SecCode Analyzer

Analysis Results

Analysis of Security Issues

1. SQL Injection Vulnerability

- Severity: Critical

- Vulnerability: The code directly interpolates user input (username and *****) into SQL queries using f-strings, making it susceptible to SQL injection attacks.

- Impact: Attackers can manipulate the input to execute arbitrary SQL commands, potentially leading to data leakage, data manipulation, or database compromise.

2. Plain Text Password Storage:

- Severity: High

- Vulnerability: The code stores user passwords in plain text in the database, which is a significant security risk. Storing passwords in plain text exposes them to unauthorized access in case of a data breach.

- Impact: User privacy and security are compromised, leading to potential account takeovers and unauthorized access.

Improvements

1. Parameterized Queries to Prevent SQL Injection

- Use parameterized queries to prevent SQL injection attacks by separating SQL code from user input.

- Example:

```python

cursor.execute("INSERT INTO users (username, *****) VALUES (?, ?)", (username, password))

```

```python

cursor.execute("SELECT * FROM users WHERE username = ? AND ***** = ?", (username, password))

```

2. Secure Password Storage

- Hash user passwords before storing them in the database using a strong hashing algorithm like bcrypt.

- Example:

```python

import bcrypt

## def add_user(username, password)

hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

cursor.execute("INSERT INTO users (username, *****) VALUES (?, ?)", (username, hashed_password))

```

3. Secure Password Authentication

- When authenticating users, hash the input password and compare it with the hashed password stored in the database.

- Example:

```python

def authenticate(username, password):

cursor.execute("SELECT ***** FROM users WHERE username = ?", (username,))

hashed_password = cursor.fetchone()

if hashed_password and bcrypt.checkpw(password.encode('utf-8'), hashed_password[0]):

print("Autenticado com sucesso!")

else:
```

```
print("Falha na autenticação!")
```

By implementing these improvements, you can mitigate the critical SQL injection vulnerability and enhance the security of password storage and authentication in the code.