EX.NO.: 06
DATE: 11.07.2025

LOG INTEGRITY MONITOR

AIM

To develop a Python-based tool that monitors one or more log files in real-time for unauthorized modifications using SHA-256 hashing. The system stores original file hashes in an SQLite database, detects tampering by comparing updated hashes upon file change events, and logs alerts with timestamps when tampering is detected.

ALGORITHM

1. Read the list of log file paths from `logs_to_monitor.txt`.
2. For each file:
    - Compute its SHA-256 hash.
    - Store the hash in an SQLite database (`database.db`) if not already present.
3. Use `watchdog` to monitor directories of the target files for changes (modified or renamed events).
4. On file change:
    - Recompute the file's SHA-256 hash.
    - Compare it with the stored hash from the database.
    - If the hashes differ:
        - Log an alert with a timestamp, file path, and message in the `alerts` table of the database.
        - Print the alert in the console.
        - Update the database with the new hash.
5. Continuously run the script, checking files in real-time.

CODE AND OUTPUT

```
import os
import hashlib
import sqlite3
import time
from datetime import datetime
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# Database setup
def setup_database():
    conn = sqlite3.connect('database.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS file_hashes (
                    path TEXT PRIMARY KEY,
                    hash TEXT
                )''')
    c.execute('''CREATE TABLE IF NOT EXISTS alerts (
                    timestamp TEXT,
                    file_path TEXT,
                    message TEXT
                )''')
    conn.commit()
    conn.close()

# Compute SHA-256 hash
```

```
def compute_hash(file_path):
    sha256 = hashlib.sha256()
    try:
        with open(file_path, 'rb') as f:
            while chunk := f.read(8192):
                sha256.update(chunk)
        return sha256.hexdigest()
    except Exception as e:
        return None

# Load monitored files
def load_files(file_list_path='logs_to_monitor.txt'):
    with open(file_list_path, 'r') as f:
        return [line.strip() for line in f.readlines() if line.strip()]

# Initialize hashes in DB
def initialize_hashes(file_paths):
    conn = sqlite3.connect('database.db')
    c = conn.cursor()
    for path in file_paths:
        hash_val = compute_hash(path)
        if hash_val:
            c.execute("INSERT OR REPLACE INTO file_hashes (path, hash) VALUES (?, ?)",
(path, hash_val))
    conn.commit()
    conn.close()

# Handle file events
class FileChangeHandler(FileSystemEventHandler):
    def on_modified(self, event):
        self.check_file(event.src_path)

    def on_moved(self, event):
        self.check_file(event.dest_path)

    def check_file(self, file_path):
        if not os.path.isfile(file_path):
            return
        conn = sqlite3.connect('database.db')
        c = conn.cursor()
        c.execute("SELECT hash FROM file_hashes WHERE path=?", (file_path,))
        row = c.fetchone()
        if row:
            old_hash = row[0]
            new_hash = compute_hash(file_path)
            if new_hash and old_hash != new_hash:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                message = f"Tampering detected in file: {file_path}"
                print(f"[ALERT] {timestamp} — {message}")
```

```
                c.execute("INSERT INTO alerts (timestamp, file_path, message) VALUES
(?, ?, ?)",
                          (timestamp, file_path, message))
                # Update to new hash
                c.execute("UPDATE file_hashes SET hash=? WHERE path=?", (new_hash,
file_path))
        conn.commit()
        conn.close()

# Run the monitor
def main():
    setup_database()
    file_paths = load_files()
    initialize_hashes(file_paths)

    event_handler = FileChangeHandler()
    observer = Observer()

    for path in file_paths:
        dir_path = os.path.dirname(path)
        observer.schedule(event_handler, path=dir_path, recursive=False)

    print("Monitoring started... Press Ctrl+C to stop.")
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()

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

```
(.venv) PS D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor> python 6-LogMonitor.py
Monitoring started... Press Ctrl+C to stop.
[ALERT] 2025-07-15 14:26:30 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
[ALERT] 2025-07-15 14:26:30 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
[ALERT] 2025-07-15 14:26:44 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
[ALERT] 2025-07-15 14:26:44 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
(.venv) PS D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor> python 6-LogMonitor.py
Monitoring started... Press Ctrl+C to stop.
[ALERT] 2025-07-15 14:29:56 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
[ALERT] 2025-07-15 14:29:56 — Tampering detected in file: D:\TARU\V th year\Intelligent Cyber security Lab\6-LogMonitor\dism.log
```

INFERENCE

The monitoring system accurately detects and alerts on any unauthorized modification to critical log files by validating their integrity through SHA-256 hashing and real-time file system monitoring. This enhances log security and supports forensic analysis.