Task 3: Secure Coding Review

1. Introduction

This report presents the results of a security audit conducted on a simple Python application named `login.py`. The objective of this audit is to identify security vulnerabilities that might exist within the code and provide recommendations to remediate these issues. Security audits help maintain the integrity, confidentiality, and availability of software systems by proactively discovering and addressing potential flaws before they can be exploited.

We followed a structured methodology that includes manual code inspection and static analysis using Bandit, a well-known Python security analyzer. The issues discovered are documented with explanations and their potential implications, followed by suggested remediation measures.

2. Application and Environment Details

- Programming Language: Python 3.13.2
- Application File Audited: login.py
- Static Analysis Tool Used: Bandit v1.8.3
- Development Environment: Visual Studio Code on Windows 10
- Execution Platform: Windows Command Prompt

3. Audit Steps and Methodology

3.1 Selecting the Application and Tools

We selected a small Python script to illustrate the use of static analysis and secure coding practices. The script takes a username as input and displays a greeting using subprocess to run a shell command.

3.2 Manual Code Review

Before using automated tools, the code was reviewed manually to identify any immediate security concerns. The most prominent issue noticed was the use of `subprocess.call()` with `shell=True` combined with user input — a well-known vector for command injection attacks.

3.3 Static Code Analysis

Bandit was used to scan the code and identify potential vulnerabilities:

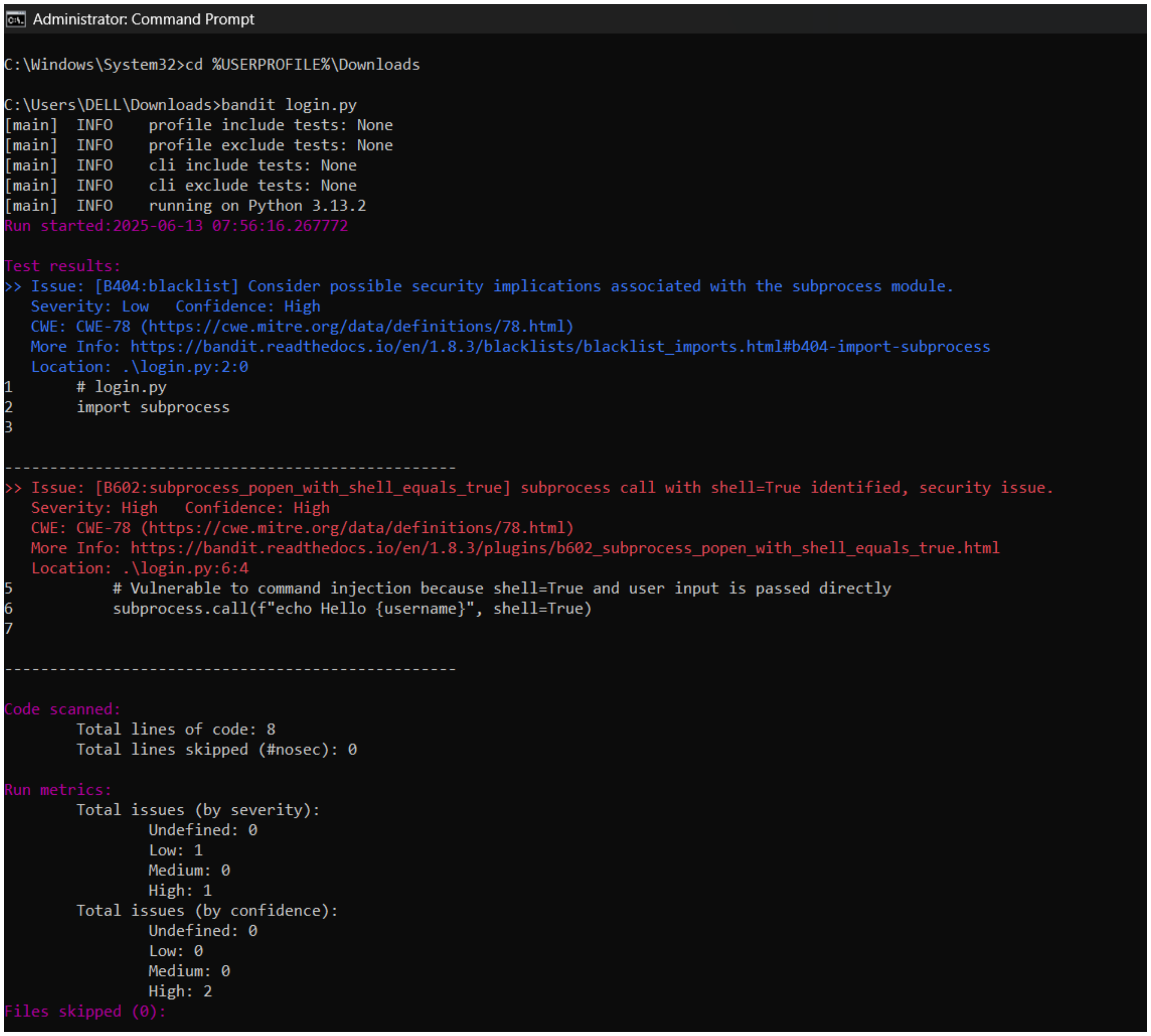
```
bandit login.py
```

The output provided issue codes, severity levels, and line numbers for each detected vulnerability.

4. Findings

4.1 Bandit Scan Results

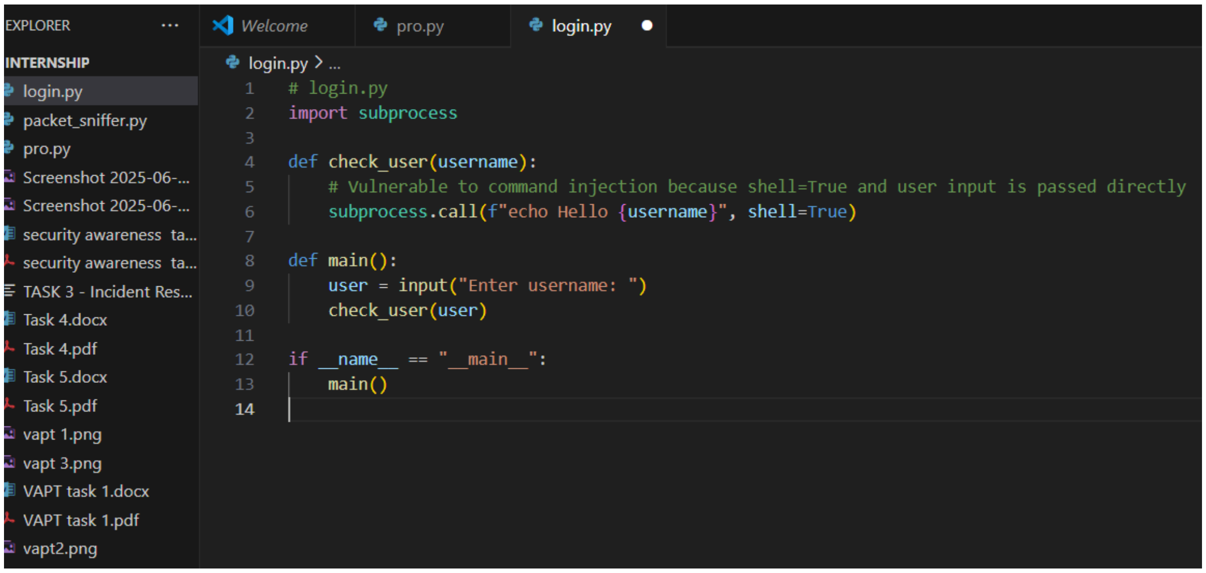
Issue Code	Description	Severity
B404	Use of subprocess module can pose security risks	Low
B602	subprocess.call() with shell=True allows command injection	High
B603	Subprocess usage without shell=True may still be insecure	Low
B607	Partial executable path in subprocess call	Low



Administrator: Command Prompt

```
C:\Windows\System32>cd %USERPROFILE%\Downloads

C:\Users\DELL\Downloads>bandit login.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 3.13.2
Run started:2025-06-13 07:56:16.267772

Test results:
>> Issue: [B404:blacklist] Consider possible security implications associated with the subprocess module.
   Severity: Low   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/blacklists/blacklist_imports.html#b404-import-subprocess
   Location: .\login.py:2:0
1       # login.py
2       import subprocess
3

--------------------------------------------------
>> Issue: [B602:subprocess_popen_with_shell_equals_true] subprocess call with shell=True identified, security issue.
   Severity: High   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b602_subprocess_popen_with_shell_equals_true.html
   Location: .\login.py:6:4
5           # Vulnerable to command injection because shell=True and user input is passed directly
6           subprocess.call(f"echo Hello {username}", shell=True)
7

--------------------------------------------------

Code scanned:
        Total lines of code: 8
        Total lines skipped (#nosec): 0

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

4.2 Manual Review Observations
- `subprocess.call(f"echo Hello {username}", shell=True)` was vulnerable to command injection.
- The code did not validate or sanitize the input.
- Windows command `echo` isn't a standalone executable, which caused FileNotFoundError.



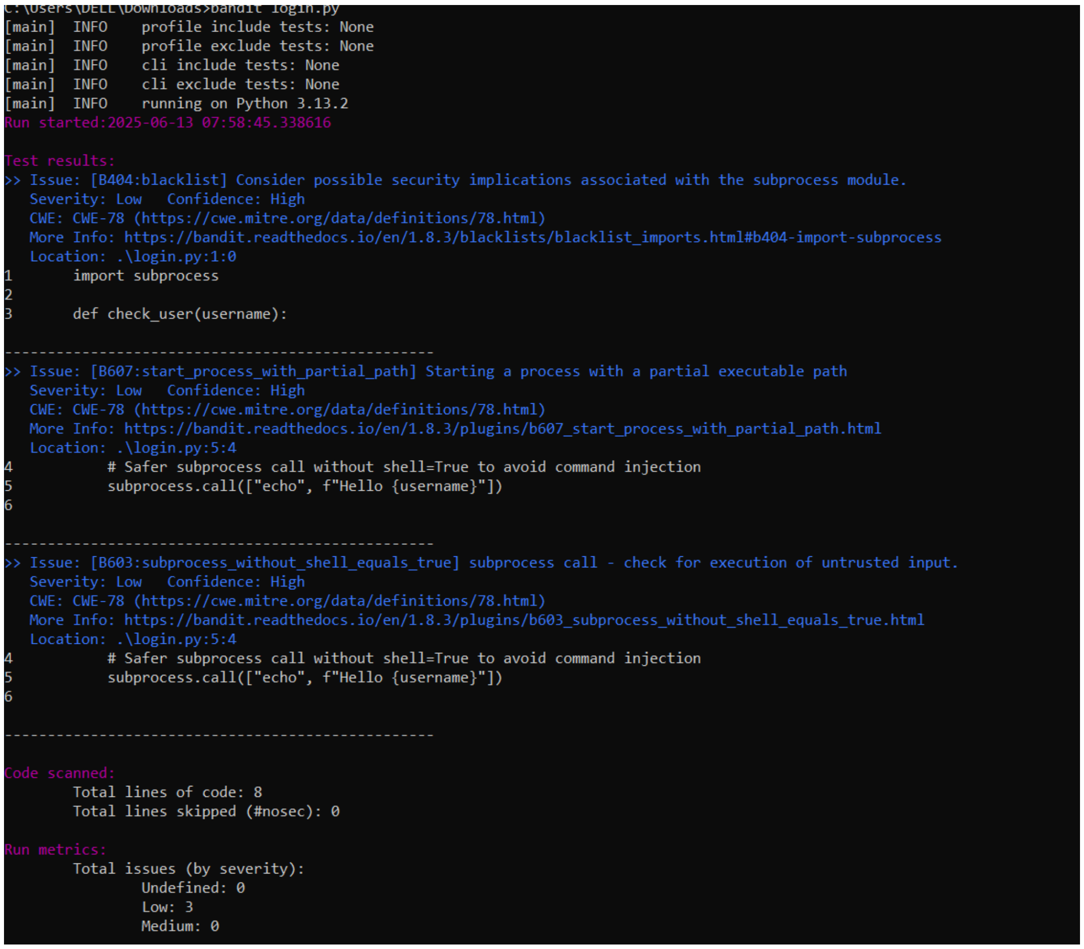
```python
# login.py
import subprocess

def check_user(username):
    # Vulnerable to command injection because shell=True and user input is passed directly
    subprocess.call(f"echo Hello {username}", shell=True)

def main():
    user = input("Enter username: ")
    check_user(user)

if __name__ == "__main__":
    main()

```

5. Recommendations and Best Practices

To address the vulnerabilities and follow secure coding practices:

- Input Validation: Always validate input to ensure it matches expected formats.
- Avoid shell=True: Never pass user input to shell commands using shell=True.
- Use Built-in Functions: Replace subprocess calls with native Python functionality when possible.
- Escape User Input: If shell execution is absolutely necessary, ensure all user inputs are properly sanitized.
- Cross-Platform Consideration: Use platform-independent logic to handle OS-level differences.

After fixing the code the result came as follow



```
C:\Users\DELL\Downloads>bandit login.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 3.13.2
Run started:2025-06-13 07:58:45.338616

Test results:
>> Issue: [B404:blacklist] Consider possible security implications associated with the subprocess module.
   Severity: Low   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/blacklists/blacklist_imports.html#b404-import-subprocess
   Location: .\login.py:1:0
1       import subprocess
2
3       def check_user(username):

--------------------------------------------------
>> Issue: [B607:start_process_with_partial_path] Starting a process with a partial executable path
   Severity: Low   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b607_start_process_with_partial_path.html
   Location: .\login.py:5:4
4           # Safer subprocess call without shell=True to avoid command injection
5           subprocess.call(["echo", f"Hello {username}"])
6

--------------------------------------------------
>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution of untrusted input.
   Severity: Low   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b603_subprocess_without_shell_equals_true.html
   Location: .\login.py:5:4
4           # Safer subprocess call without shell=True to avoid command injection
5           subprocess.call(["echo", f"Hello {username}"])
6

--------------------------------------------------

Code scanned:
        Total lines of code: 8
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 3
                Medium: 0
```

The code was no longer vulnerable

6. Remediation Steps

The script was updated as follows :

```
def is_valid_username(username):

   return username.isalnum()

def check_user(username):

   if not is_valid_username(username):

      print("Invalid username! Use only letters and numbers.")

      return

   subprocess.call(["cmd", "/c", "echo", f"Hello {username}"])
```

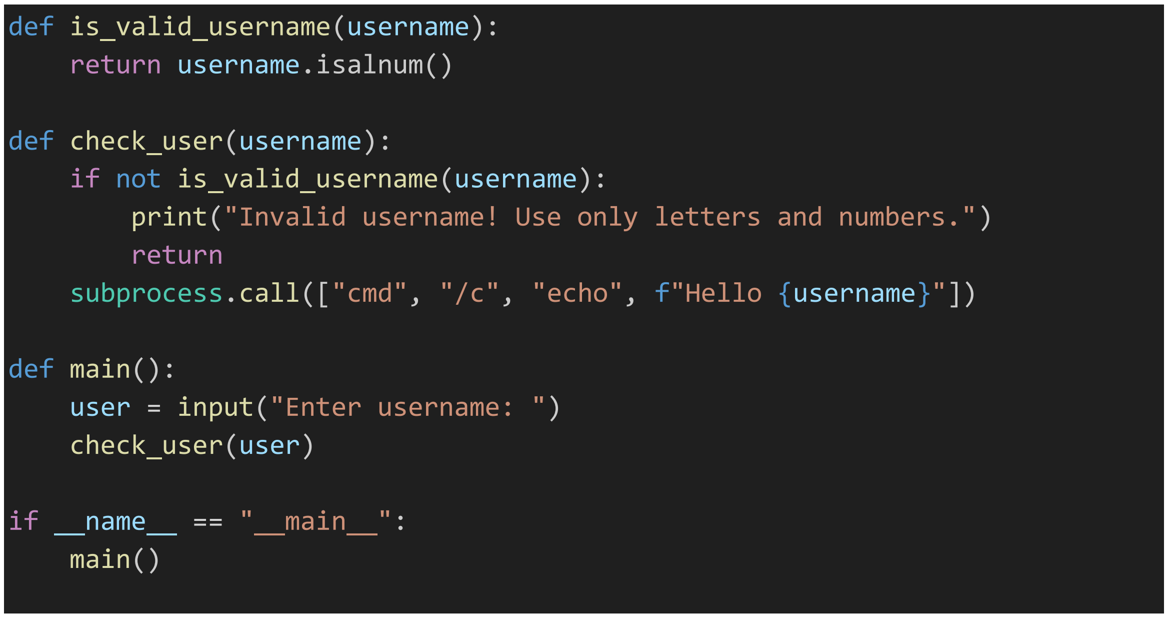
def main():

user = input("Enter username: ")

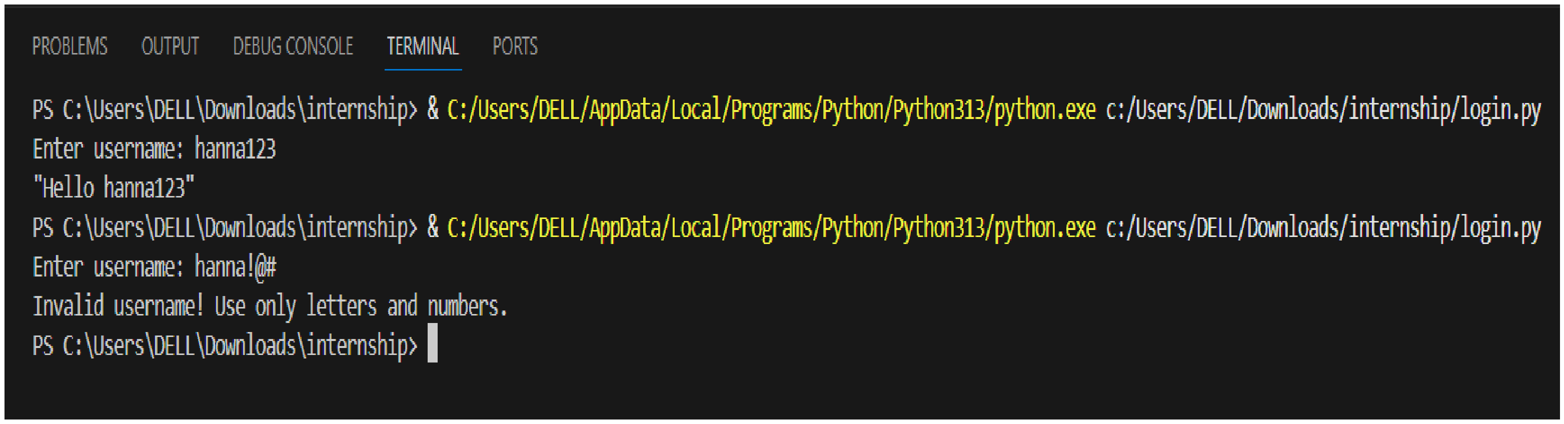
check_user(user)

if _name_ == "_main_":

main()



```
def is_valid_username(username):
    return username.isalnum()

def check_user(username):
    if not is_valid_username(username):
        print("Invalid username! Use only letters and numbers.")
        return
    subprocess.call(["cmd", "/c", "echo", f"Hello {username}"])

def main():
    user = input("Enter username: ")
    check_user(user)

if __name__ == "__main__":
    main()
```



```
PS C:\Users\DELL\Downloads\internship> & C:/Users/DELL/AppData/Local/Programs/Python/Python313/python.exe c:/Users/DELL/Downloads/internship/login.py
Enter username: hanna123
"Hello hanna123"
PS C:\Users\DELL\Downloads\internship> & C:/Users/DELL/AppData/Local/Programs/Python/Python313/python.exe c:/Users/DELL/Downloads/internship/login.py
Enter username: hanna!@#
Invalid username! Use only letters and numbers.
PS C:\Users\DELL\Downloads\internship>
```

7. Conclusion

The audit of login.py revealed significant security flaws, particularly in handling system commands with user input. By applying best practices such as validating input and avoiding shell=True, the application was secured from potential command injection attacks.

Tools like Bandit offer a quick and reliable way to identify such problems, but manual review remains important to catch logic issues that tools may miss. The revised code is now both safer and more robust.

8. Appendix

- Bandit Version: 1.8.3
- Python Version: 3.13.2
- Command Used: bandit login.py
- Editor Used: Visual Studio Code
- Operating System: Windows 10