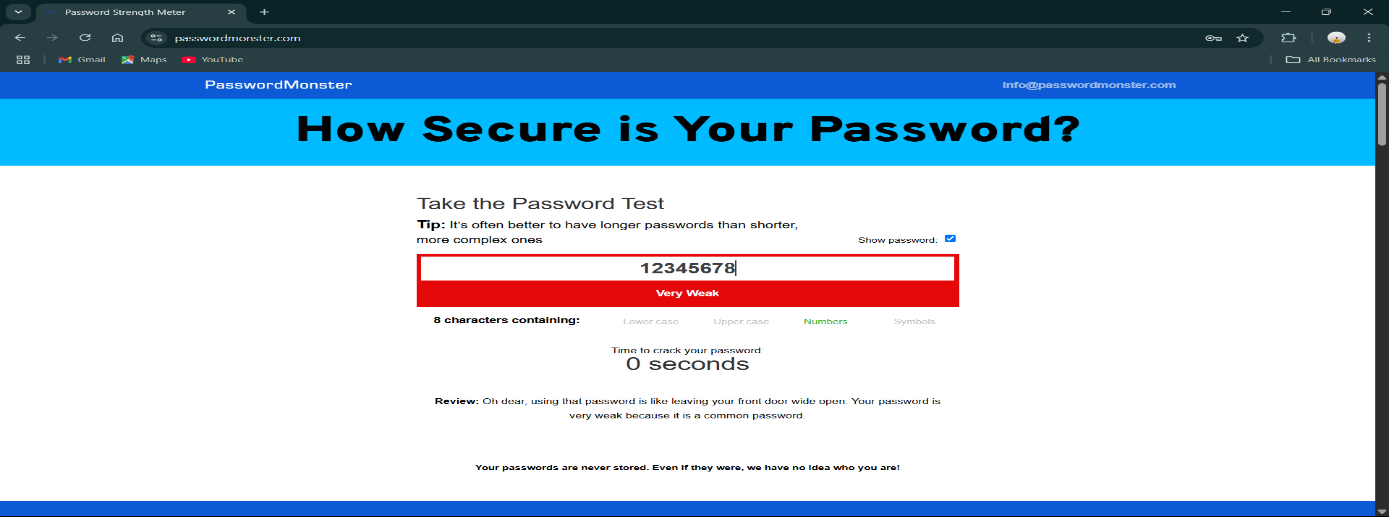
TASK – 6

## Understanding password security and best practices.

**Password Testing Results**

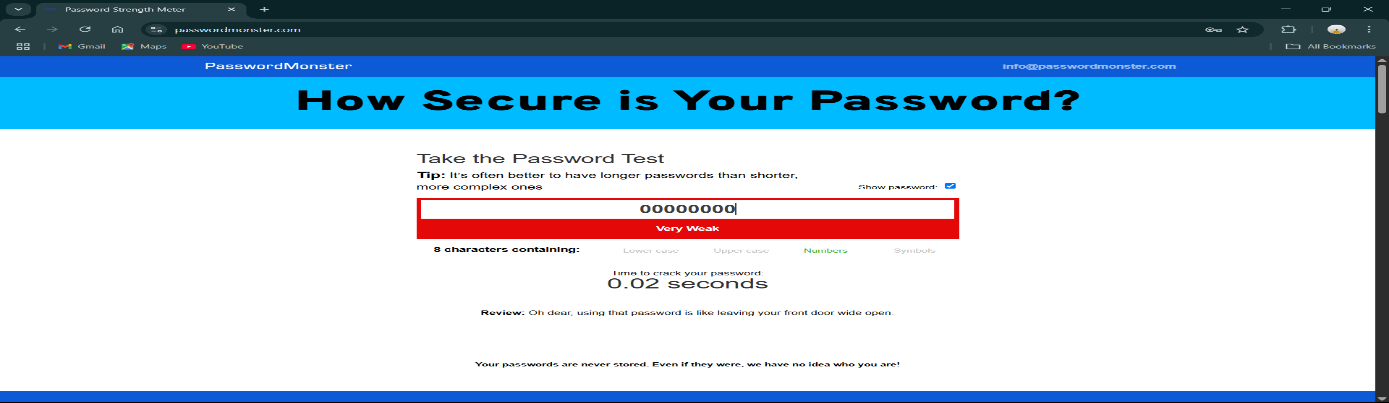
**Test 1: Simple Sequential Numbers**

****

**Password:** 12345678  
**Length:** 8 characters  
**Character Types:** Numbers only  
**Strength Rating:** Very Weak (Red)  
**Time to Crack:** 0 seconds  
**Tool Feedback:** "Oh dear, using that password is like leaving your front door wide open. Your password is very weak because it is a common password."

This password failed completely because it uses a predictable sequence that appears in common password lists. Attackers can crack this instantly.

**Test 2: Repeated Numbers**

****

**Password:** 00000000 **Length:** 8 characters **Character Types:** Numbers only **Strength Rating:** Very Weak (Red) **Time to Crack:** 0.02 seconds **Tool Feedback:** "Oh dear, using that password is like leaving your front door wide open."

Even though this has 8 characters, using the same digit repeatedly makes it extremely vulnerable. The slight increase in crack time (0.02 vs 0 seconds) shows minimal improvement over sequential numbers.

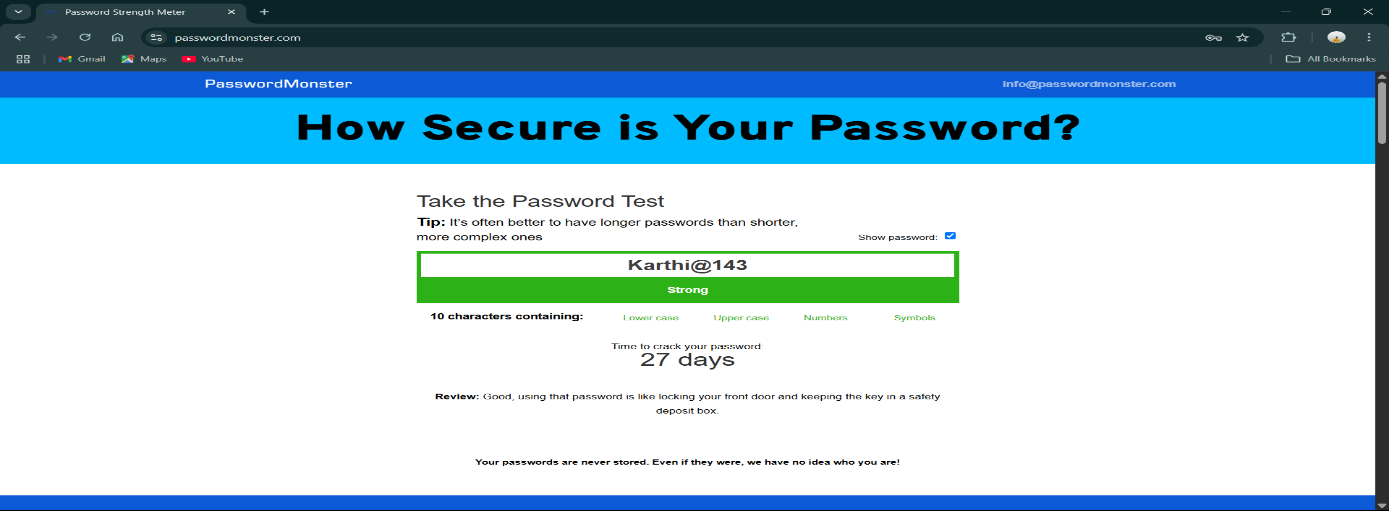
**Test 3: Name with Numbers and a Special character (Medium Complexity)**

****

**Password:** karthi@321  
**Length:** 10 characters  
**Character Types:** Lowercase letters, numbers, symbols  
**Strength Rating:** Medium (Yellow)  
**Time to Crack:** 28 hours  
**Tool Feedback:** "Hmm, using that password is like locking your front door, but leaving the key under the mat. Your password is of medium strength because it contains 2 dictionary words and a sequence of characters."

This password showed improvement by mixing character types, but using dictionary words (names) and predictable number sequences reduces its effectiveness.

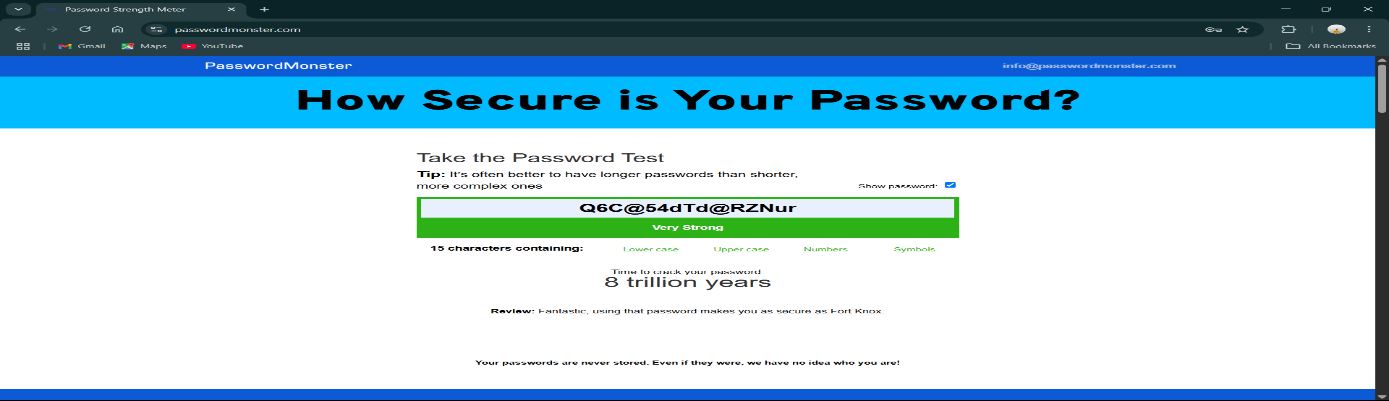
**Test 4: Name including Upper case , Numbers , Special character.**

****

**Password:** Karthi@143  
**Length:** 10 characters  
**Character Types:** Uppercase, lowercase, numbers, symbols  
**Strength Rating:** Strong (Green)  
**Time to Crack:** 27 days  
**Tool Feedback:** "Good, using that password is like locking your front door and keeping the key in a safety deposit box."

Adding uppercase letters significantly improved the strength. The combination of different character types increased crack time to weeks rather than hours.

**Test 5: Random Complex Password (Computer suggested).**

****

**Password:** Q6C@54dTd@RZNur  
**Length:** 15 characters  
**Character Types:** Uppercase, lowercase, numbers, symbols  
**Strength Rating:** Very Strong (Green)  
**Time to Crack:** 8 trillion years  
**Tool Feedback:** "Fantastic, using that password makes you as secure as Fort Knox."

This random, long password with mixed character types achieved the highest security rating. The dramatic increase in crack time shows how length and randomness create exponential security improvements.

**What Makes Passwords Weak:**

* Using common sequences (12345678, 00000000)
* Dictionary words or personal names
* Predictable patterns
* Single character types (only numbers, only letters)

**What Makes Passwords Strong:**

* Length (15+ characters is much better than 8-10)
* Mix of uppercase, lowercase, numbers, and symbols
* Random combinations without dictionary words
* Avoiding personal information

**Best Practices Learned**

1. **Length Matters Most:** The 15-character password was significantly stronger than 10-character ones
2. **Avoid Dictionary Words:** Personal names and common words make passwords vulnerable
3. **Use All Character Types:** Include uppercase, lowercase, numbers, and symbols
4. **Randomness is Key:** Avoid predictable patterns or sequences
5. **Don't Use Personal Information:** Names, birthdays, and familiar numbers reduce security

**Basic code for password checking in C++ :**

// C++ program to check if a given password is

// strong or not.

#include <bits/stdc++.h>

using namespace std;

void printStrongNess(string& input)

{

int n = input.length();

// Checking lower alphabet in string

bool hasLower = false, hasUpper = false;

bool hasDigit = false, specialChar = false;

string normalChars = "abcdefghijklmnopqrstu"

"vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 ";

for (int i = 0; i < n; i++) {

if (islower(input[i]))

hasLower = true;

if (isupper(input[i]))

hasUpper = true;

if (isdigit(input[i]))

hasDigit = true;

size\_t special = input.find\_first\_not\_of(normalChars);

if (special != string::npos)

specialChar = true;

}

// Strength of password

cout << "Strength of password:-";

if (hasLower && hasUpper && hasDigit &&

specialChar && (n >= 8))

cout << "Strong" << endl;

else if ((hasLower || hasUpper) &&

specialChar && (n >= 6))

cout << "Moderate" << endl;

else

cout << "Weak" << endl;

}

// Driver code

int main()

{

string input = "GeeksforGeeks!@12";

printStrongNess(input);

return 0;

}

**Common Password Attacks :**

**Brute Force Attacks:** Systematically trying every possible combination. Strong passwords with mixed characters make this extremely time-consuming.

**Dictionary Attacks:** Using lists of common passwords and words. This is why "karthi@321" was only medium strength - it contained recognizable words.

**Pattern-Based Attacks:** Exploiting predictable patterns like "12345678" or repeated characters like "00000000".

**Credential Stuffing Attacks:** Attackers use username/password combinations leaked from other breaches to try the same credentials on different websites, exploiting people's habit of reusing passwords across multiple platforms. This is why using the same password for multiple accounts is dangerous.

**Rainbow Table Attacks:** These attacks use precomputed tables of hash values to quickly reverse-engineer plaintext passwords from their encrypted forms.