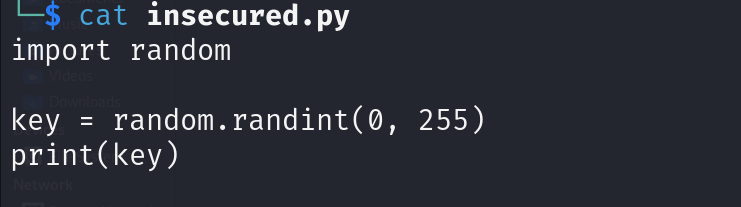
**Task 3**

**Secure Coding Review**

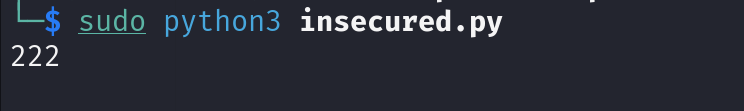
**Choose a programming language and application. Review the code for security vulnerabilities and provide recommendations for secure coding practices. Use tools like static code analyzers or manual code review.**

**INSECURED VERSION :**

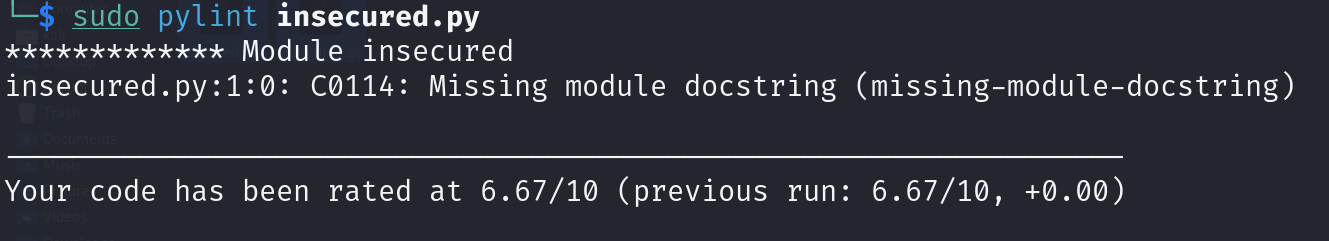
First python program where the python code is vulnerable. Because in generates a randon number below is the python code:



Below is the output of this python program:



Below is the rating of the program analyze with pylint static code analyzer:



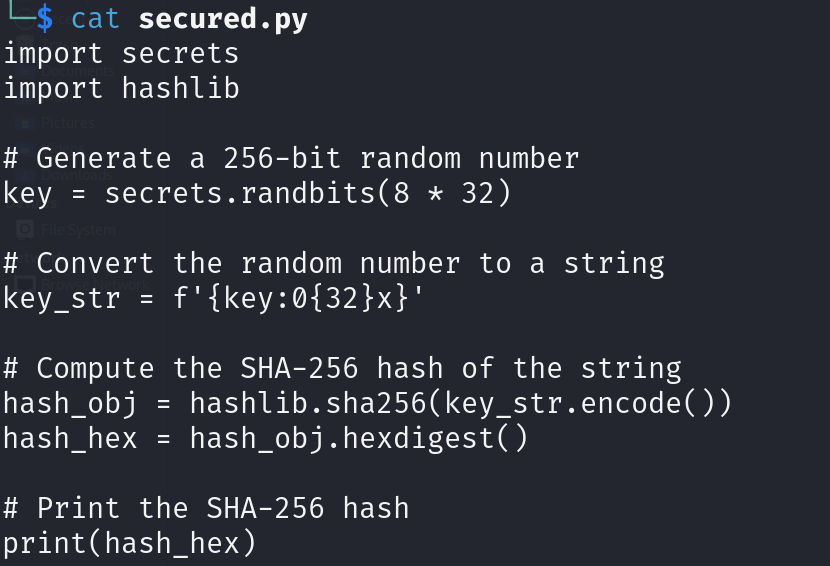
**SECURED VERSION :**

Below In this code, we first generate a 256-bit random number using the **secrets** module. We then convert the random number to a string using the **f** string syntax and the **hex()** function.

Next, we create a **hashlib.sha256()** object to compute the SHA-256 hash of the string. We call the **hexdigest()** method of the hash object to get the hash as a hexadecimal string.

Finally, we print the SHA-256 hash of the random number.

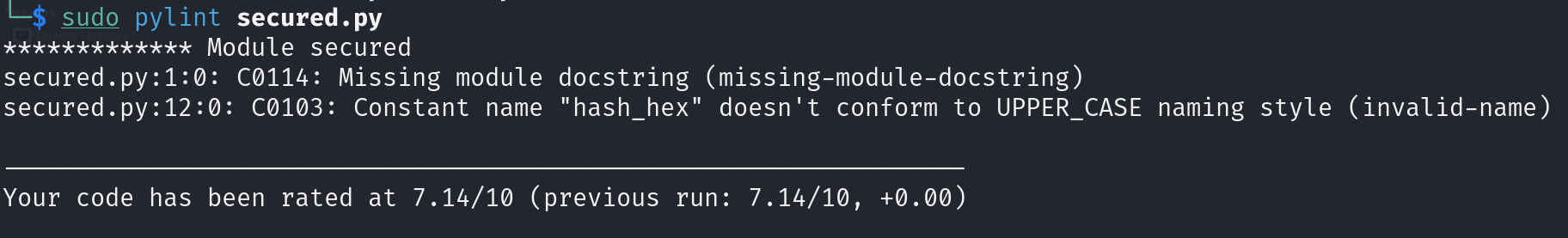
Here's the code in markdown format:



Below is the output of this python program:



Below is the rating of the program analyze with pylint static code analyzer:



\*\*\*\*\*\*\*\*\*\*\*Below are the things we must consider when coding a program\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **Input Validation:** Always validate and sanitize inputs, whether from a user, a file, or an API.
2. **SQL Injection Prevention:** Avoid SQL injection attacks by using parameterized queries when interfacing with databases.
3. **Avoid Dynamic Code Execution:** Refrain from using functions like eval(), exec(), or compile() with dynamic inputs, as they can execute malicious code. Instead, use safer methods or avoid dynamic execution altogether.
4. **Dependency Management:** Use tools like pipenv check or safety to check for known vulnerabilities in your dependencies. The fewer dependencies you have, the smaller the potential attack surface.
5. **Secret Management:** Never hard-code secrets like API keys, database passwords, or cryptographic keys directly in your code. Use secret management tools or environment variables instead.
6. **Secure Data Transmission:** When transmitting data over the internet, always use secure methods like HTTPS. Libraries such as requests make this relatively straightforward in Python.
7. **Secure Logging:** While logging is essential for debugging, ensure you're not logging sensitive information. Use logging levels appropriately and review logs to ensure they don't inadvertently expose secrets.
8. **Data Encryption:** If you're storing sensitive data, consider encrypting it. Python offers libraries like cryptography to help with symmetric and asymmetric encryption.
9. **Least Privilege Principle:** Run your Python applications with the least privilege necessary. Avoid running scripts as root or Administrator unless required.
10. **Continuous Learning:** Secure coding is an ongoing process. Continuously educate yourself on new threats and vulnerabilities. Join Python communities and security forums to stay in the loop.

By following these secure coding practices, you can significantly reduce the risk of security vulnerabilities in your Python applications.