

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.

```
❶ unsafe.py x
❷ unsafe.py ...
❸
❹ import sqlite3
❺ import os
❻
❼ # Connect to database
➋ conn = sqlite3.connect('users.db')
⌽ cursor = conn.cursor()
⌾
⌿ # Create table
⌽ cursor.execute("CREATE TABLE IF NOT EXISTS users (username TEXT, password TEXT)")
⌾
⌿
⌿ # Allow edit / insert / delete / Document
⌽ def register():
⌽     username = input("Enter username: ")
⌽     password = input("Enter password: ")
⌽     cursor.execute("INSERT INTO users VALUES ('{username}', '{password}')") # ✖ SQL injection risk
⌽     conn.commit()
⌽     print("User registered successfully!")
⌾
⌿
⌿ # Allow edit / insert / delete / Document
⌽ def login():
⌽     username = input("Enter username: ")
⌽     password = input("Enter password: ")
⌽     query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'" # ✖ SQL injection risk
⌽     cursor.execute(query)
⌽     user = cursor.fetchone()
⌽     if user:
⌽         print("Login successful!")
⌽     else:
⌽         print("Invalid credentials.")
⌾
⌿
⌿ while True:
⌽     choice = input("Register (R) or login (L)? ").strip().lower()
⌽     if choice == 'r':
⌽         register()
⌽     elif choice == 'l':
⌽         login()
⌽     else:
⌽         print("Invalid choice!")
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PREFERENCES AZURE COMMENTS Python Debug Console + - X
```

```
PS C:\Users\Yaroslav\Desktop\task3> & "C:\Users\Yaroslav\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Yaroslav\vscode\extensions\ms-python.python\debug\2025-4-3-Main.py" >>> "C:\Users\Yaroslav\Desktop\task4\ans4.py"
Register (R) or login (L)? R
Enter username: Fawzy_Karla
Enter password: 123456789
user registered successfully!
Register (R) or Login (L)? L
Enter username: Fawzy_Karla
Enter password: 123456789
Login successful!
Register (R) or login (L)? L
```

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

```
securepy.pyx
# securepy.pyx

1 import sqlite3
2 import bcrypt
3 import re
4 import time
5
6 # Connect to database securely
7 conn = sqlite3.connect('secure_users.db', check_same_thread=False)
8 cursor = conn.cursor()
9
10
11 # Create table securely
12 cursor.execute("CREATE TABLE IF NOT EXISTS users (username TEXT UNIQUE, password TEXT)")
13 conn.commit()
14
15
16 # Function to hash passwords
17 def hash_password(password):
18     return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode()
19
20
21 # Function to verify password
22 def verify_password(stored_password, provided_password):
23     return bcrypt.checkpw(provided_password.encode(), stored_password.encode())
24
25
26 # Function to validate username and password
27 def is_valid_input(username, password):
28     if not re.match("[a-zA-Z0-9]{3,15}$", username): # Allow only alphanumeric usernames (3-15 characters)
29         print("Invalid username! Use 3-15 alphanumeric characters!")
30         return False
31     if len(password) < 8: # Ensure strong password
32         print("Password must be at least 8 characters long!")
33         return False
34     return True
35
36
37 # Function to register users securely
38 def register():
39     username = input("Enter username: ")
40     password = input("Enter password: ")
41
42     if not is_valid_input(username, password):
43         return
```

```
❸ unsafe.py ❹ secure.py ✘
❺ secure.py >...
32 # Function to register users securely
33 Tabnine (Edit) Test Explain Document
34 def register():
35     username = input("Enter username: ")
36     password = input("Enter password: ")
37
38     if not is_valid_input(username, password):
39         return
40
41     try:
42         hashed_pw = hash_password(password)
43         cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_pw)) # Secure query
44         conn.commit()
45         print("User registered securely!")
46     except sqlite3.IntegrityError:
47         print("Username already exists! Choose a different one.")
48
49 # Function to prevent brute force attacks (rate limiting)
50 failed_attempts = {}
51
52 Tabnine (Edit) Test Explain Document
53 def rate_limit(username):
54     global failed_attempts
55     current_time = time.time()
56
57     if username in failed_attempts:
58         last_attempt, count = failed_attempts[username]
59         if count >= 3 and (current_time - last_attempt) < 30: # limit to 3 attempts per 30 seconds
60             print("Too many failed attempts! Try again later!")
61             return False
62         failed_attempts[username] = (current_time, count + 1)
63     else:
64         failed_attempts[username] = (current_time, 1)
65     return True
66
67 # Function to log in users securely
68 Tabnine (Edit) Test Explain Document
69 def login():
70     username = input("Enter username: ")
71     password = input("Enter password: ")
72
73     if not rate_limit(username): # Check for brute force attempts
74         return
75
76     cursor.execute("SELECT password FROM users WHERE username = ?", (username,)) # Secure query
77     result = cursor.fetchone()
78
79     if result and verify_password(result[0], password):
80         print("Login successful!")
81         failed_attempts.pop(username, None) # Reset failed attempts on success
82     else:
83         print("Invalid credentials!")
84
85 # Main menu loop
86 while True:
87     choice = input("Register (R) or login (L)? ").strip().lower()
88     if choice == 'r':
89         register()
90     elif choice == 'l':
91         login()
92     else:
93         print("Invalid choice!")
```

```
❸ unsafe.py ❹ secure.py ✘
❺ secure.py >...
51 def rate_limit(username):
52     return True
53
54 # Function to log in users securely
55 Tabnine (Edit) Test Explain Document
56 def login():
57     username = input("Enter username: ")
58     password = input("Enter password: ")
59
60     if not rate_limit(username): # Check for brute force attempts
61         return
62
63     cursor.execute("SELECT password FROM users WHERE username = ?", (username,)) # Secure query
64     result = cursor.fetchone()
65
66     if result and verify_password(result[0], password):
67         print("Login successful!")
68         failed_attempts.pop(username, None) # Reset failed attempts on success
69     else:
70         print("Invalid credentials!")
71
72 # Main menu loop
73 while True:
74     choice = input("Register (R) or login (L)? ").strip().lower()
75     if choice == 'r':
76         register()
77     elif choice == 'l':
78         login()
79     else:
80         print("Invalid choice!")
```

```
⑧ 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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE COMMENTS

PS C:\Users\tanu\Desktop\Task> bandit unsafe.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 05:13:28.849095

Test results:
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
  Severity: Medium  Confidence: Medium
  CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
  More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b608_hardcoded_sql_expressions.html
  Location: unsafe.py:14:21
13     password = input("Enter password: ")
14     cursor.execute(f"INSERT INTO users VALUES ('{username}', '{password}')") # ✘ SQL Injection Risk
15     conn.commit()

>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
  Severity: Medium  Confidence: Low
  CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
  More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b608_hardcoded_sql_expressions.html
  Location: unsafe.py:21:14
20     password = input("Enter password: ")
21     query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'" # ✘ SQL injection Risk
22     cursor.execute(query)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PREFERENCES ACTIVITY COMMENTS

Code statistics:
  Total lines of code: 29
  Total lines skipped (missed): 0

Run metrics:
  Total issues (by severity):
    undefined: 0
    Low: 0
    Medium: 2
    High: 0
  Total issues (by confidence):
    undefined: 0
    Low: 1
  Total lines skipped (missed): 0

Run metrics:
  Total issues (by severity):
    undefined: 0
    Low: 0
    Medium: 2
    High: 0
  Total issues (by confidence):
    undefined: 0
    Low: 3

Run metrics:
  Total issues (by severity):
    undefined: 0
    Low: 0
    Medium: 2
    High: 0
  Total issues (by confidence):
    undefined: 0
```

```
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\tanna\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

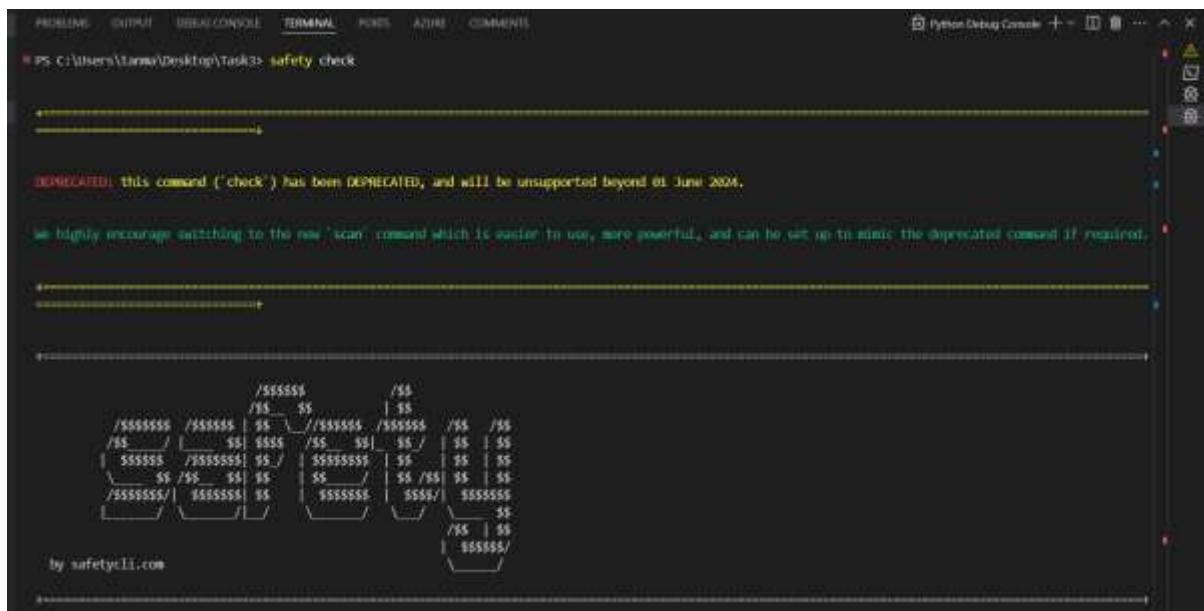
Use Safety to scan for insecure Python libraries.

Install Safety

- pip install safety

Run Safety Scan

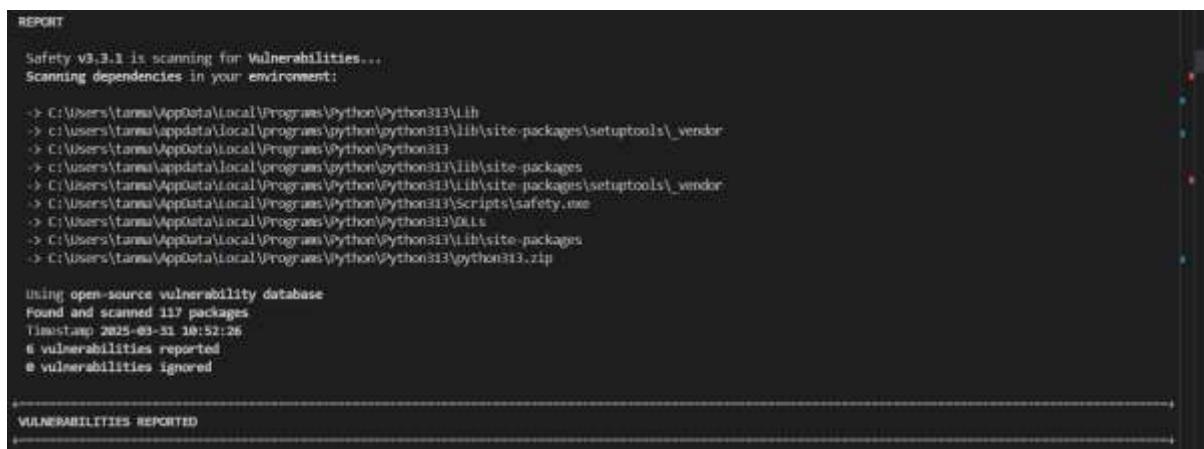
- safety check



```
PROBLEMS OUTPUT USER CONSOLE TERMINAL FOCUS AZURE COMMENTS
PS C:\users\tarun\Desktop\Tasks> safety check

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 alias the deprecated command if required.

SASSY
by safetycli.com
```



```
REPORT
Safety v3.3.1 is scanning for vulnerabilities...
Scanning dependencies in your environment:
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\lib
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\lib\site-packages\setuptools\_vendor
> C:\Users\tarun\AppData\Local\Programs\Python\Python33
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\lib\site-packages
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\lib\site-packages\setuptools\_vendor
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\Scripts\safety.exe
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\XML
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\lib\site-packages
> C:\Users\tarun\AppData\Local\Programs\Python\Python33\python33.zip

Using open-source vulnerability database
Found and scanned 117 packages
Timestamp: 2025-03-31 10:52:26
6 vulnerabilities reported
8 vulnerabilities ignored

VULNERABILITIES REPORTED
```

```
SI deivate [username]: ~
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FORMS AZURE COMMENTS

VULNERABILITIES REPORTED

→ Vulnerability found in werkzeug version 2.0.5
Vulnerability ID: 73969
Affected spec: <3.0.6
ADVISORY: Affected versions of Werkzeug are vulnerable to Path Traversal (OW-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-2024-09766
For more information about this vulnerability, visit <https://data.safetycli.com/v/73969/97c>
To ignore this vulnerability, use PyPI vulnerability id 73969 in safety's ignore command-line argument or add the ignore to your safety policy file.

→ Vulnerability found in werkzeug version 2.0.5
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-2024-09767
For more information about this vulnerability, visit <https://data.safetycli.com/v/73889/97c>
To ignore this vulnerability, use PyPI 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-2024-56281
For more information about this vulnerability, visit <https://data.safetycli.com/v/74735/97c>
To ignore this vulnerability, use PyPI vulnerability id 74735 in safety's ignore command-line argument or add the ignore to your safety policy file.

→ Vulnerability found in django version 5.1.2
Vulnerability ID: 74396
Affected spec: >=5.1.0,<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-2024-53988
For more information about this vulnerability, visit <https://data.safetycli.com/v/74396/97c>
To ignore this vulnerability, use PyPI vulnerability id 74396 in safety's ignore command-line argument or add the ignore to your safety policy file.

→ Vulnerability found in django version 5.1.2
Vulnerability ID: 74395
Affected spec: >=5.1.0,<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-2024-53987
For more information about this vulnerability, visit <https://data.safetycli.com/v/74395/97c>
To ignore this vulnerability, use PyPI vulnerability id 74395 in safety's ignore command-line argument or add the ignore to your safety policy file.

→ Vulnerability found in django version 5.1.2
Vulnerability ID: 74085
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_ip6_address, is_valid_ip6_address, and the django.forms.GenericIPAddressField form field allowed...
CVE-2024-56374
For more information about this vulnerability, visit <https://data.safetycli.com/v/74085/97c>
To ignore this vulnerability, use PyPI vulnerability id 74085 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 31 June 2024.

We highly encourage switching to the new 'sanit' command which is easier to use, more powerful, and can be set up to inherit 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) |