**CSDF Code:**

**Assignment 1: Email Header**

import json

import datetime

import pytz

def extract\_header(file\_name: str) -> dict:

with open(file\_name, 'r') as f:

spam\_header = f.read()

spam\_header\_text = spam\_header.replace("\t", "").split('\n')

final\_text = []

last\_colon = 0

for text in spam\_header\_text:

if text.find(': ') != -1:

final\_text.append(text)

last\_colon = len(final\_text) - 1

else:

final\_text[last\_colon] += text

spam\_header\_text = [x.split(": ") for x in final\_text]

spam\_header\_json = {x[0]: x[1] for x in spam\_header\_text}

return spam\_header\_json

def dump\_in\_json(content: dict, filename: str) -> None:

json.dump(content, open(filename, 'w'), indent=4)

def time\_required\_to\_deliver\_mail(header\_dict: dict) -> datetime.timedelta:

sent\_datetime = header\_dict['Received'].split(";")[-1].replace("\n", "").replace(" ", "").split("(")[0].strip()

received\_datetime = header\_dict['X-Received'].split(";")[-1].replace("\n", "").replace(" ", "").split("(")[0].strip()

strp\_format = "%a, %d %b %Y %H:%M:%S %z"

sent\_datetime = datetime.datetime.strptime(sent\_datetime, strp\_format).astimezone(pytz.utc)

received\_datetime = datetime.datetime.strptime(received\_datetime, strp\_format).astimezone(pytz.utc)

time\_required = received\_datetime - sent\_datetime

return time\_required

if \_\_name\_\_ == '\_\_main\_\_':

data = extract\_header('spam\_header2.txt')

dump\_in\_json(data, 'spam\_header2.json')

print(time\_required\_to\_deliver\_mail(data))

**Assignment 2: CAPTCHA Generate & verify**

import random

from captcha.image import ImageCaptcha

import streamlit as st

def verify\_captcha(user\_input, captcha\_text):

if user\_input == captcha\_text:

st.success("CAPTCHA Verification Successful!")

st.balloons()

else:

st.error("CAPTCHA Verification Failed!")

title = "CAPTCHA Verification"

st.title(title)

if title == "CAPTCHA Verification":

btn = st.button("Generate CAPTCHA", key='generate')

if btn:

image = ImageCaptcha(width=280, height=90)

captcha\_text = str(random.randint(1000, 9999))

data = image.generate(captcha\_text)

image.write(captcha\_text, 'CAPTCHA.png')

st.image('CAPTCHA.png')

user\_input = st.text\_input("Enter the CAPTCHA")

btn2 = st.button("Verify CAPTCHA", key='verify', help="Click this button to verify the CAPTCHA", on\_click=verify\_captcha, args=(user\_input, captcha\_text))

**Assignment 3:** **Recovering permanent Deleted Files and Deleted Partitions**

pip install pytsk3

import os

import shutil

import pytsk3

import datetime

from pathlib import Path

# Function to list the file entries in a partition

def list\_files\_in\_partition(disk\_image):

# Open the disk image

img = pytsk3.Img\_Info(disk\_image)

# Open the file system from the disk image

filesystem = pytsk3.FS\_Info(img)

# Getting the root directory of the partition

root\_dir = filesystem.open\_dir("/")

# Recursively list the files

def list\_directory(directory):

for entry in directory:

if entry.info.name:

print(f"File: {entry.info.name.name.decode()}")

# Here we would ideally try to recover the file by copying

# it to another location.

if entry.info.meta:

try:

# If the file is deleted, it will have a flag indicating so.

if entry.info.meta.flags == 2: # Deleted file

print(f"Deleted File Found: {entry.info.name.name.decode()}")

# Implement recovery logic here

except:

pass

if entry.info.meta and entry.info.meta.type == pytsk3.TSK\_FS\_META\_TYPE\_DIR:

list\_directory(entry.as\_directory())

list\_directory(root\_dir)

# Function to recover a deleted file

def recover\_deleted\_file(disk\_image, file\_path, output\_dir):

img = pytsk3.Img\_Info(disk\_image)

filesystem = pytsk3.FS\_Info(img)

# Recover the file by copying it

try:

file\_obj = filesystem.open(file\_path)

with open(os.path.join(output\_dir, os.path.basename(file\_path)), "wb") as output\_file:

output\_file.write(file\_obj.read\_random(0, file\_obj.info.meta.size))

print(f"Recovered file: {file\_path} to {output\_dir}")

except Exception as e:

print(f"Error recovering file: {str(e)}")

# Function to check for deleted partitions (using TestDisk)

def recover\_deleted\_partition(disk\_path):

# Use TestDisk (external tool) to recover partitions

try:

subprocess.run(["testdisk", disk\_path])

except Exception as e:

print(f"Error recovering partition: {str(e)}")

# Main function to run recovery based on user choice

def main():

print("1. Recover deleted files")

print("2. Recover deleted partitions")

choice = input("Enter choice (1 or 2): ")

disk\_image = input("Enter the path to the disk image (e.g., /dev/sda): ")

if choice == "1":

# Recover deleted files

list\_files\_in\_partition(disk\_image)

output\_dir = input("Enter the directory to save recovered files: ")

file\_path = input("Enter the path of the deleted file to recover: ")

recover\_deleted\_file(disk\_image, file\_path, output\_dir)

elif choice == "2":

# Recover deleted partitions using TestDisk

recover\_deleted\_partition(disk\_image)

else:

print("Invalid choice.")

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

main()

**Assignment 4:** **Log Capturing and Event Correlation**

import time

import json

from datetime import datetime

from collections import defaultdict

# Simulate a log generator that generates system activity logs

class LogGenerator:

def \_\_init\_\_(self):

self.events = [

{"event\_id": 1, "event\_type": "INFO", "description": "User login successful", "user": "Alice", "ip": "192.168.0.101"},

{"event\_id": 2, "event\_type": "INFO", "description": "User logout", "user": "Alice", "ip": "192.168.0.101"},

{"event\_id": 3, "event\_type": "ERROR", "description": "Failed login attempt", "user": "Bob", "ip": "192.168.0.102"},

{"event\_id": 4, "event\_type": "WARNING", "description": "Suspicious login from new IP", "user": "Alice", "ip": "192.168.0.105"},

{"event\_id": 5, "event\_type": "ERROR", "description": "Failed login attempt", "user": "Alice", "ip": "192.168.0.101"},

{"event\_id": 6, "event\_type": "INFO", "description": "Password changed", "user": "Alice", "ip": "192.168.0.101"},

{"event\_id": 7, "event\_type": "INFO", "description": "User login successful", "user": "Bob", "ip": "192.168.0.102"},

]

def generate\_logs(self, num\_logs=5):

for \_ in range(num\_logs):

event = self.events[\_ % len(self.events)]

event["timestamp"] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

yield json.dumps(event)

time.sleep(1)

# Function to capture logs

class LogCapture:

def \_\_init\_\_(self, log\_generator):

self.log\_generator = log\_generator

def capture\_logs(self, num\_logs=5):

logs = []

for log in self.log\_generator.generate\_logs(num\_logs):

logs.append(log)

return logs

# Event correlation logic

class EventCorrelation:

def \_\_init\_\_(self, logs):

self.logs = logs

def correlate\_events(self):

# Dictionary to store correlated events

correlation\_results = defaultdict(list)

# Iterate through logs to find suspicious activity

for log in self.logs:

log\_data = json.loads(log)

user = log\_data.get("user")

ip = log\_data.get("ip")

event\_type = log\_data.get("event\_type")

description = log\_data.get("description")

# Correlate multiple failed login attempts from the same IP or user

if event\_type == "ERROR" and ("Failed login attempt" in description):

correlation\_results[user].append(log\_data)

# Identify suspicious logins based on new IP addresses

if event\_type == "WARNING" and "Suspicious login from new IP" in description:

correlation\_results["suspicious"].append(log\_data)

return correlation\_results

# Main function to simulate log capturing and event correlation

def main():

# Step 1: Simulate log capturing

print("Capturing logs...")

log\_generator = LogGenerator()

log\_capture = LogCapture(log\_generator)

captured\_logs = log\_capture.capture\_logs(7) # Capturing 7 logs

print("Logs captured successfully!")

# Step 2: Correlate events

print("\nCorrelating events...")

event\_correlation = EventCorrelation(captured\_logs)

correlated\_events = event\_correlation.correlate\_events()

# Step 3: Display correlation results

print("\nEvent Correlation Results:")

for user, events in correlated\_events.items():

if user == "suspicious":

print("\nSuspicious Activity Detected:")

for event in events:

print(f"Timestamp: {event['timestamp']} | Event: {event['description']} | User: {event['user']} | IP: {event['ip']}")

else:

print(f"\nUser: {user} | Failed login attempts:")

for event in events:

print(f"Timestamp: {event['timestamp']} | Event: {event['description']} | User: {event['user']} | IP: {event['ip']}")

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

main()