Task 6 : Create a Strong Password and Evaluate Its Strength.

Step 1 :-

Create multiple passwords with varying the complexity.

Level	Example password	Description
Very Weak	1234567890	Only Numbers
Weak	admin123	Common words + digits
Moderate	admin123!	Word + Number + Symbol
Strong	Tr@v3l_Pradf3q4#	Mixed case + Symbols, Numbers
Very Strong	Hgs2*2q3LKNca	Randomly Generated

Step 2 :-

Testing the passwords that we have designed.

Install some libraires like libcrack2, cracklib-runtime, etc

```
┌──(root㉿kali)-[/home/kali]
└─# apt install libcrack2 cracklib-runtime
libcrack2 is already the newest version (2.9.6-5.2+b1).
libcrack2 set to manually installed.
cracklib-runtime is already the newest version (2.9.6-5.2+b1).
cracklib-runtime set to manually installed.
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 866
```

Let’s check the password strength.

```
┌──(root㉿kali)-[/home/kali]
└─# echo "Tr@v3l_P4ss#92" | cracklib-check
Tr@v3l_P4ss#92: OK
```

Step 3 :-

Let’s create the table to note all the feedback from the tools.

Level	Example password	Description	Feedback	Note
Very Weak	1234567890	Only numbers	Password can be cracked easily — common sequence, no entropy.	Never use numeric-only passwords. Change immediately; use a manager to generate a stronger password.
Weak	admin123	Common words + digits	It is too simple and would take about **~1 minute** to crack using common offline tools / dictionary + rules on consumer hardware.	Avoid dictionary words, predictable suffixes like 123. Use a longer passphrase or add randomness.
Moderate	admin123!	Word + Number + Symbol	Mixed characters help but still predictable. Estimated **~16 hours** to crack under typical offline brute-force/dictionary + rules assumptions.	Better than weak, but still uses a dictionary root. Use longer length and replace with random words or generated string.
Strong	Tr@v3l_Pradf3q4#	Mixed case + symbols + numbers + underscore	High complexity and length — resistant to casual cracking. Will take a very long time (orders of magnitude longer than moderate) with offline brute-force; effectively safe for most threats.	Good for important accounts if unique. Prefer a random or manager-generated password of equal length for maximal safety.
Very Strong	Hgs2*2q3LKNca	Randomly generated (example)	Very high entropy and randomness — practically infeasible to crack with current hardware for offline brute-force attacks.	Store in a password manager. Use 16+ characters or a 4+ word passphrase for memorability + strength.

Step 4 :-

Identify the best practice to create strong passwords.

- Use at least 12–16 characters
- Mix uppercase, lowercase, numbers, and symbols
- Avoid dictionary words
- Don’t reuse passwords
- Use a password manager to generate/store them

Step 5 :-

Learnings during the password evaluation.

- Longer passwords exponentially increase brute-force time.
- Substituting letters with symbols (e.g., @ for a) helps, but isn’t enough alone.
- Randomly generated passwords are far stronger than human-chosen ones.
- Use a passphrase (multiple random words) for memorability and strength.

Step 6 :-

Common Password’s Attack.

a. Brute-force attack:
Tries every possible combination — longer and more complex passwords take exponentially more time.

b. Dictionary attack:
Uses common words and patterns. Avoid any dictionary-based passwords.

c. Phishing / credential stuffing:
Attackers steal known credentials — use unique passwords per site.

Step 7 :-

Summarized the Complexity Affects Security.

Factor	Effect’s on Security
Length	Increases brute-force time dramatically
Character variety	Reduces predictability
Randomness	Prevents dictionary attacks
Uniqueness	Stops reuse attacks