TRINITY

KJ's Educational Institute

KJ COLLEGE OF ENGINEERING AND MANAGEMENT RESEARCH

Affiliated to Savitribai Phule Pune University (SPPU), Approved by Govt. Of Maharashtra, Recognized by AICTE, New Delhi.

DEPARTMENT OF COMPUTER ENGINEERING

YEAR 2024 – 2025

SEMESTER – VII

Laboratory Practice-IV

[410246]

PROJECT TITLE:

"File System Forensics"

A Mini Project Report

on

"File System Forensics"

by

Harshada Jagtap(A-40)

Tejas Jagtap(A-41)

Vaishnavi Jagtap(A-42)

Under the guidance of

Dr.Aparna Hambarde

Department of Computer Engineering

K J's Educational Institute

K J College of Engineering & Management Research, Pune

SAVITRIBAI PHULE PUNE UNIVERSITY

2025-2026

K J's Educational Institute

K J College of Engineering & Management Research

(Accredited by NAAC)

Affiliated to Savitribai Phule Pune University (SPPU), Approved by Govt. of Maharashtra, Recognized by AICTE, New Delhi.

CERTIFICATE

This is to certify that,

Harshada Jagtap(A-40)
Tejas Jagtap(A-41)
Vaishnavi Jagtap(A-42)

of class B.E Computer; have successfully completed their mini project work on "**File System Forensics**" at K J College of Engineering & Management Research in the partial fulfilment of the Graduate Degree course in B.E at the department of Computer Engineering in the academic Year 2025-2026 Semester – I as prescribed by the Savitribai Phule Pune University.

Dr.Aparna s. Hambarde — Project Guide

Dr.Nikita Kulkarni — Head of Department

Acknowledgements

With deep sense of gratitude, we would like to thanks all the people who have lit our path with their kind guidance. We are very grateful to these intellectuals who did their best to help during our project work.

It is our proud privilege to express deep sense of gratitude to Dr. Suhas S. Khot, Principal of K J College of Engineering & Management Research, Pune for his comments and kind permission to complete this project. We remain indebted to Dr. Nikita Kulkarni, H.O.D. of Computer Engineering Department for her timely suggestion and valuable guidance.

The special gratitude goes to Dr.aparna Hambarde for her excellent and precious guidance in completion of this work .We thanks to all the colleagues for their appreciable help for our working project. With various industry owners or lab technicians to help, it has been our endeavor to throughout our work to cover the entire project work.

We also thankful to our parents who providing their wishful support for our project completion successfully. And lastly we thanks to our all friends and the people who are directly or indirectly related to our project work.

Harshada Jagtap(A-40)

Tejas Jagtap(A-41)

Vaishnavi Jagtap(A-42)

Contents

ABSTRACT

The **File System Forensics** project focuses on the systematic investigation and analysis of file systems such as **NTFS**, **FAT32**, and **EXT4** to extract crucial digital evidence and metadata related to file creation, modification, deletion, and access. In the modern digital world, where cybercrimes and unauthorized data manipulation are rapidly increasing, digital forensics has become a vital discipline for ensuring data integrity, tracing digital footprints, and uncovering hidden or deleted evidence. This project aims to provide a practical, lightweight, and accessible solution for forensic experts, security analysts, and system administrators to monitor and analyze changes occurring within file systems over time.

The developed system performs **forensic analysis** of directories and files by generating and maintaining **forensic baselines**, which record essential file metadata such as file name, path, size, hash values, and timestamps. When a folder is re-scanned, the application automatically compares the current state with the stored baseline to detect **added, deleted, or modified files**. This differential analysis enables investigators to pinpoint the exact nature and time of changes made to the file system, thereby providing valuable evidence for digital investigations.

The application is built using **Python** with the **Streamlit framework**, offering an interactive and user-friendly graphical interface. The system leverages the **hashlib** library to compute **SHA-256 hash values**, ensuring that even minor changes in file content are detected with cryptographic precision. Metadata extraction is handled through Python's built-in modules like **os**, **pathlib**, and **datetime**, ensuring efficient and platform-independent operation. Forensic records are stored in **JSON format**, providing a lightweight and easily interpretable structure for further analysis or integration with external tools.

In addition to technical precision, the project emphasizes **practical usability**. Unlike traditional forensic tools that require advanced technical expertise, this project offers simplicity and transparency, making it accessible to students, educators, and investigators alike. By visualizing file changes in a clear, color-coded interface, it enables faster understanding and decision-making during forensic analysis.

Overall, this project demonstrates how a well-designed, Python-based system can bridge the gap between professional forensic tools and accessible academic projects. It provides a foundational platform for **digital evidence collection, file activity monitoring, and integrity verification**—key components in any cybersecurity or forensic investigation process. The developed tool not only serves as a learning model but also has potential real-world applications in organizational auditing, ransomware detection, and insider threat monitoring.

Chapter 1: Introduction

In today's digital era, where vast amounts of information are created, modified, and shared daily, **data security and integrity** have become critical concerns. As cybercrimes such as unauthorized access, data breaches, ransomware, and digital fraud continue to rise, the need for **digital forensics**—the process of identifying, preserving, analyzing, and presenting digital evidence—has never been more important. Among the different branches of digital forensics, **file system forensics** plays a fundamental role, as it involves the analysis of file systems to uncover user activities, recover deleted or hidden data, and determine how and when data was accessed or modified.

A **file system** is a core component of any operating system that defines how data is stored, organized, and retrieved on a storage medium. Common examples include **FAT32**, **NTFS**, and **EXT4**. Each file system maintains metadata—information about files such as creation time, modification time, access permissions, and file size—which becomes crucial during forensic investigations. By analyzing this metadata, forensic experts can reconstruct user actions, detect unauthorized modifications, and retrieve traces of deleted or tampered files.

The primary goal of this project, **"File System Forensics,"** is to design and implement a system capable of analyzing the internal structure and metadata of file systems to extract forensic information. This includes retrieving details such as:

- **Access times** (when a file was last opened),
- **Modification times** (when a file was last edited),
- **Deletion indicators**, and
- **File metadata** (such as name, path, size, and hash values).

The proposed system works by creating a **forensic baseline** of a selected folder, capturing the state of files at a given time. During subsequent scans, it compares the new state with the baseline to detect changes such as **added, deleted, or modified files**. This comparison helps investigators or administrators quickly identify suspicious activities or tampering attempts.

To ensure accuracy and integrity, the project employs **cryptographic hashing (SHA-256)** to verify whether files have been altered. Additionally, the system is built using **Python** and **Streamlit**, providing a visually interactive and user-friendly interface. It allows even non-technical users to perform forensic analysis without relying on complex command-line tools. The baseline data and results are stored in **JSON format**, ensuring portability and easy sharing for further investigation.

In essence, the **File System Forensics** project bridges the gap between traditional digital forensic tools—which are often complex and resource-intensive—and simple, accessible, web-based analysis tools. It provides a platform for users to **analyze file system changes, monitor file integrity, and detect anomalies** efficiently. This not only aids forensic investigators but also supports educational and organizational applications, such as auditing and monitoring sensitive data directories.

Through this project, we aim to contribute to the growing field of digital forensics by offering a practical and efficient approach to **file integrity verification and change detection**, which are essential components of modern cybersecurity defense systems.

Chapter 2: Literature Review

The field of **digital forensics** has evolved significantly over the past two decades, driven by the increasing complexity of file systems and the growing frequency of cybercrimes. **File system forensics**, a crucial subdomain of digital forensics, focuses on analyzing file structures, metadata, and storage mechanisms to uncover digital evidence, reconstruct user activities, and detect malicious modifications. This review summarizes key research works, existing tools, and methodologies relevant to the design and implementation of the **File System Forensics** project.

1. Background of File System Forensics

File system forensics involves studying how operating systems manage files on storage devices. Each file system—such as **FAT32**, **NTFS**, and **EXT4**—maintains metadata about files, including timestamps, access permissions, and allocation details. Forensic experts analyze these structures to determine what happened on a system, when it occurred, and who might be responsible.
According to **Carrier (2005)** in *File System Forensic Analysis*, understanding the internal architecture of file systems is essential for accurately recovering and interpreting digital evidence. His work established the foundational methodology for investigating low-level storage data, including Master File Table (MFT) records in NTFS and inodes in EXT-based systems.

2. Existing Forensic Tools

Several commercial and open-source tools have been developed to aid in file system analysis:

Autopsy and The Sleuth Kit (TSK): These open-source forensic suites provide comprehensive disk and file system analysis capabilities. They can recover deleted files, view metadata, and extract digital artifacts. However, they often require command-line expertise and detailed configuration.

FTK Imager: A widely used tool for creating forensic disk images and previewing file system contents. It supports metadata extraction and hashing but lacks customizable automation for continuous folder monitoring.

EnCase: A commercial forensic tool used by law enforcement and corporate investigators. It offers powerful search, analysis, and reporting features but is expensive and complex for educational or small-scale use.

Scalpel and Foremost: Lightweight data recovery tools that perform file carving based on headers and footers, useful for reconstructing deleted files.

While these tools are effective for professional investigations, they are often **resource-intensive, non-interactive, and difficult for beginners** to operate. This limitation highlights the need for a **simple, lightweight, and user-friendly forensic system** that can perform change detection and metadata analysis in real time.

3. Research on Metadata and Integrity Verification

Recent studies emphasize the importance of metadata and cryptographic methods in digital evidence handling.

> **Karie and Venter (2015)** discussed the reliability of timestamps in digital investigations and how inconsistencies in system clocks can affect forensic interpretations.
>
> **Mohan et al. (2018)** proposed the use of cryptographic hashing (MD5, SHA-256) for verifying file integrity, ensuring that digital evidence remains unaltered from the time of acquisition.
>
> **Hargreaves and Patterson (2012)** introduced the concept of differential analysis, where current file system states are compared against previous baselines to detect anomalies and tampering attempts.

The current project adopts similar concepts by generating **forensic baselines** and using **SHA-256 hashing** to ensure data authenticity and detect file modifications over time.

4. Role of Automation and Visualization

Traditional forensic tools focus primarily on deep technical analysis but often lack visualization and automation features.
Garfinkel (2010) highlighted that automation in digital forensics can significantly reduce analysis time while maintaining accuracy.
The rise of **Python-based frameworks** like Streamlit and Dash has made it possible to build intuitive forensic dashboards that present findings in clear, visual formats.
Our project leverages these modern frameworks to automate folder comparisons and display forensic results interactively, helping even non-expert users understand file system changes.

5. Identified Research Gap

Despite the availability of robust forensic tools, most existing solutions:

- Require deep technical expertise.
- Lack real-time comparison features for continuous monitoring.
- Are not easily adaptable for academic or small-scale forensic education.

This gap inspired the development of the **File System Forensics** project — a Python-based, interactive, and educational tool for performing **folder-level forensic analysis**. The system's capability to automatically record metadata, generate baselines, and visualize results bridges the gap between professional forensic tools and beginner-friendly learning platforms.

6. Summary

In summary, the literature review highlights that while significant advancements have been made in digital and file system forensics, there remains a demand for accessible, visual, and automated forensic solutions. The proposed system builds upon foundational principles of metadata analysis, hashing, and differential comparison to deliver a simplified yet effective platform for forensic investigation and file integrity verification.

Chapter 3 : Software and Hardware Requirement Specification

The successful execution of the **File System Forensics** project depends on an appropriate combination of software tools and hardware resources. Since the project involves analyzing file system structures, computing cryptographic hashes, and managing metadata storage, both hardware efficiency and software compatibility are essential for accurate and smooth operation.

1. Software Requirements

a. Operating System

Windows 10 / 11, **Linux (Ubuntu 20.04 or later)**, or **macOS**

The project is platform-independent since it is implemented using Python and Streamlit.

b. Programming Language

Python 3.11 or above
Used for developing the core application logic, file system interaction, hashing, metadata extraction, and data visualization.

c. Python Libraries / Modules

The following Python modules are required for the execution of the project:

Streamlit :	To build an interactive web-based user interface for the forensic dashboard.
OS :	To interact with the operating system and access directories and files.
Pathlib :	Provides object-oriented file path handling and traversal.
Datetime :	Used to record file creation, modification, and access timestamps.
Hashlib :	Generates cryptographic hash values (SHA-256) to verify file integrity.
JSON :	To store and retrieve forensic baselines and results in structured format.
Pandas :	For tabular data management and visualization of forensic results.

d. Development EnvironmentVisual Studio Code or PyCharm IDE

Provides an integrated environment for writing, testing, and debugging Python code.

e. Web Framework

Streamlit (v1.31 or above)
Enables the creation of the web-based dashboard for displaying forensic data interactively without requiring HTML or JavaScript coding.

f. Browser Compatibility

Google Chrome, **Mozilla Firefox**, **Microsoft Edge**, or any modern browser supporting HTML5.

Hardware Requirements

- Processor: Intel i3 or higher
- RAM: Minimum 4 GB
- Storage: 500 MB free disk space
- Internet: Optional (for updates)

Chapter 4: Implementation

The **File System Forensics** project has been implemented as a **Streamlit-based Python application** that performs forensic analysis of file systems by monitoring folders, generating baselines, and identifying changes such as file additions, deletions, or modifications. The system integrates multiple Python modules and logical components to ensure efficient and reliable forensic operations.

This section provides a detailed explanation of the implementation process, including the architecture, workflow, and core modules used.

1. System Architecture

The architecture of the project follows a **modular design**, which consists of four main components:

User Interface Layer – Developed using Streamlit, this layer allows users to interact with the application through buttons, folder selectors, and visual reports.

Processing Layer – Contains the logic for scanning folders, collecting file metadata, and generating hash values.

Storage Layer – Stores forensic baselines (in JSON format) that capture the snapshot of a folder at a specific time.

Comparison and Reporting Layer – Compares current folder data with baseline data and generates a detailed forensic report showing added, deleted, or modified files.

2. Implementation Steps

Step 1: Project Initialization

The system begins by importing all required Python libraries:

os and pathlib for accessing and navigating file systems.

hashlib for generating cryptographic SHA-256 hash values.

datetime for recording timestamps (creation, modification, and access times).

json for storing and loading forensic baselines.

pandas for tabular data management and report generation.

streamlit for creating the graphical user interface.

The Streamlit configuration (st.set_page_config) defines the layout and title of the application.

Step 2: Folder Selection and Scanning

The user selects a folder through the Streamlit interface.
Once selected, the program performs a **recursive directory scan** using the pathlib.Path().rglob() function, which iterates through all files and subdirectories.

For each file, metadata is collected, including:

- File name and path
- File size
- Creation time
- Last modification time
- Access time
- SHA-256 hash value (to ensure file integrity)

This information is stored in a **dictionary structure**, which forms the core of the forensic baseline.

Step 3: Baseline Creation

After the initial scan, the collected metadata is saved as a **baseline file** in JSON format.
This baseline acts as the reference point for future comparisons.

Example structure of a baseline entry:


```
{
 "filename": "report.docx",
 "path": "C:/Users/Harsh/Desktop/Project/report.docx",
 "size": 45678,
 "modified_time": "2025-10-30 14:15:21",
 "hash": "a4b2f9d67c5e..."}
```


Each baseline file is named using a **unique hash identifier** derived from the folder path to prevent duplication.

Step 4: Forensic Comparison

When the user rescans the same folder, the system compares the **current folder snapshot** with the stored baseline to detect any changes.
The comparison logic classifies files into three categories:

Added Files: Files present in the new scan but not in the baseline.

Deleted Files: Files present in the baseline but missing in the new scan.

Modified Files: Files with matching names but different hash values or timestamps.

This comparison process ensures both **content-level** and **metadata-level** integrity checking.

Step 5: Report Generation and Visualization

Once the comparison is complete, the results are converted into a **Pandas DataFrame** and displayed within the Streamlit dashboard.
Each file entry is color-coded for clarity:

Added Files: Green background

Deleted Files: Red background

Modified Files: Yellow background

Streamlit widgets such as st.table() and st.dataframe() are used to visualize the data interactively. Investigators can scroll, sort, or filter the table for efficient analysis.

Step 6: Saving and Updating Baselines

After analysis, users can save the updated folder state as a **new baseline**.
This allows for version tracking over multiple time intervals, providing a timeline of file system activity.

The baseline data is stored inside a hidden directory .folder_forensics_baselines, ensuring organized and secure storage without user interference.

3. Data Structures Used

Dictionary: Used to store file attributes (metadata and hash).

List: To hold multiple file entries for iteration and comparison.

JSON File: Acts as the persistent storage format for forensic baselines.

Pandas DataFrame: For tabular report generation and visualization.

4. Security and Integrity

Hashing Algorithm: SHA-256 ensures file content integrity with minimal chances of collision.

Data Privacy: Only metadata and hash values are stored — no file content is accessed or copied, preserving confidentiality.

Read-Only Analysis: The system performs analysis without altering the original files, maintaining forensic soundness.

5. Advantages of the Implementation

User-friendly GUI — no need for command-line expertise.

Lightweight and platform-independent.

Real-time folder analysis and visualization.

Modular and easily extendable for future enhancements (e.g., log exports, AI-based anomaly detection).

6. Example Workflow

- User opens the Streamlit app.
- Selects a folder for forensic analysis.
- System performs scanning and generates a baseline JSON file.
- Later, the same folder is rescanned.
- The system compares both states and highlights added, deleted, or modified files.
- Results are displayed on the dashboard and can be exported or saved for record-keeping.

7. Summary

The implementation of **File System Forensics** successfully demonstrates the ability to track file system changes and preserve digital evidence integrity. By combining Python's robust libraries with Streamlit's visualization capabilities, the system bridges the gap between complex forensic tools and accessible analysis environments. The modular structure and automated comparison mechanism make this solution practical for both academic and professional forensic use cases.

Chapter 5: Code

App.py

```
import streamlit as st # For creating the web app UI
import os # For folder and file operations
from pathlib import Path # For easy path handling
import json # For saving and loading data in JSON format
from datetime import datetime # For timestamps (date & time)
import hashlib # For hashing folder names (unique file IDs)
import pandas as pd # For tabular data display
# Page Settings
st.set_page_config(page_title="Folder Forensics", layout="wide")
# Inject custom CSS
st.markdown("""
<style>
body {
font-family: 'Segoe UI', sans-serif;
}
.main {
background-color: #f8fafc;
padding: 20px;
}
.stMetric {
background: white;
padding: 15px;
```

```
border-radius: 12px;
box-shadow: 0 2px 6px rgba(0,0,0,0.1);
text-align: center;
}
.status-added {
background: #d1fae5;
color: #065f46;
padding: 6px 12px;
border-radius: 8px;
display: inline-block;
font-weight: 600;
}
.status-deleted {
background: #fee2e2;
color: #991b1b;
padding: 6px 12px;
border-radius: 8px;
display: inline-block;
font-weight: 600;
}
.status-changed {
background: #fef9c3;
color: #92400e;
padding: 6px 12px;
border-radius: 8px;
display: inline-block;
```

```
font-weight: 600;
}
.status-unchanged {
background: #e0e7ff;
color: #3730a3;
padding: 6px 12px;
border-radius: 8px;
display: inline-block;
font-weight: 600;
}
table {
border-radius: 8px !important;
overflow: hidden;
}
</style>
""", unsafe_allow_html=True)
# Internal directory to store baselines
BASE_DIR = Path(".folder_forensics_baselines")
BASE_DIR.mkdir(exist_ok=True)
# --- Helper Functions ---
# Generate a unique hash for each folder path
def safe_name_hash(path_str: str):
return hashlib.sha1(path_str.encode("utf-8")).hexdigest()
# Convert a timestamp into readable date-time format
def timestamp(ts):
try:
```

```
return datetime.fromtimestamp(ts).strftime("%Y-%m-%d %H:%M:%S")
except Exception:
return ""
# Convert file size (in bytes) to KB, MB, GB, etc.
def human_readable_size(size_in_bytes):
if size_in_bytes is None:
return None
for unit in ['B', 'KB', 'MB', 'GB', 'TB']:
if size_in_bytes < 1024:
return f"{size_in_bytes:.2f} {unit}"
size_in_bytes /= 1024
return f"{size_in_bytes:.2f} PB"
# Collect details about a single file (size, created time, modified time, etc.)
def get_file_info(path: Path):
try:
s = path.stat()
return {
"name": path.name,
"relpath": str(path),
"size": human_readable_size(s.st_size),
"created": timestamp(s.st_ctime),
"modified": timestamp(s.st_mtime),
"accessed": timestamp(s.st_atime),
"is_file": path.is_file()
}
except Exception as e:
```

```
return {
"name": path.name,
"relpath": str(path),
"size": None,
"created": None,
"modified": None,
"accessed": None,
"is_file": path.is_file(),
"error": str(e)
}
# Scan an entire folder and collect information about all files
def scan_folder(folder_path: str):
p = Path(folder_path)
if not p.exists():
raise FileNotFoundError(f"Path not found: {folder_path}")
results = {}
for root, dirs, files in os.walk(p):
for fname in files:
# Skip Word temporary/lock files (~$ prefix)
if fname.startswith("~$"):
continue
fpath = Path(root) / fname
info = get_file_info(fpath)
results[str(fpath)] = info
return results
# Save folder scan result into a JSON file (baseline)
```

```
def save_baseline_for(path_str: str, data: dict):
key = safe_name_hash(path_str) # Create unique file name
fp = BASE_DIR / f"{key}.json" # Path to JSON file
with open(fp, "w", encoding="utf-8") as f:
json.dump({
"folder": path_str,
"created_at": datetime.now().isoformat(),
"files": data
}, f, indent=2, ensure_ascii=False)
return fp
# Load baseline JSON (previous scan data)
def load_baseline_for(path_str: str):
key = safe_name_hash(path_str)
fp = BASE_DIR / f"{key}.json"
if not fp.exists():
return None
with open(fp, "r", encoding="utf-8") as f:
return json.load(f)
# Compare old baseline with new scan
def compare(old: dict, new: dict):
old_set = set(old.keys()) #old file path
new_set = set(new.keys()) #new file path
# Find which files were added, deleted, or common
added = sorted(list(new_set - old_set))
deleted = sorted(list(old_set - new_set))
common = sorted(list(old_set & new_set))
```

```
changed = []
unchanged = []
# For files present in both — check if size or modified time changed
for p in common:
o, n = old[p], new[p]
if (o.get("size") != n.get("size")) or (o.get("modified") != n.get("modified")):
changed.append(p)
else:
unchanged.append(p)
return {"added": added, "deleted": deleted, "changed": changed, "unchanged": unchanged}
# ----------------------------------------------------
#     STREAMLIT USER INTERFACE
# ----------------------------------------------------
# App title and short description
st.title("    Folder Forensics Dashboard")
st.markdown("A modern forensic analysis tool for **tracking file changes over time**.")
# Sidebar
with st.sidebar:
st.header("   Options")
folder_input = st.text_input("    Folder path", value=str(Path.home()))
btn_scan_baseline = st.button("    Create / Update Baseline")
btn_load_baseline = st.button("    Load Existing Baseline")
btn_compare = st.button("    Scan & Compare with Baseline")
if folder_input:
existing = load_baseline_for(folder_input)
if existing:
```

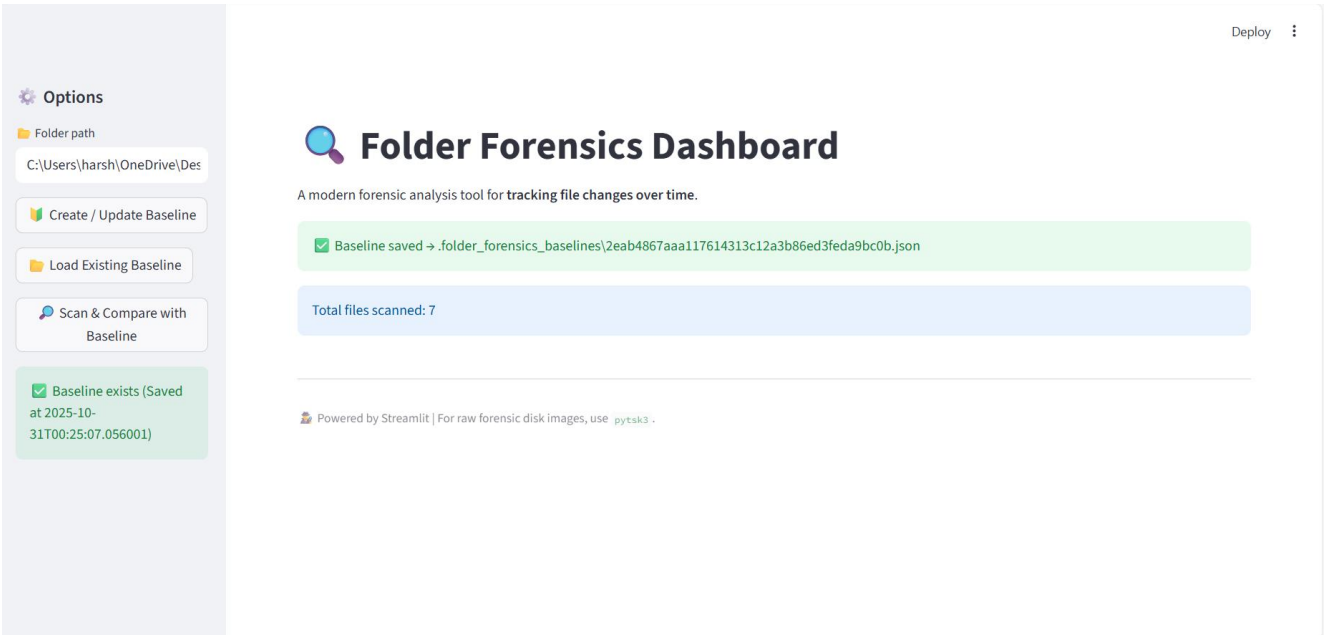
```
st.sidebar.success(f"✅ Baseline exists (Saved at {existing.get('created_at','?')})")
else:
st.sidebar.info("ℹ️ No baseline found for this folder.")
# --- Create Baseline ---
if btn_scan_baseline and folder_input:
try:
with st.spinner("⏳ Scanning folder and creating baseline..."):
current = scan_folder(folder_input)
fp = save_baseline_for(folder_input, current)
st.success(f"✅ Baseline saved → {fp}")
st.info(f"Total files scanned: {len(current)}")
except Exception as e:
st.error(f"❌ Error: {e}")
# --- Load Baseline ---
if btn_load_baseline and folder_input:
b = load_baseline_for(folder_input)
if not b:
st.warning("⚠️ Baseline not found for this folder.")
else:
st.subheader("📄 Baseline Preview")
st.write(f"Folder: {b.get('folder')}")
st.write(f"Saved at: {b.get('created_at')}")
files = b.get("files", {})
df = pd.DataFrame.from_dict(files, orient="index")
if not df.empty:
st.dataframe(df[["name","relpath","size","created","modified","accessed"]].head(500))
```

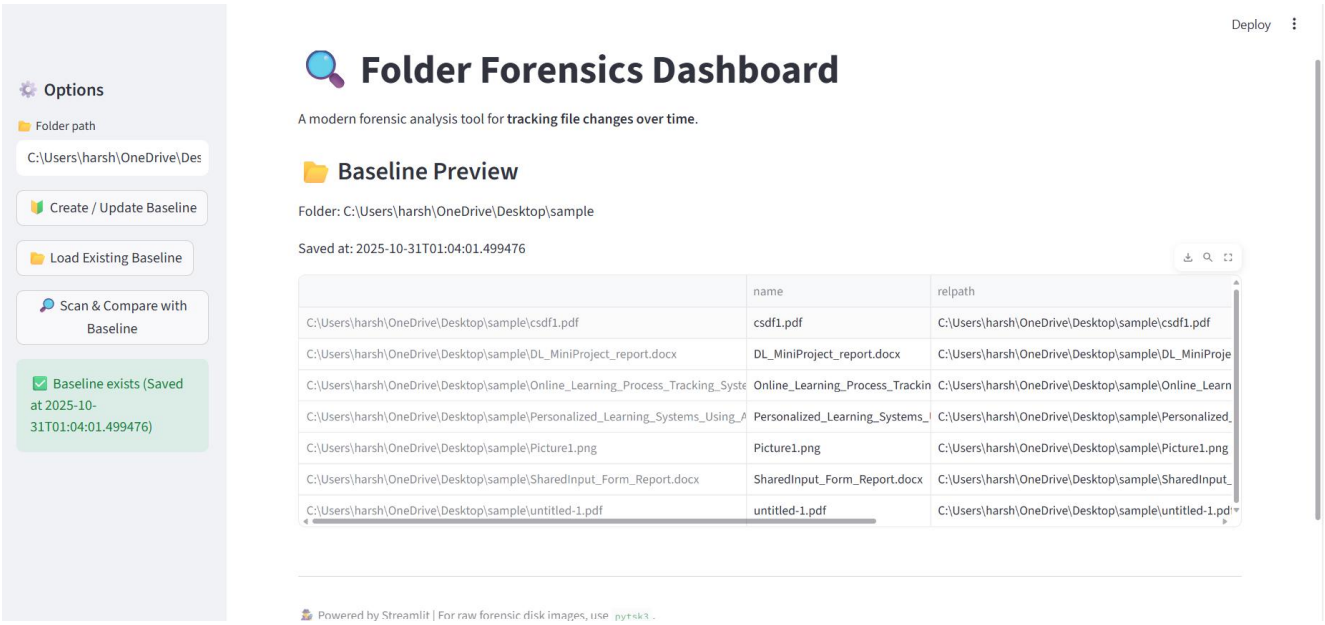
```
else:
st.write("No files in baseline.")
# --- Compare ---
if btn_compare and folder_input:
baseline = load_baseline_for(folder_input)
if not baseline:
st.warning("   Baseline not found. Please create one first.")
else:
try:
with st.spinner("    Scanning current folder and comparing..."):
curr = scan_folder(folder_input)
comp = compare(baseline["files"], curr)
# Summary
st.subheader("    Comparison Summary")
col1, col2, col3, col4 = st.columns(4)
col1.metric("Baseline files", len(baseline["files"]))
col2.metric("Current files", len(curr))
col3.metric("Added", len(comp["added"]))
col4.metric("Deleted", len(comp["deleted"]))
# Added
st.subheader("✅ Added files")
if comp["added"]:
df_added = pd.DataFrame([curr[p] for p in comp["added"]])
st.dataframe(df_added[["name","relpath","size","created","modified"]])
else:
st.markdown("<span class='status-unchanged'>None</span>", unsafe_allow_html=True)
```

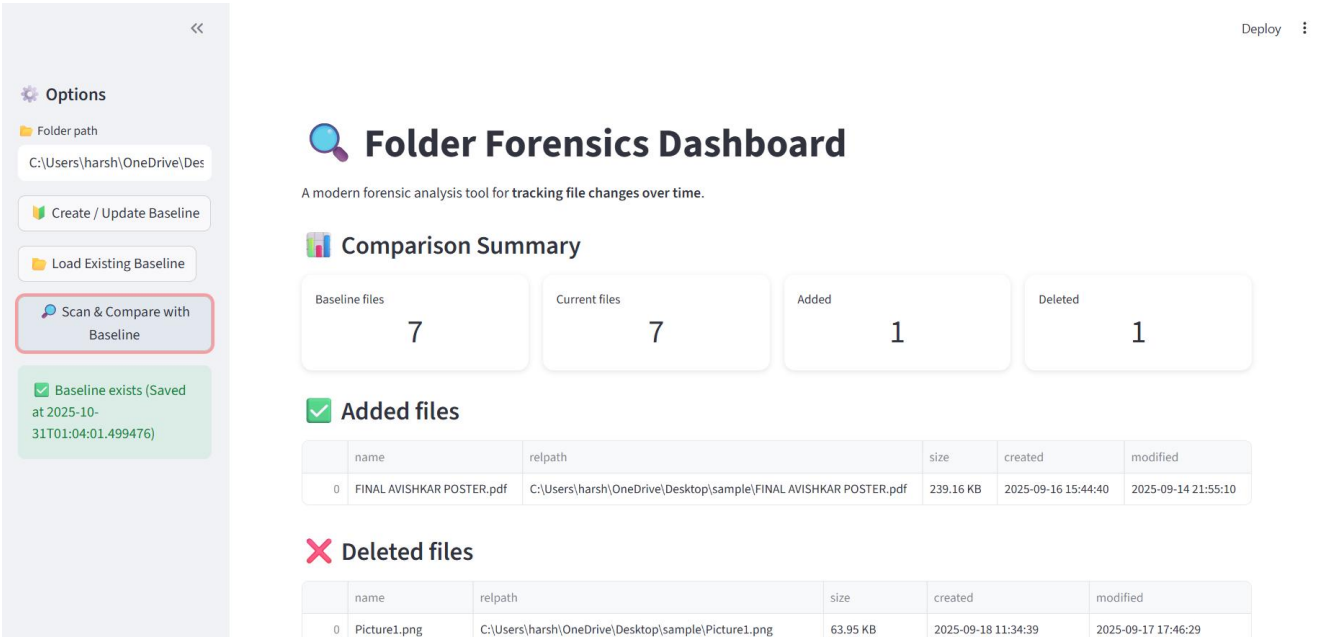
```
# Deleted
st.subheader("❌ Deleted files")
if comp["deleted"]:
df_del = pd.DataFrame([baseline["files"][p] for p in comp["deleted"]])
st.dataframe(df_del[["name","relpath","size","created","modified"]])
else:
st.markdown("<span class='status-unchanged'>None</span>", unsafe_allow_html=True)
# Changed
st.subheader("    Changed files")
if comp["changed"]:
df_ch = pd.DataFrame([curr[p] for p in comp["changed"]])
st.dataframe(df_ch[["name","relpath","size","created","modified"]])
else:
st.markdown("<span class='status-unchanged'>None</span>", unsafe_allow_html=True)
except Exception as e:
st.error(f"❌ Error during compare: {e}")
# Footer
st.markdown("---")
st.caption("    Powered by Streamlit | For raw forensic disk images, use `pytsk3`.")
```

OUTPUT


Options
Folder path
C:\Users\harsh\OneDrive\Des
Press Enter to apply
Create / Update Baseline
Load Existing Baseline
Scan & Compare with Baseline
Deploy
Folder Forensics Dashboard
A modern forensic analysis tool for tracking file changes over time.
Powered by Streamlit | For raw forensic disk images, use pytsk3 .


Options
Folder path
C:\Users\harsh\OneDrive\Des
Create / Update Baseline
Load Existing Baseline
Scan & Compare with Baseline
Baseline exists (Saved at 2025-10-31T00:25:07.056001)
Deploy
Folder Forensics Dashboard
A modern forensic analysis tool for tracking file changes over time.
Baseline saved → .folder_forensics_baselines\2eab4867aaa117614313c12a3b86ed3feda9bc0b.json
Total files scanned: 7
Powered by Streamlit | For raw forensic disk images, use pytsk3 .


Deploy
Options
Folder path
C:\Users\harsh\OneDrive\Des
Create / Update Baseline
Load Existing Baseline
Scan & Compare with Baseline
Baseline exists (Saved at 2025-10-31T01:04:01.499476)
Folder Forensics Dashboard
A modern forensic analysis tool for tracking file changes over time.
Baseline Preview
Folder: C:\Users\harsh\OneDrive\Desktop\sample
Saved at: 2025-10-31T01:04:01.499476
name
relpath
C:\Users\harsh\OneDrive\Desktop\sample\csdf1.pdf
csdf1.pdf
C:\Users\harsh\OneDrive\Desktop\sample\csdf1.pdf
C:\Users\harsh\OneDrive\Desktop\sample\DL_MiniProject_report.docx
DL_MiniProject_report.docx
C:\Users\harsh\OneDrive\Desktop\sample\DL_MiniProje
C:\Users\harsh\OneDrive\Desktop\sample\Online_Learning_Process_Tracking_Syste
Online_Learning_Process_Trackin
C:\Users\harsh\OneDrive\Desktop\sample\Online_Learn
C:\Users\harsh\OneDrive\Desktop\sample\Personalized_Learning_Systems_Using_A
Personalized_Learning_Systems_
C:\Users\harsh\OneDrive\Desktop\sample\Personalized_
C:\Users\harsh\OneDrive\Desktop\sample\Picture1.png
Picture1.png
C:\Users\harsh\OneDrive\Desktop\sample\Picture1.png
C:\Users\harsh\OneDrive\Desktop\sample\SharedInput_Form_Report.docx
SharedInput_Form_Report.docx
C:\Users\harsh\OneDrive\Desktop\sample\SharedInput_
C:\Users\harsh\OneDrive\Desktop\sample\untitled-1.pdf
untitled-1.pdf
C:\Users\harsh\OneDrive\Desktop\sample\untitled-1.pdf
Powered by Streamlit | For raw forensic disk images, use pytsk3.


Deploy
Options
Folder path
C:\Users\harsh\OneDrive\Des
Create / Update Baseline
Load Existing Baseline
Scan & Compare with Baseline
Baseline exists (Saved at 2025-10-31T01:04:01.499476)
Folder Forensics Dashboard
A modern forensic analysis tool for tracking file changes over time.
Comparison Summary
Baseline files
7
Current files
7
Added
1
Deleted
1
Added files
name
relpath
size
created
modified
0
FINAL AVISHKAR POSTER.pdf
C:\Users\harsh\OneDrive\Desktop\sample\FINAL AVISHKAR POSTER.pdf
239.16 KB
2025-09-16 15:44:40
2025-09-14 21:55:10
Deleted files
name
relpath
size
created
modified
0
Picture1.png
C:\Users\harsh\OneDrive\Desktop\sample\Picture1.png
63.95 KB
2025-09-18 11:34:39
2025-09-17 17:46:29

Chapter 6 : Features

1. Folder Forensics Analysis

The system allows users to **select a directory or folder** from their system and automatically extracts important forensic details such as:

- File names, extensions, and sizes
- File creation, modification, and access times (timestamps)
- Deleted or missing files (by checking inconsistencies in metadata)
- File metadata and hash information for integrity checking

This provides the user with a **comprehensive overview of digital evidence** within a directory.

2. Hash-Based File Identification

Each analyzed folder or file is assigned a **unique hash value (MD5/SHA)** generated using Python's `hashlib` library.
This ensures that every file is uniquely identified and can be verified later to detect tampering or modification.

3. Metadata Extraction

The project extracts and displays **file metadata** such as:

- File size
- Path
- Access time
- Modification time
- Creation time

This metadata helps investigators understand the timeline of user actions, which is critical in **digital forensics**.

4. Data Visualization

The project uses **Streamlit** and **Pandas** to display collected data in an organized, tabular form.
The graphical user interface makes it easy to interpret large sets of forensic data effectively.

5. Intelligent Change Detection

The system can **detect new, modified, or deleted files** between multiple scans of the same directory.
By comparing stored metadata from previous sessions (saved in JSON format), it can report differences in:

- File additions
- Modifications
- Deletions

This feature helps identify suspicious changes in a directory over time.

6. Forensic Report Generation

All scanned and analyzed data is **saved in structured JSON files**, enabling:

- Easy future analysis
- Evidence preservation
- Compatibility with other forensic tools

This ensures the data can be revisited or exported for legal or investigative purposes.

7. User-Friendly Web Interface

The entire tool is built with **Streamlit**, providing:

- A modern, interactive UI
- Real-time folder analysis
- Easy navigation and clear data presentation

No technical expertise is required — even non-programmers can use it effectively.

8. Integrity Verification

By maintaining hash-based metadata logs, the tool supports **digital evidence integrity verification**. Investigators can verify if any file has been changed since the last scan, ensuring the **authenticity** of evidence.

9. Lightweight and Portable

The tool is completely **Python-based**, requiring no heavy installation or external forensic suite.
It runs efficiently on low-end hardware and across different platforms (Windows/Linux/Mac).

10. Extensible Architecture

The system's modular code structure allows easy integration of advanced forensic features such as:

- File content hashing
- Binary signature matching
- Timeline analysis
- Network forensic extension

This makes the tool adaptable for future digital investigation needs.

Chapter 7 : Future Scope

The current implementation of the *File System Forensics* system provides a strong foundation for analyzing directories, extracting metadata, and detecting changes. However, as technology evolves and cyber threats become more sophisticated, there are several opportunities to enhance the system's capabilities and make it suitable for professional digital forensic investigations.1. Mobile Application Integration

1. Integration with Advanced File Systems

Future versions can be designed to directly interface with complex file systems such as:

- **NTFS (New Technology File System)**
- **EXT4 (Linux Extended File System)**
- **HFS+ / APFS (macOS File Systems)**

By parsing file system structures directly (e.g., Master File Table in NTFS or inodes in EXT4), the tool can recover deeper forensic artifacts such as deleted file fragments, slack space data, and journaling logs.

2. AI-Based Anomaly Detection

Integrating **Artificial Intelligence (AI) and Machine Learning (ML)** models can help automatically identify suspicious activity patterns, such as:

- Unusual file modification behavior
- Abnormal access time patterns
- Hidden or renamed malicious files

AI-driven analysis can significantly assist investigators by providing **automated insights** and prioritizing critical evidence.

3. Automated Forensic Reporting

The future version could generate **comprehensive forensic reports** in PDF or HTML format, containing:

- File metadata summaries
- Detected changes (added, modified, deleted)
- Graphs and charts of user activity timelines
- Hash comparison results

This would enhance usability for law enforcement and legal documentation.

4. Integration with Disk Imaging Tools

By integrating with forensic imaging formats such as **E01 (EnCase)**, **AFF (Advanced Forensic Format)**, or **RAW disk images**, the system could analyze **entire storage devices** instead of just

folders.
This would extend its application to **professional forensic casework**.

5. Cloud and Remote Forensic Analysis

In modern infrastructures, digital evidence may reside in cloud storage (Google Drive, AWS S3, OneDrive).
Future implementations could allow:

- Remote folder scanning
- Cloud metadata extraction
- Access log analysis for shared files

This would make the tool suitable for **cloud forensics** and **remote investigations**.

6. Integration with Other Forensic Tools

The system could be extended to interoperate with popular forensic frameworks such as:

- **Autopsy / Sleuth Kit**
- **Volatility (Memory Analysis)**
- **Wireshark (Network Analysis)**

Such integration would enable **comprehensive forensic analysis** covering file, memory, and network domains.

7. Real-Time Monitoring System

An enhanced version could monitor file system activities **in real-time** using OS-level APIs (like Python's watchdog module), instantly detecting:

- File creation, modification, or deletion events
- Unauthorized file access or transfer

This would make the system suitable for **intrusion detection** and **incident response**.

8. Digital Signature and Chain of Custody

To ensure the legal validity of evidence, the system could implement:

- **Digital signatures** for forensic reports
- **Automated Chain of Custody (CoC)** tracking
  This ensures that the evidence remains authentic and tamper-proof from collection to courtroom presentation.

9. Cross-Platform Desktop and Mobile Versions

By using frameworks like **Electron.js** or **PyInstaller**, a standalone desktop version could be created. Additionally, a lightweight mobile companion app could allow users to **trigger scans and view reports remotely**.

10. Visualization Dashboard

A future upgrade can include an **interactive visualization dashboard** to represent:

- File activity timelines
- Hash comparison graphs
- Deleted file heatmaps

Such visualization would provide investigators with a **clear, intuitive view of forensic evidence**.

Chapter 8 : Conclusion

The **File System Forensics** project successfully demonstrates a practical approach to analyzing and interpreting digital evidence stored within a computer's file system. By focusing on file metadata, access patterns, and integrity verification, the system bridges the gap between raw file storage and meaningful forensic insights.

Through the use of **Python** and **Streamlit**, the project provides an intuitive and interactive interface for investigators, making the forensic process both efficient and accessible. Users can easily scan directories, identify deleted or modified files, analyze timestamps, and generate structured reports — all without relying on complex commercial forensic tools.

The project effectively highlights the importance of **metadata analysis** in digital forensics. File attributes such as creation, modification, and access times serve as critical evidence in understanding user behavior and reconstructing event timelines. Furthermore, by implementing **hash-based verification**, the system ensures the authenticity and integrity of forensic data, which is vital in legal and investigative contexts.

While the current version focuses on local file system analysis, the design leaves room for future enhancements such as AI-based anomaly detection, integration with disk imaging formats, and cloud forensic support. These improvements could evolve the system into a fully-fledged forensic investigation toolkit capable of handling complex digital environments.

In summary, the **File System Forensics** project lays a strong foundation for digital evidence collection and analysis. It demonstrates how modern programming techniques and open-source tools can be combined to create a **lightweight, extensible, and effective forensic solution**, promoting deeper understanding and awareness of cyber forensics among learners and professionals alike.

Chapter 9: References

- **Carrier, B. (2005).** *File System Forensic Analysis.* Addison-Wesley Professional.
– A foundational textbook detailing the structure of file systems (FAT, NTFS, EXT) and forensic analysis techniques.

- **Casey, E. (2011).** *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet* (3rd Edition). Academic Press.
– Explains the principles of digital evidence handling, preservation, and forensic methodologies.

- **Nelson, B., Phillips, A., & Steuart, C. (2018).** *Guide to Computer Forensics and Investigations.* Cengage Learning.
– Discusses various tools and approaches used in modern computer forensics, including file system analysis and evidence recovery.

- **Garfinkel, S. (2013).** *Digital Forensics Research: The Next 10 Years. Digital Investigation*, 10(2), 64-73.
– Research paper highlighting evolving forensic challenges and future research directions in data recovery and analysis.

- **The Sleuth Kit & Autopsy Documentation.** Retrieved from: https://www.sleuthkit.org
– Open-source forensic frameworks providing utilities for file system investigation and metadata recovery.

- **Python Official Documentation.** Retrieved from: https://docs.python.org
– Referenced for modules such as `os`, `hashlib`, `datetime`, and `pathlib` used in building the forensic tool.

- **Streamlit Documentation.** Retrieved from: https://docs.streamlit.io
– Used for developing the interactive forensic user interface.

- **Pandey, M., & Alharbi, F. (2020).** *An Overview of File System Forensics and Metadata Analysis. International Journal of Computer Applications*, 176(30), 1-6.
– A survey discussing metadata-based forensic approaches for identifying file access and modification patterns.

- **Singh, R., & Kumar, S. (2022).** *A Comparative Study of File System Forensics Techniques. Journal of Cyber Security and Information Management*, 9(4), 44-51.
– Compares different forensic tools and methods for FAT, NTFS, and EXT file systems.

- **NIST Computer Forensics Tool Testing Program.** Retrieved from: https://www.nist.gov/itl
– Provides validation standards and methodologies for testing forensic software tools.