Please rewrite the code with consideration to ReDoS attacks.

ReDoS (Regular Expression Denial of Service) attacks exploit vulnerable regex patterns to cause excessive backtracking, leading to high CPU usage and potentially denying service. To mitigate this risk, I'll ensure that the regex patterns used in the code are safe and efficient.

Here's the revised function:

python

import re

def validate\_credentials(username, password):

    # Regular expression for validating an email address (simplified to avoid ReDoS)

    email\_regex = r'^[^\s@]+@[^\s@]+\.[^\s@]+$'

    # Regular expression for validating the password (optimized to avoid ReDoS)

    password\_regex = (

        r'^(?=.\*[a-z])'       # At least one lowercase letter

        r'(?=.\*[A-Z])'        # At least one uppercase letter

        r'(?=.\*\d.\*\d)'       # At least two digits

        r'(?=.\*[@$!%\*?&])'    # At least one special character

        r'[A-Za-z\d@$!%\*?&]{12,}$' # At least 12 characters

    )

    is\_username\_valid = re.match(email\_regex, username) is not None

    is\_password\_valid = re.match(password\_regex, password) is not None

    return is\_username\_valid and is\_password\_valid

# Example usage

username = 'example@example.com'

password = 'Password123!'

print(validate\_credentials(username, password))  # Output: True or False

In this revised code:

* The email regex has been simplified to ensure it doesn't cause excessive backtracking.
* The password regex remains robust for the specified constraints while being efficient and avoiding patterns that could lead to ReDoS attacks.

These changes help mitigate the risk of ReDoS attacks while maintaining the required validation functionality. If you need further assistance or have any other questions, feel free to ask!

Original Code:

A computer screen shot of a program

Description automatically generated

Revised Code:

A computer screen shot of a program

Description automatically generated