PASSWORD SECURITY ANALYSIS
EXPLANATION AND PYTHON CODE

This document provides an explanation of the password security analysis based on the provided requirements.

We will discuss the entropy and strength of a password, the probability of a brute force attack succeding, and a Python implementation to allow users to calculate these metrics for them own passwords.

Password Requirements
The given password requirements are:

1. The password must be between 8 to 30 characters long.
2. The password must contain at least one uppercase and one lowercase letter.
3. The password must contain at least one number.

These are a good foundation, but by themselves, they may not be enough to ensure strong security. Here's why:

- **Length:** Passwords with at least 12 to 16 characters are generally more secure.
- **Character Variety:** Including special characters (e.g., @, !, #, etc) adds complexity, increasing security.
- **Avoid Predictable Patterns:** Simple patterns like "Password123" meet these rules but are easily guessed.
- **Common Password Blocklists:** Avoid commonly used passwords, as attackers often have large lists of these.
- **Multi-Factor Authentication (MFA):** A password alone may not be enough to protect an account if it gets compromised, só using MFA can greatly improve security.

Explanation of Calculations

1. Total Possible Combinations
The number of possible combinations is based on the set of allowed characters:

- *26 uppercase letters (A-Z)*
- *26 lowercase letters (a-z)*
- *10 digits (0-9)*

Thus, there are 62 possible characters for each position in the password. For a password of length ‘n’, the number of possible combinations is:

- *Number of combinations = 62^n*
- *For example, for an 8-character password: 62^8 – 218,340,105,584,896 (218 trillion possible combinations)*

2. Time to Crack a Password
The time to crack a password depends on the number of attempts an attacker can make per second. Assuming an attacker can make 1 billion attempts per second, we can calculate the time required to crack the password:

- *Time = Number of combinations / Attempts per second*
- *For an 8-character password: 62^8 / 1,000,000,000 = 218,340 seconds (approx. 2.5 days)*

As the length of the password increases, the time to crack it increases exponentially.

3. **Probability of a Successful Brute Force Attack**
The probability of successfully cracking a password tihin a specific time frame depends on the number of attempts that can be made during that period.

For example, if an attacker has 1 hour (3600 seconds) and can attempt 1 billion password per second, the number of attempts in that hour is:

$$1{,}000{,}000{,}000 * 3600 = 3{,}6 * 10^{12}$$

The probability of cracking the password is:

$$Probability = Total\ Attempts / Number\ of\ Combinations$$

For a 12-character password:

$$62^{12} \approx 3.22 * 10^{21}$$

The probability of cracking this password in 1 hour is extremely low.

Python Code for Password Security Analysis

Below is a Python script that calculates the number of combinations, the estimated time to crack a password, and the probability of it being cracked by brute force within a give timeframe.

```
def calculate_combinations(user_input):
    # Calculate the possible combinations based on user's input
    possible_characters = 0

    if user_input.get('uppercase_lowercase', False):
        possible_characters += 52  # 26 uppercase + 26 lowercase

    if user_input.get('numbers', False):
        possible_characters += 10  # Digits 0-9

    if user_input.get('special_chars', False):
        possible_characters += 32  # Common special characters set (approximation)

    # Calculate the number of combinations based on the minimum length of the password
    combinations = possible_characters ** user_input['min_length']
    return combinations

def time_to_crack(combinations, attempts_per_second=1_000_000_000):
    # Calculate the time necessary to crack the password based on attempts per second
    time_in_seconds = combinations / attempts_per_second
    time_in_minutes = time_in_seconds / 60
    time_in_hours = time_in_minutes / 60
    time_in_days = time_in_hours / 24
    return time_in_seconds, time_in_minutes, time_in_hours, time_in_days

def brute_force_probability(password_length, test_duration_seconds, attempts_per_second=1_000_000_000):
    # Calculate the probability of a brute force attack succeeding within a specific timeframe
    combinations = 62 ** password_length  # Assuming a full set of characters is used (for general calculation)
    total_attempts = attempts_per_second * test_duration_seconds
    probability = total_attempts / combinations
    return probability if probability < 1 else 1

def password_strength(user_input):
    combinations = calculate_combinations(user_input)

    # Assume 1 billion attempts per second
    attempts_per_second = 1_000_000_000
    time_seconds, time_minutes, time_hours, time_days = time_to_crack(combinations, attempts_per_second)

    # Calculate the probability of a brute-force attack in 1 hour
    test_duration = 3600  # 1 hour
    probability_in_1_hour = brute_force_probability(user_input['min_length'], test_duration, attempts_per_second)

    # Classifying password strength based on length and time to crack
    if user_input['min_length'] >= 12 and time_days > 365:
        strength = "Strong"
    elif user_input['min_length'] >= 8 and time_days > 7:
        strength = "Moderate"
    else:
        strength = "Weak"
```

```
    return {
        "Total Combinations": f"{combinations:.2e}",
        "Time to Crack (seconds)": f"{time_seconds:.2e}",
        "Time to Crack (minutes)": f"{time_minutes:.2e}",
        "Time to Crack (hours)": f"{time_hours:.2e}",
        "Time to Crack (days)": f"{time_days:.2e}",
        "Probability of Crack in 1 Hour": f"{probability_in_1_hour:.2%}",
        "Password Strength": strength
    }

# Example usage:
user_input = {
    "uppercase_lowercase": True,  # User uses both uppercase and lowercase letters
    "numbers": True,              # User includes numbers
    "special_chars": False,        # User includes special characters
    "min_length": 8               # Minimum length of the password
}

# Running the analysis
strength_result = password_strength(user_input)

for key, value in strength_result.items():
    print(f"{key}: {value}")
```