerful password may look like G!tM4r$T@h9z.

How the Function Checks

|  |  |
| --- | --- |
| Check | Python Logic Used |
| Minimum length of 8 | len(password) >=8 |
| Has uppercase letters | |  | | --- | | any(char.isupper() for char in password) |  |  | | --- | |  | |
| Has lowercase letters | |  | | --- | | any(char.islower() for char in password) |  |  | | --- | |  | |
| Has digits (numbers) | |  | | --- | | any(char.isdigit() for char in password) |  |  | | --- | |  | |
| Has special characters | any(char in string.punctuation for char in password) |

Example

Weak Password outcome

A screenshot of a computer

Description automatically generated

Moderate Password

A screenshot of a computer

Description automatically generated

Strong Password

A screenshot of a computer

Description automatically generated

DICTIONARY ATTACK OVERVIEW

A dictionary attack attempts to guess a password by repeatedly using common passwords from a pre-compiled list. It's one of the simplest and most used password cracking methods.

In this dictionary attack test, I created a list of common password file, to explain how it works, you can also outsource this.

How it works

* Obtains a list of popular passwords from common\_passwords.txt.
* Ask the user for a password to test.
* Loop through each item on the list
* Removes spaces or newline character.
* Compare against the user’s input.
* If a match is found it prints success
* If there are no matches in the entire lists, it prints fails.

A black rectangular object with a white border

Description automatically generated

HASHED PASSWORD & DICTIONARY ATTACK

Below is an example of a hashed password.

A screenshot of a computer

Description automatically generated

Hashing is the process of converting data—such as text, numbers, or files—into a fixed-length string of letters and numbers, known as a hash value or digest, using a special algorithm called a hash function. Regardless of the input size, the output (hash) has a consistent, predetermined length. Hashing is widely used for data indexing, authentication, digital signatures, and in cryptographic applications to ensure data integrity and security.

Hash functions are intentionally designed to be one-way, meaning they are easy to compute in one direction (from input to hash) but extremely difficult to reverse (from hash back to input). Several reasons explain this:

Collision Resistance: Good hash function are programmed so that it is computationally impossible to discover two different inputs that generate the hash value. This improves security by preventing attackers from fabricating data using the same hash.

Brute Force Impracticality: The only theoretical way to reverse a hash is by brute force—trying every possible input until the hash matches. For strong hash functions, this would require an astronomical amount of time and computing resources, making it practically impossible.

Fixed Output Length: Hash functions output a fixed-length value, regardless of input size. This means many different inputs can produce the same hash (a property called “many-to-one”), making it impossible to determine which original input created a given hash.

One-wayness: The mathematical operations used in secure hash functions are complex and irreversible. Even knowing the algorithm, you cannot feasibly reconstruct the original input from the hash.

Example

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Algorithm | Hash Value(Example) | Length(Hex) |
| “hello” | SHA-256 | 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 | 64 |

In summary hashing transforms data into a unique, fixed-length value using a one-way function, making it secure and efficient for various applications. The design of hash functions—irreversibility, fixed output, and collision resistance—ensures that hashes cannot be feasibly reversed or used to reconstruct the original data.