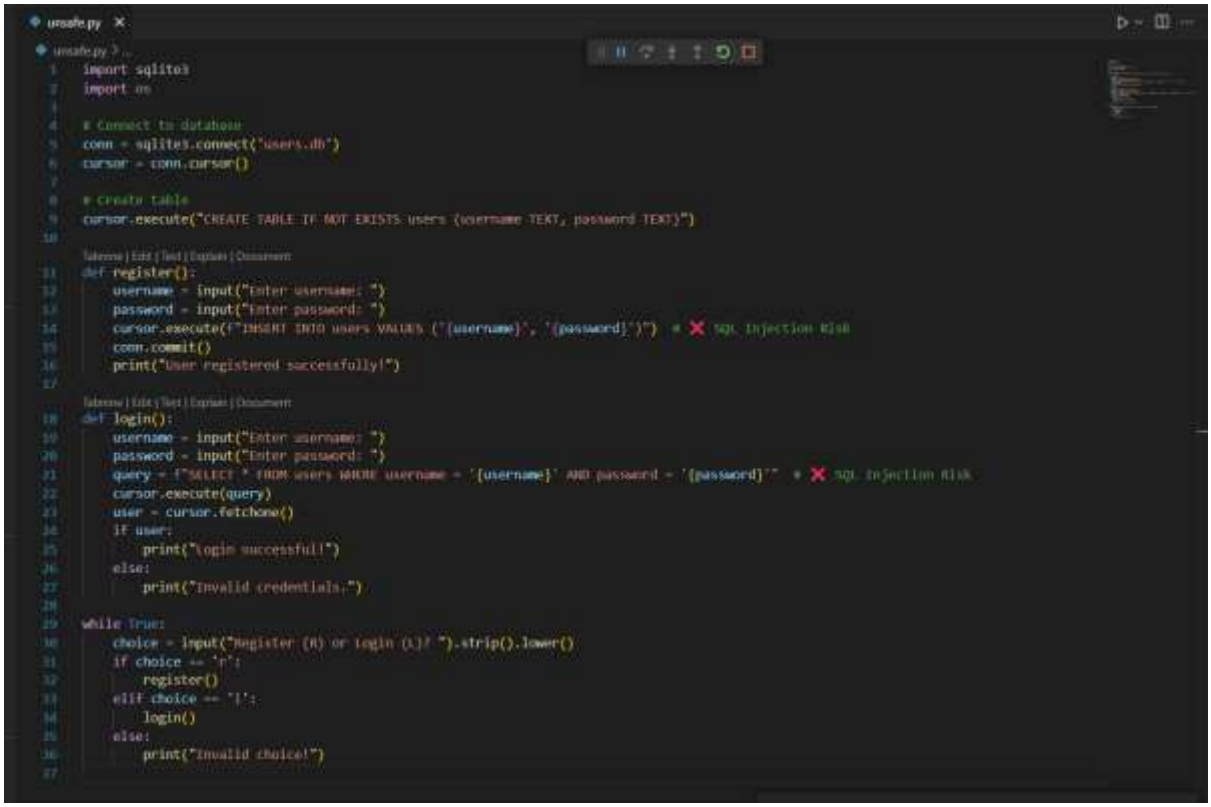
Task 3: Secure Coding Review (Python) – Step-by-Step Implementation

This guide walks through the process of secure coding review in Python. We will:

- Write a sample vulnerable Python script
- Analyze it for security flaws
- Fix the vulnerabilities using secure coding practices
- Use automated tools for security checks

Step 1: Create a Sample Vulnerable Python Script

Let's assume we have a login system where users can register and log in.



Step 2: Identify Security Vulnerabilities

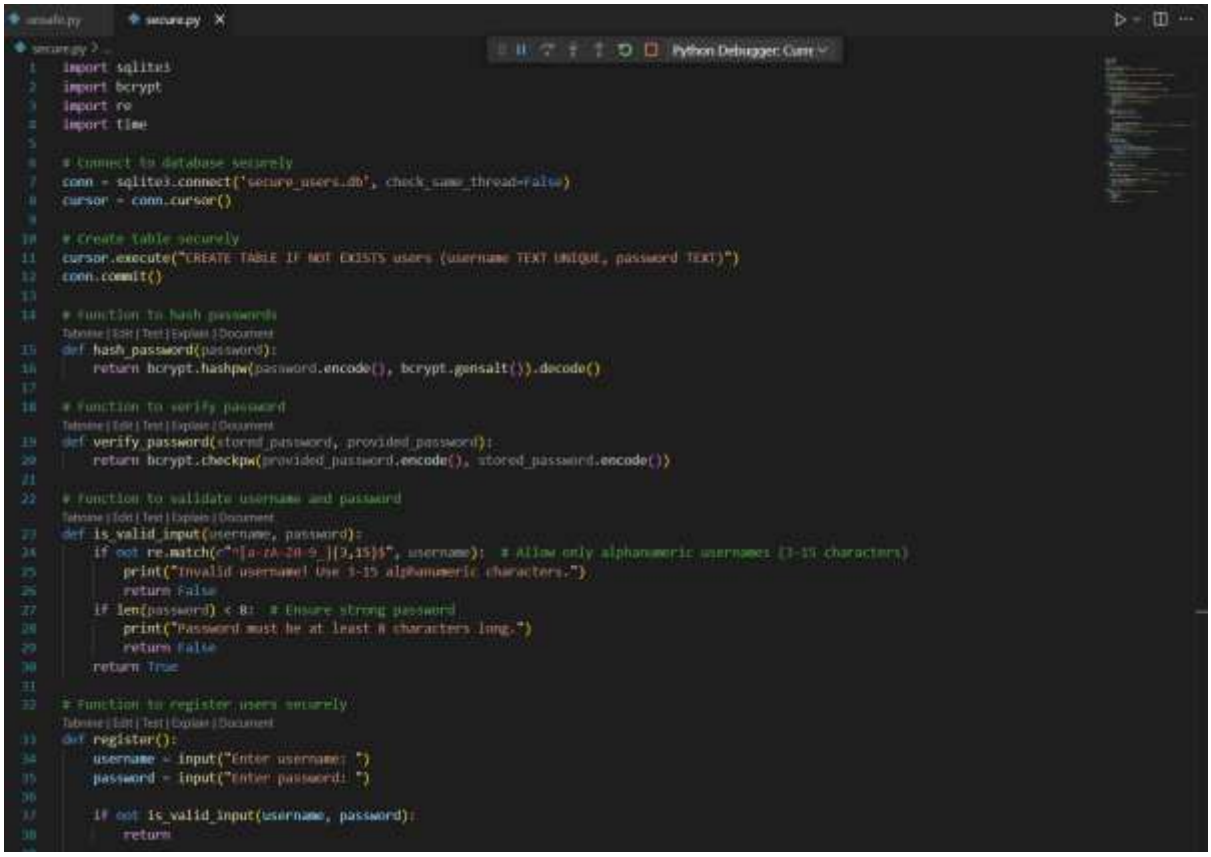
This script has multiple security flaws:

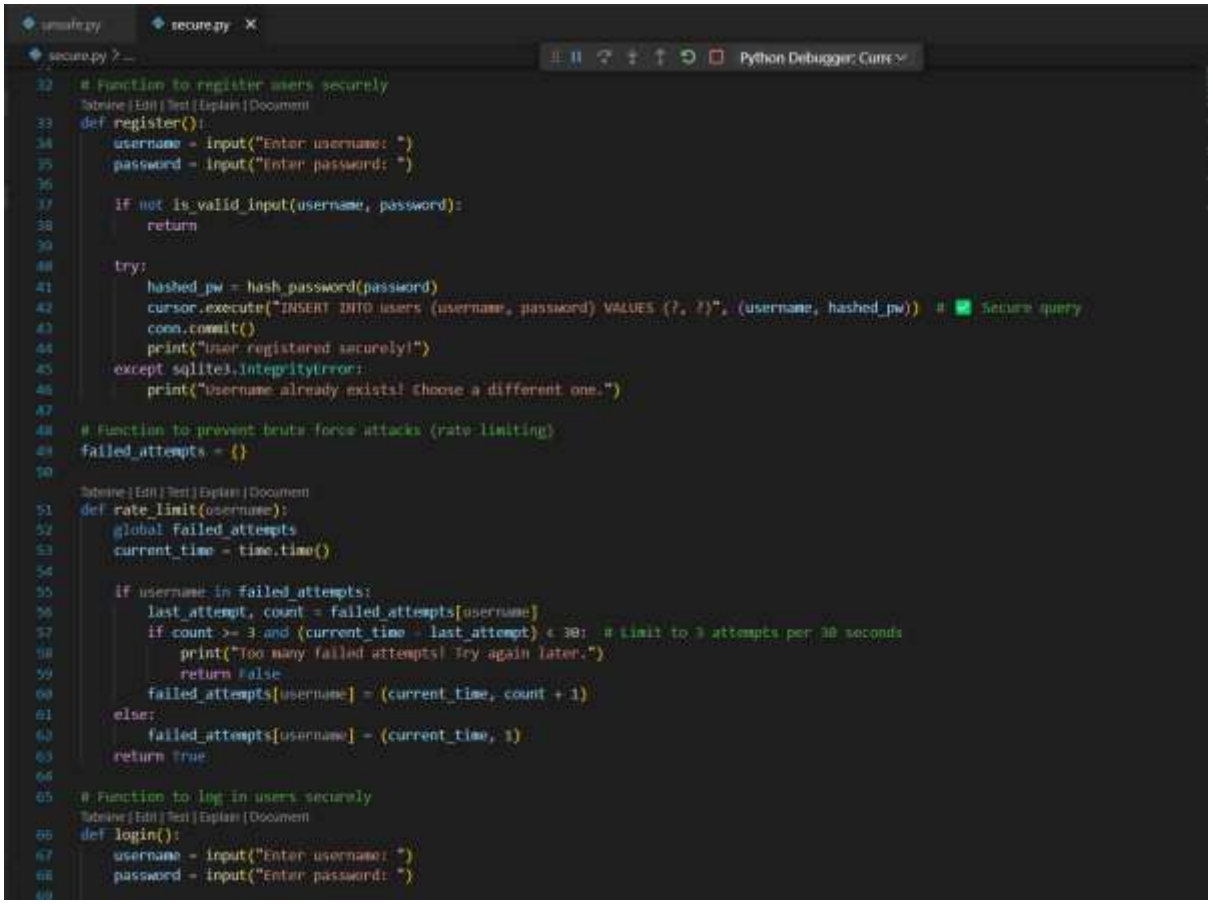
- **SQL Injection** – Uses string formatting in SQL queries, allowing attackers to inject malicious SQL commands.
- **Plaintext Password Storage** – Stores passwords directly without hashing.
- **Lack of Input Validation** – No checks for invalid usernames or weak passwords.
- **No Rate Limiting** – Attackers can attempt multiple logins (brute force attack).

Step 3: Fix the Security Issues

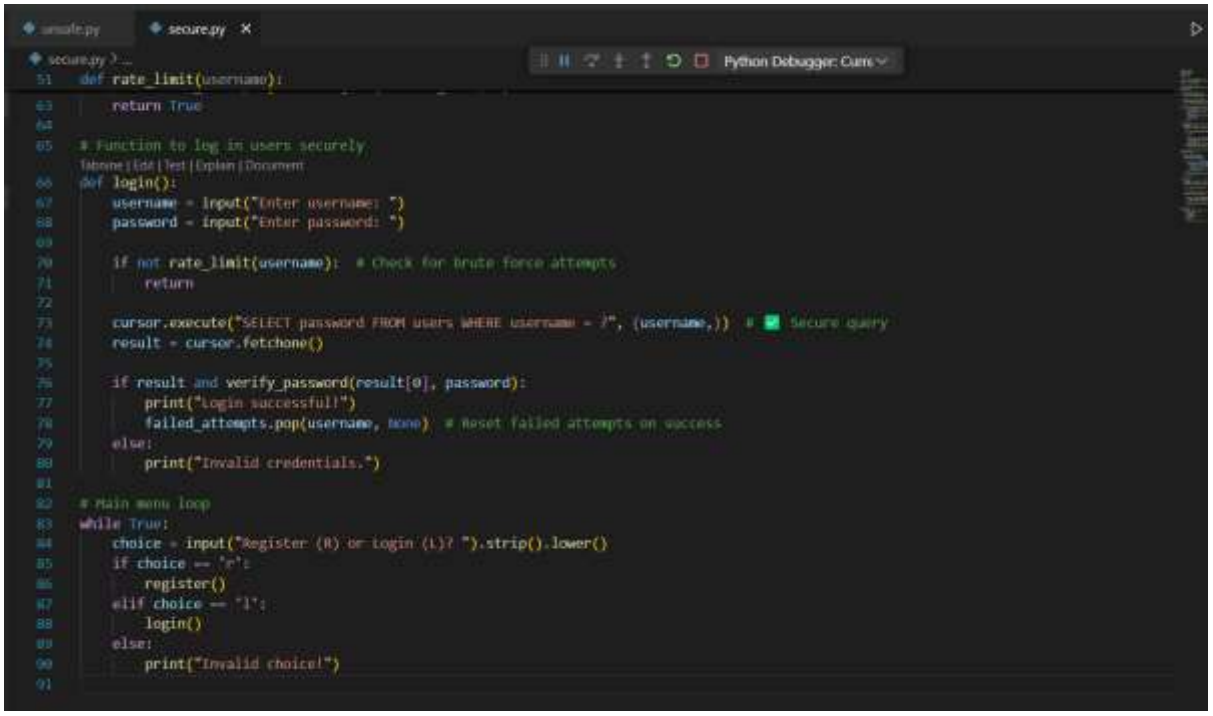
Now, let's secure the script by:

- Using parameterized queries to prevent SQL injection
- Hashing passwords using bcrypt
- Validating inputs
- Adding rate limiting





```python
# Function to register users securely
def register():
    username = input("Enter username: ")
    password = input("Enter password: ")

    if not is_valid_input(username, password):
        return

    try:
        hashed_pw = hash_password(password)
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_pw))  # ✅ Secure query
        conn.commit()
        print("User registered securely!")
    except sqlite3.IntegrityError:
        print("Username already exists! Choose a different one.")

# Function to prevent brute force attacks (rate limiting)
failed_attempts = {}

def rate_limit(username):
    global failed_attempts
    current_time = time.time()

    if username in failed_attempts:
        last_attempt, count = failed_attempts[username]
        if count >= 3 and (current_time - last_attempt) < 30:  # Limit to 3 attempts per 30 seconds
            print("Too many failed attempts! Try again later.")
            return False
        failed_attempts[username] = (current_time, count + 1)
    else:
        failed_attempts[username] = (current_time, 1)
    return True

# Function to log in users securely
def login():
    username = input("Enter username: ")
    password = input("Enter password: ")
```



```python
def rate_limit(username):
    return True

# Function to log in users securely
def login():
    username = input("Enter username: ")
    password = input("Enter password: ")

    if not rate_limit(username):  # Check for brute force attempts
        return

    cursor.execute("SELECT password FROM users WHERE username = ?", (username,))  # ✅ Secure query
    result = cursor.fetchone()

    if result and verify_password(result[0], password):
        print("Login successful!")
        failed_attempts.pop(username, None)  # Reset failed attempts on success
    else:
        print("Invalid credentials.")

# Main menu loop
while True:
    choice = input("Register (R) or Login (L)? ").strip().lower()
    if choice == 'r':
        register()
    elif choice == 'l':
        login()
    else:
        print("Invalid choice!")
```

```
PS C:\Users\tanma\Desktop\Task3> python secure.py
Register (R) or Login (L)? R
Enter username: Tany_Karle
Enter password: 123456789
User registered securely!
Register (R) or Login (L)? L
Enter username: Tany_Karle
Enter password: 123456789
Login successful!
```

Step 4: Perform Static Code Analysis

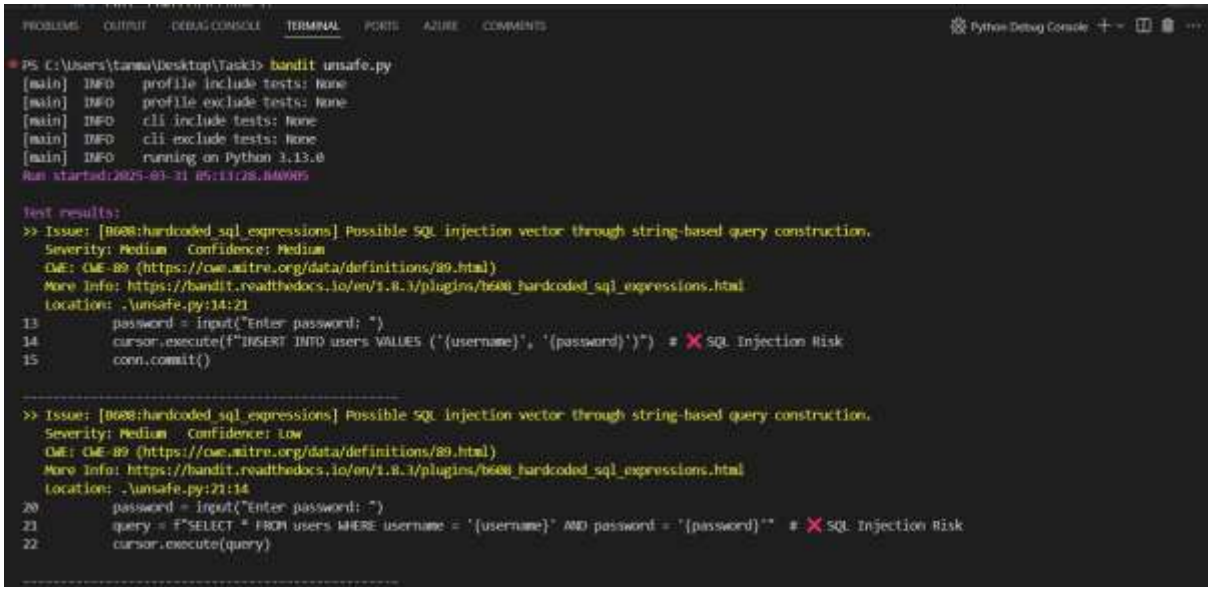
Use Bandit (Python security analyzer) to check for vulnerabilities.

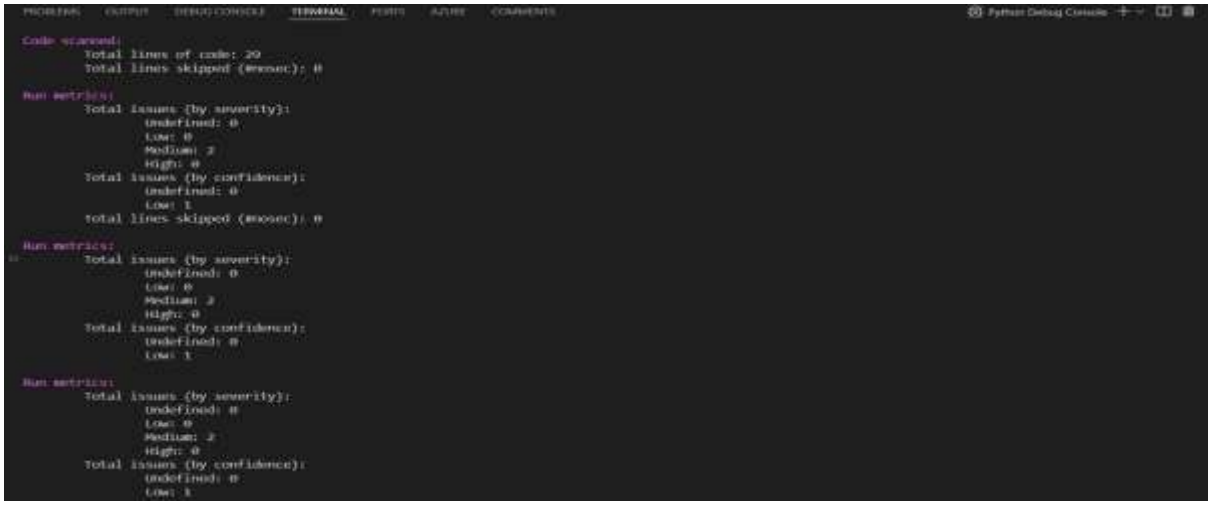
pip install bandit

Run Bandit on Secure Code

bandit secure.py

A. For unsafe.py





```
Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 1
                Low: 0
                Medium: 2
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 1
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 1
                Medium: 1
                High: 0
                Undefined: 0
                Low: 1
                Medium: 1
                High: 0
                Medium: 1
                High: 0
Files skipped (0):
```

B. For secure.py

```
PS C:\Users\tanma\Desktop\Task3> bandit secure.py
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.13.0
Run started:2025-03-31 05:13:16.427042

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 65
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
```

Step 5: Check Dependencies for Vulnerabilities

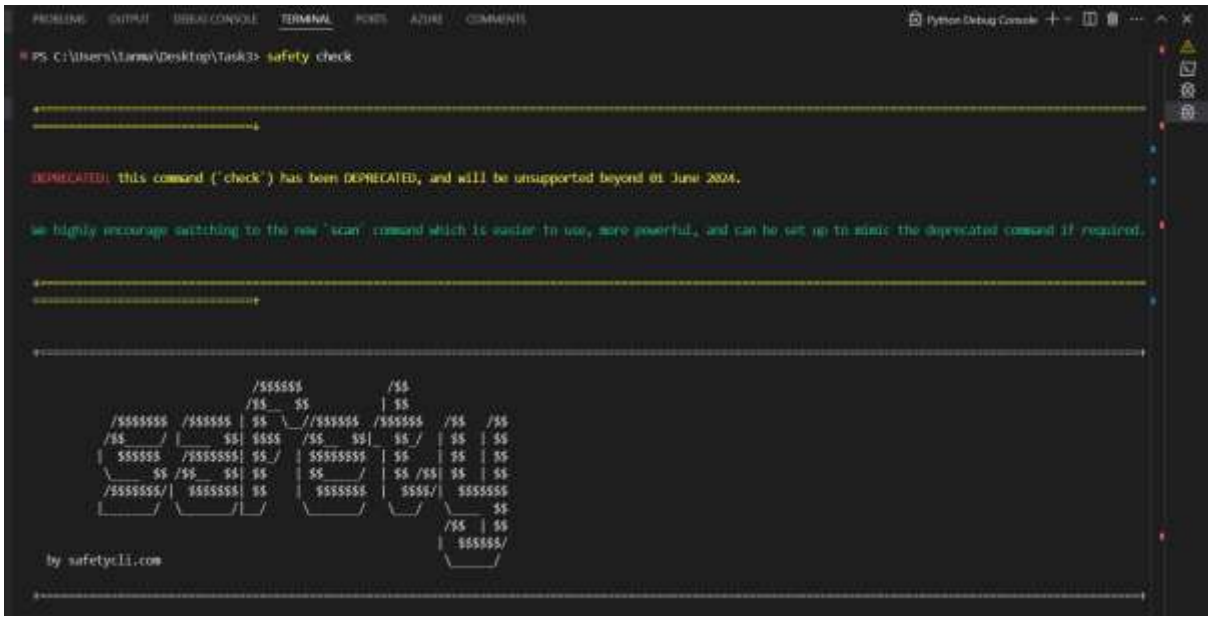
Use Safety to scan for insecure Python libraries.

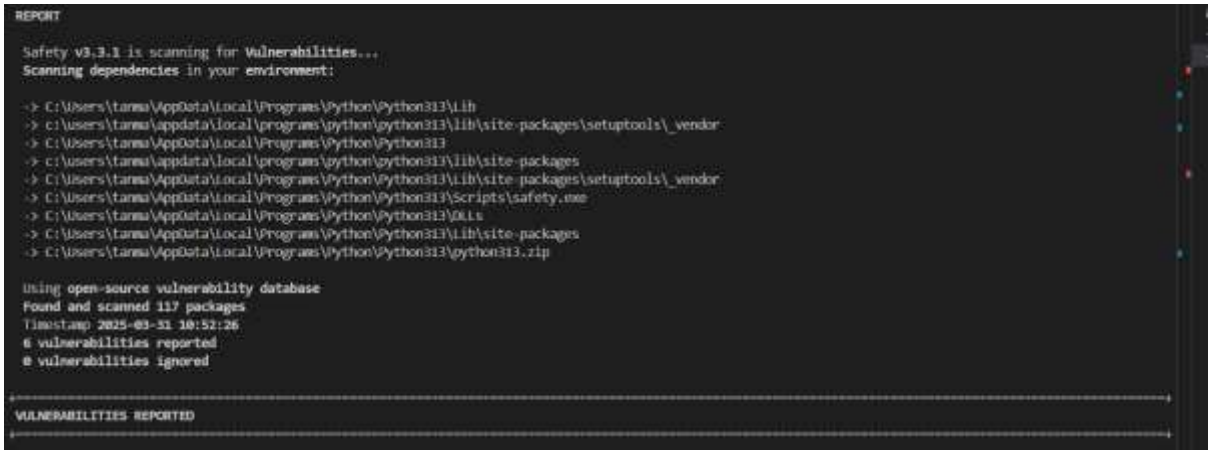
Install Safety

- pip install safety

Run Safety Scan

- safety check





```
51    def rate_limit(username):
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE  COMMENTS                    Python Debug Console

+-------------------------------------------------------------------------------+
 VULNERABILITIES REPORTED
+-------------------------------------------------------------------------------+

-> Vulnerability found in werkzeug version 3.0.5
   Vulnerability ID: 73969
   Affected spec: <3.0.6
   ADVISORY: Affected versions of Werkzeug are vulnerable to Path Traversal (CWE-22) on Windows systems running Python versions below 3.11. The
   safe_join() function failed to properly detect certain absolute paths on Windows, allowing attackers to potentially access files outside the...
   CVE-2024-49766
   For more information about this vulnerability, visit https://data.safetycli.com/v/73969/97c
   To ignore this vulnerability, use PyUp vulnerability id 73969 in safety's ignore command-line argument or add the ignore to your safety policy file.


-> Vulnerability found in werkzeug version 3.0.5
   Vulnerability ID: 73889
   Affected spec: <3.0.6
   ADVISORY: Affected versions of Werkzeug are potentially vulnerable to resource exhaustion when parsing file data in forms. Applications using
   `werkzeug.formparser.MultiPartParser` to parse `multipart/form-data` requests (e.g. all flask applications) are vulnerable to a relatively simple...
   CVE-2024-49767
   For more information about this vulnerability, visit https://data.safetycli.com/v/73889/97c
   To ignore this vulnerability, use PyUp vulnerability id 73889 in safety's ignore command-line argument or add the ignore to your safety policy file.


-> Vulnerability found in jinja2 version 3.1.4
   Vulnerability ID: 74735
   Affected spec: <3.1.5
   ADVISORY: A vulnerability in the Jinja compiler allows an attacker who can control both the content and filename of a template to execute
   arbitrary Python code, bypassing Jinja's sandbox protections. To exploit this vulnerability, an attacker must have the ability to manipulate both...
   CVE-2024-56201
   For more information about this vulnerability, visit https://data.safetycli.com/v/74735/97c
   To ignore this vulnerability, use PyUp vulnerability id 74735 in safety's ignore command-line argument or add the ignore to your safety policy file.
```

```
-> Vulnerability found in django version 5.1.3
   Vulnerability ID: 74396
   Affected spec: >=5.1a1,<5.1.4
   ADVISORY: Django affected versions are vulnerable to a potential SQL injection in the HasKey(lhs, rhs) lookup on Oracle databases. The
   vulnerability arises when untrusted data is directly used as the lhs value in the django.db.models.fields.json.HasKey lookup. However, applications...
   CVE-2024-53908
   For more information about this vulnerability, visit https://data.safetycli.com/v/74396/97c
   To ignore this vulnerability, use PyUp vulnerability id 74396 in safety's ignore command-line argument or add the ignore to your safety policy file.


-> Vulnerability found in django version 5.1.3
   Vulnerability ID: 74395
   Affected spec: >=5.1a1,<5.1.4
   ADVISORY: Affected versions of Django are vulnerable to a potential denial-of-service (DoS) attack in the `django.utils.html.strip_tags()`
   method. The vulnerability occurs when the `strip_tags()` method or the `striptags` template filter processes inputs containing large sequences of...
   CVE-2024-53907
   For more information about this vulnerability, visit https://data.safetycli.com/v/74395/97c
   To ignore this vulnerability, use PyUp vulnerability id 74395 in safety's ignore command-line argument or add the ignore to your safety policy file.


-> Vulnerability found in django version 5.1.3
   Vulnerability ID: 74985
   Affected spec: >=5.1.0,<5.1.5
   ADVISORY: Affected versions of Django are vulnerable to a potential denial-of-service attack due to improper IPv6 validation. The lack of upper
   limit enforcement for input strings in clean_ipv6_address, is_valid_ipv6_address, and the django.forms.GenericIPAddressField form field allowed...
   CVE-2024-56374
   For more information about this vulnerability, visit https://data.safetycli.com/v/74985/97c
   To ignore this vulnerability, use PyUp vulnerability id 74985 in safety's ignore command-line argument or add the ignore to your safety policy file.
```

```
+-------------------------------------------------------------------------------+
  REMEDIATIONS

  6 vulnerabilities were reported in 3 packages. For detailed remediation & fix recommendations, upgrade to a commercial license.

+-------------------------------------------------------------------------------+

 Scan was completed. 6 vulnerabilities were reported.

+-------------------------------------------------------------------------------+


+-------------------------------------------------------------------------------+
+-------------------------------+


DEPRECATED: this command (`check`) has been DEPRECATED, and will be unsupported beyond 01 June 2024.

We highly encourage switching to the new `scan` command which is easier to use, more powerful, and can be set up to mimic the deprecated command if required.

+-------------------------------------------------------------------------------+
+-------------------------------+
```

Summary of Security Improvements

Security Issue	Vulnerable Code ❌	Secure Code ✅
SQL Injection	Uses f"SELECT * FROM users WHERE username='{username}'"	Uses cursor.execute("SELECT * FROM users WHERE username=?", (username,))
Password Storage	Stores plaintext passwords	Uses bcrypt to hash passwords
Input Validation	No validation for usernames/passwords	Uses regex for username validation and enforces strong passwords
Brute Force Protection	No limit on login attempts	Implements rate limiting (3 attempts per 30 sec)