Assignment: Secure OpenSSL Integration with Python and SSDLC Principles

Assignment: Secure OpenSSL Integration with Python and SSDLC Principles

Course: Secure Software Development
Due Date: *[Insert Date]*
Total Points: 100
Assignment Type: Individual

Learning Outcomes

By completing this assignment, students will be able to:

- Apply **SSDLC principles** to software development.
- Implement secure command execution using Python.
- Use **OpenSSL** in Python scripts to generate cryptographic keys and certificates.
- Apply secure file handling and logging practices.
- Recognize and mitigate **common security vulnerabilities** such as command injection and hardcoded secrets.

Problem Statement

Your task is to analyze, modify, and extend a secure Python script that wraps OpenSSL operations while following Secure Software Development Lifecycle (SSDLC) best practices. The current implementation generates keys, encrypts/decrypts files, creates certificates, and logs critical operations.

Tasks

Part 1: Code Review and SSDLC Analysis (20 points)

1. Download the provided Python script: `secure_openssl.py`.
2. Identify and explain how each SSDLC stage is represented in the code.
3. List at least **three security improvements or recommendations** for future versions.

Submit your findings in a short PDF or Markdown document.

Part 2: Modify the Script (40 points)

Enhance the existing script to do the following:

- Take password input securely using `getpass.getpass()` (no hardcoded passwords).
- Add user input validation for file paths using regex or `os.path`.
- Add command-line arguments using `argparse` to:
  - Specify the encryption password
  - Toggle encryption or decryption modes
  - Print hash of input/decrypted file

Part 3: Logging and Testing (20 points)

- Add log entries for:
  - Success or failure of key generation
  - Start and end of encryption/decryption processes
  - Hash output logging
- Create a test plan or checklist verifying:
  - Proper file generation
  - Successful encryption/decryption
  - Secure logging (no sensitive data in logs)

Part 4: Documentation (10 points)

Provide a short user manual that includes:

- How to run the script
- How to pass parameters
- Example usage
- Security considerations

Bonus Challenge (Up to 10 Points)

Implement a secure password vault integration using the `keyring` module or environment variables instead of plain text passwords.

Submission Instructions

Submit the following as a ZIP file or a GitHub repository:

- Modified `secure_openssl.py`
- SSDLC analysis and documentation (PDF or Markdown)
- Log file (`secure_openssl.log`)
- Test plan/checklist

1 Python Program: Secure OpenSSL Wrapper

Listing 1: Secure OpenSSL Integration in Python

```
import subprocess
import os
import sys
import tempfile
import logging

# Configure logging
logging.basicConfig(filename="secure_openssl.log", level=logging.INFO, format="%(
    asctime)s␣-␣%(levelname)s␣-␣%(message)s")

def run_openssl(command):
"""Executes an OpenSSL command securely."""
try:
result = subprocess.run(command, capture_output=True, text=True, check=True)
return result.stdout
except subprocess.CalledProcessError as e:
logging.error(f"OpenSSL␣command␣failed:␣{e.stderr}")
return None

def generate_private_key(key_path):
"""Generates a secure RSA private key."""
command = ["openssl", "genpkey", "-algorithm", "RSA", "-out", key_path, "-aes256"]
return run_openssl(command)

def generate_public_key(private_key, public_key):
"""Generates a public key from a private key."""
command = ["openssl", "rsa", "-in", private_key, "-pubout", "-out", public_key]
return run_openssl(command)

def create_self_signed_certificate(private_key, cert_path, days=365):
"""Creates a self-signed certificate."""
command = ["openssl", "req", "-x509", "-new", "-key", private_key, "-out", cert_path
    , "-days", str(days), "-subj", "/CN=SecureApp"]
return run_openssl(command)

def encrypt_file(input_file, encrypted_file, password):
"""Encrypts a file securely using AES-256."""
command = ["openssl", "enc", "-aes-256-cbc", "-salt", "-in", input_file, "-out",
    encrypted_file, "-pass", f"pass:{password}"]
return run_openssl(command)

def decrypt_file(encrypted_file, decrypted_file, password):
"""Decrypts a file securely."""
command = ["openssl", "enc", "-aes-256-cbc", "-d", "-in", encrypted_file, "-out",
    decrypted_file, "-pass", f"pass:{password}"]
return run_openssl(command)

def hash_file(file_path):
"""Computes SHA-256 hash of a file."""
command = ["openssl", "dgst", "-sha256", file_path]
return run_openssl(command)

if __name__ == "__main__":
try:
with tempfile.TemporaryDirectory() as tmpdir:
private_key = os.path.join(tmpdir, "private_key.pem")
public_key = os.path.join(tmpdir, "public_key.pem")
cert_path = os.path.join(tmpdir, "certificate.crt")
plaintext_file = os.path.join(tmpdir, "plaintext.txt")
encrypted_file = os.path.join(tmpdir, "encrypted.txt")
decrypted_file = os.path.join(tmpdir, "decrypted.txt")

# Create dummy plaintext file for encryption
with open(plaintext_file, "w") as f:
```

```
f.write("This␣is␣a␣secure␣test␣file.")

# Generate keys and certificate
generate_private_key(private_key)
generate_public_key(private_key, public_key)
create_self_signed_certificate(private_key, cert_path)

# Encrypt and Decrypt
password = "strongpassword123"  # In a real-world scenario, use a secure vault
encrypt_file(plaintext_file, encrypted_file, password)
decrypt_file(encrypted_file, decrypted_file, password)

# Hash the decrypted file
hash_result = hash_file(decrypted_file)
logging.info(f"SHA-256␣Hash:␣{hash_result.strip()}")

print("Secure␣OpenSSL␣operations␣completed␣successfully.")

except Exception as e:
logging.error(f"Unexpected␣error:␣{str(e)}")
sys.exit(1)
```