```
import time
import logging
import hashlib
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

# --- CONFIGURATION SECTION ---
# DLP Watchlist: Keywords that trigger critical alerts if found in filenames
SENSITIVE_KEYWORDS = ['password', 'secret', 'confidential', 'card', 'budget',
'salary']

# LOGGING CONFIGURATION
# Creates a permanent audit trail file named 'file_audit.log'
logging.basicConfig(filename='file_audit.log', level=logging.INFO,
                    format='%(asctime)s - %(message)s', datefmt='%Y-%m-%d
%H:%M:%S')

def calculate_file_hash(filepath):
    """
    Calculates the SHA-256 hash of a file.
    This acts as a 'Digital Fingerprint' for File Integrity Monitoring (FIM).
    """
    sha256_hash = hashlib.sha256()
    try:
        with open(filepath, "rb") as f:
            # Read file in 4K chunks to handle large files efficiently
            for byte_block in iter(lambda: f.read(4096), b""):
                sha256_hash.update(byte_block)
        return sha256_hash.hexdigest()
    except Exception:
        return "N/A (File Busy or Deleted)"

class AuditHandler(FileSystemEventHandler):
    """
    The 'Security Guard' class that handles file system events.
    """
    def process(self, event):
        # Prevent feedback loop: Ignore the log file itself
        if event.src_path.endswith("file_audit.log"): return

        # --- DLP POLICY CHECK ---
        # 1. Check if the filename contains sensitive keywords
        is_sensitive = any(k in event.src_path.lower() for k in SENSITIVE_KEYWORDS)

        prefix = "ALERT"
        if is_sensitive:
            prefix = "!!! CRITICAL POLICY VIOLATION !!!"

        # 2. FILE INTEGRITY CHECK (FIM)
        # Calculate hash for created or modified files
```

```
        file_hash = "N/A"
        if event.event_type in ['created', 'modified'] and not event.is_directory:
            file_hash = calculate_file_hash(event.src_path)

        # 3. GENERATE ALERT MESSAGE
        if event.event_type == 'created':
            msg = f"{prefix}: New File Detected - {event.src_path} | Integrity
Hash: {file_hash}"
        elif event.event_type == 'modified':
            msg = f"{prefix}: File Modified - {event.src_path} | New Hash:
{file_hash}"
        elif event.event_type == 'deleted':
            msg = f"{prefix}: FILE DELETED - {event.src_path}"
        elif event.event_type == 'moved':
            msg = f"{prefix}: FILE MOVED/RENAMED - From {event.src_path} to
{event.dest_path}"
        else:
            return

        # 4. OUTPUT & LOGGING
        print(msg)
        if is_sensitive:
            logging.warning(msg) # Log as WARNING for violations
        else:
            logging.info(msg)    # Log as INFO for normal activity

    # Map all event types to the process function
    def on_modified(self, event): self.process(event)
    def on_created(self, event): self.process(event)
    def on_deleted(self, event): self.process(event)
    def on_moved(self, event): self.process(event)

# --- MAIN EXECUTION ---
if __name__ == "__main__":
    path_to_watch = "."
    observer = Observer()
    observer.schedule(AuditHandler(), path_to_watch, recursive=True)

    print("-" * 50)
    print(f"[*] SECURE FILE MONITORING SYSTEM ONLINE")
    print(f"[*] Mode: Active Monitoring")
    print(f"[*] DLP Watchlist: {SENSITIVE_KEYWORDS}")
    print(f"[*] Logging to: file_audit.log")
    print("-" * 50)

    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
```

```
    observer.stop()
    print("\n[!] Monitor Stopped.")
observer.join()
```