EXPERIMENT – 04

AIM:

Test wireless network security using Aircrack-ng or Wireshark; understand WPA2/WPA3 encryption.

DESCRIPTION:

This experiment demonstrates wireless network security analysis using Wireshark and explores the encryption mechanisms of WPA2 and WPA3 protocols. It focuses on understanding how wireless authentication, encryption, and key exchange ensure confidentiality and data protection in Wi-Fi communications.

Using Wireshark, network traffic is captured and analyzed to observe management and data frames, including the 4-way handshake process. The experiment also introduces the purpose of tools like Aircrack-ng for authorized security testing. This exercise enhances understanding of wireless security concepts, encryption standards, and methods to secure Wi-Fi networks against potential threats..

PROCEDURE:

Step 1: Pre-Lab Setup

1. Install and open **Wireshark** with administrative privileges.
2. Optionally install **Aircrack-ng** (only in a legal, controlled lab setup).
3. Connect to a Wi-Fi network that you own or have explicit permission to analyze.

Part A: Capture and Analyze Wireless Traffic Using Wireshark

1. **Launch Wireshark** → Start with admin rights.
2. **Select Wireless Interface** (e.g., Wi-Fi or wlan0).
3. **Start Packet Capture** → Observe real-time Management, Control, and Data frames.
4. **Apply Filters:**
   - wlan.fc.type == 0 → Management frames
   - wlan.fc.type == 2 → Data frames
   - eapol → WPA2/WPA3 handshake packets
5. **Analyze WPA2/WPA3 Encryption Process:**
   - WPA2 uses a **4-way handshake** for authentication and key generation.
   - WPA3 adds **Simultaneous Authentication of Equals (SAE)** for improved protection.
6. **Export and Save Capture** as a .pcapng file.

Part B: Understanding Wireless Encryption and Security Testing with Aircrack-ng (Conceptual)

1. **Aircrack-ng Overview:** Used for analyzing packet captures and testing key strength in authorized environments.
2. **Capture Authentication Handshake:** Use a monitoring-capable adapter to record WPA2 handshakes.
3. **Understand Encryption Concepts:**
   - **WPA2 (AES-CCMP):** 128-bit AES encryption and 4-way handshake.
   - **WPA3 (SAE):** Replaces PSK with password-authenticated key exchange (resists offline attacks).
4. **Analyze Captured Packets:** Inspect EAPOL messages and encryption details.
5. **Defensive Measures:**

- Use **WPA3-Personal** or **WPA2-AES**.
- Disable **WPS**.
- Set **strong, unique passwords**.
- Update **router firmware** regularly.

Step 3: Verification and Reporting

- Confirm presence of handshake and encrypted frames.
- Identify encryption protocol (WPA2/WPA3).
- Document observations and note any security misconfigurations.

OUTPUT ANALYSIS:

Wireshark successfully captured management, control, and data frames. The WPA2 handshake packets were identified using the **EAPOL filter**, showing authentication and key exchange sequences between the client and access point. The encrypted data packets confirmed proper use of AES encryption. WPA3 handshake visualization (conceptually) illustrated SAE's enhanced security against offline attacks.

CONCLUSION:

The experiment demonstrated wireless network security analysis using Wireshark and introduced Aircrack-ng's ethical testing capabilities. The observation of the **WPA2 4-way handshake** and understanding of **WPA3 SAE** authentication clarified how encryption protocols protect Wi-Fi communications. WPA3 provides enhanced resilience to brute-force and dictionary attacks, strengthening overall wireless network security.

EXPERIMENT – 05

AIM:

Run vulnerability scans with OpenVAS or Nessus; identify, prioritize, and mitigate vulnerabilities.

DESCRIPTION:

Vulnerability scanning is a critical process in cybersecurity that identifies weaknesses, misconfigurations, and outdated software in systems, networks, and applications. Tools like **OpenVAS** (Open Vulnerability Assessment System) and **Nessus** automate the detection of security flaws, categorize vulnerabilities based on severity, and suggest mitigation measures.

By analyzing the scan reports, security teams can prioritize risks, apply timely patches, and reduce the attack surface, thereby preventing potential exploitation or data breaches. This experiment demonstrates the complete process of vulnerability scanning, from setup to mitigation.

PROCEDURE:

1. Pre-Scan Setup

- Install and configure **OpenVAS** or **Nessus** on your system.
- Ensure the **target system or network is authorized** for scanning.
- Update the **vulnerability database** to include the latest CVEs and plugins.

2. Create a Scan Policy

- Choose an appropriate scan type:
  - *Full and Fast* (comprehensive system scan)
  - *Web Application Scan* (for web servers)
  - *Network Scan* (for IP range scanning)
- Configure target details such as IP addresses, ports, and credentials if performing authenticated scans.

3. Run the Scan

- Start the vulnerability scan on the selected targets.
- Monitor scan progress and system performance to ensure stability.
- Wait until the scan completes and results are compiled.

4. Review Results

- Access the **scan report** generated by OpenVAS or Nessus.
- Identify vulnerabilities and their **severity ratings** (Critical, High, Medium, Low).
- Note down vulnerable hosts, open ports, and affected software versions.

5. Prioritize Vulnerabilities

- Use **risk-based prioritization** considering:
  - **CVSS (Common Vulnerability Scoring System)** scores.
  - **Business impact** of affected assets.
  - **Exploitability** of the vulnerability.
- Address **Critical** and **High** vulnerabilities first.

6. Mitigation & Remediation

- Apply necessary **security patches or software updates**.
- Reconfigure insecure settings (e.g., weak passwords, open ports).
- Deploy **firewalls**, **IDS/IPS systems**, or **endpoint protection** as needed.

7. Re-Scan

- After remediation, **rerun the scan** to ensure vulnerabilities are resolved.
- Verify that no high-risk vulnerabilities remain open.

OUTPUT ANALYSIS:

The scan report provided detailed insight into multiple vulnerabilities across the network. Examples included outdated software versions, exposed ports, and weak SSL configurations. Using CVSS scores, vulnerabilities were prioritized — critical issues such as open remote desktop ports and unpatched web servers were remediated first. Subsequent rescans confirmed that mitigation efforts effectively reduced the vulnerability count and improved the overall security posture.

CONCLUSION:

Vulnerability scanning using **OpenVAS** or **Nessus** is a vital step in maintaining network and system security. It provides visibility into potential attack surfaces, helps prioritize risks using CVSS metrics, and guides administrators in implementing effective countermeasures. Regular vulnerability assessments, followed by timely remediation, strengthen the organization's defense and ensure compliance with cybersecurity best practices.

EXPERIMENT – 06

AIM:

Establish a secure VPN with OpenVPN and analyze potential vulnerabilities using Wireshark.

DESCRIPTION:

A **Virtual Private Network (VPN)** provides a secure communication channel over untrusted networks by encrypting data to ensure confidentiality and integrity. **OpenVPN**, an open-source VPN tool, uses **SSL/TLS** for secure connections between clients and servers.
After establishing the VPN tunnel, **Wireshark** can be used to capture and analyze network traffic to verify that data remains encrypted and to detect possible vulnerabilities such as weak ciphers, misconfigurations, or data leaks.
This experiment demonstrates the creation of a secure VPN setup and validates encryption using traffic analysis, highlighting best practices for maintaining VPN security.

PROCEDURE:

1. OpenVPN Setup

- Install **OpenVPN** on both client and server machines.
- Use **EasyRSA** to generate server and client certificates and keys.
- Configure the server using server.conf (defining IP range, port, and protocol such as UDP/TCP).
- Create the **client configuration file** (client.ovpn) including server details and client certificates.
- Start the OpenVPN server and connect the client.

2. Test VPN Connectivity

- Verify the VPN connection by **pinging the server** from the client.
- Check assigned **VPN IP addresses** and **routing tables** to confirm tunnel establishment.

3. Capture Traffic with Wireshark

- Launch **Wireshark** on either the client or server system.
- Start capturing packets on the network interface used by OpenVPN.
- Apply the filter ip.addr == <VPN_Server_IP> to focus on VPN traffic.

4. Analyze Potential Vulnerabilities

- Confirm that the captured packets are **TLS/SSL-encrypted**.
- Ensure no plaintext credentials or sensitive data are visible.
- Identify any **anomalies**, **misconfigurations**, or **unexpected traffic patterns** in the VPN communication.

5. Mitigation

- Strengthen encryption settings (e.g., **AES-256**, **TLS 1.2/1.3**).
- Disable unused or insecure protocols and ports.
- Enforce strict **certificate validation** and **key management** policies.

OUTPUT ANALYSIS:

- **VPN Connection Verification:** Client successfully connected; VPN IP address assigned and reachable through ping.
- **Packet Capture:** Wireshark showed **encrypted TLS packets** between client and server, with no plaintext or sensitive data exposure.
- **Traffic Patterns:** Only expected OpenVPN port and protocol were active; no suspicious activity observed.
- **Encryption Check:** All packets displayed encrypted payloads, confirming correct VPN encryption configuration.
- **Vulnerabilities:** No major issues detected; improper cipher configurations or invalid certificates would have appeared as handshake errors or unencrypted traffic.

CONCLUSION:

The experiment demonstrated the establishment of a **secure VPN tunnel using OpenVPN**, verified through **Wireshark traffic analysis**. The captured traffic confirmed that all communication was encrypted and protected from interception. The analysis reinforced the importance of proper configuration, strong encryption, and regular monitoring to maintain VPN integrity and prevent data leaks.
A correctly configured OpenVPN ensures safe, private communication over untrusted networks, and Wireshark serves as a powerful validation and diagnostic tool for maintaining security assurance.

EXPERIMENT – 08

AIM:

Analyze malware in Cuckoo Sandbox, using IDA Pro or Ghidra for reverse engineering.

DESCRIPTION:

Malware analysis combines dynamic sandboxing and static reverse engineering to determine what a suspicious binary does, how it persists, and which indicators it produces. Use **Cuckoo Sandbox** for automated dynamic behavioral analysis (process activity, network IOCs, dropped files) and **IDA Pro** or **Ghidra** for static disassembly/decompilation and code-path recovery. Always perform analysis in an isolated, air-gapped lab environment with snapshots and explicit authorization.

PROCEDURE:

1. Lab & Safety Setup (Pre-analysis)

- Prepare an isolated VM template (Windows or Linux as appropriate). Disable shared folders and clipboard, and use an instrumented NAT or host-only network with filtering.
- Take a clean snapshot of the analysis VM. Create a network sink (e.g., INetSim) or dummy DNS/HTTP endpoints.
- Install and configure **Cuckoo Sandbox** on the analysis host and register the analysis VM (install the Cuckoo agent on the guest).
- Ensure availability of analysis utilities: process tracing (procmon-like), Sysinternals, strings, peid/exiftool, and **IDA Pro** or **Ghidra**.

2. Dynamic Analysis with Cuckoo Sandbox

- Submit the sample to Cuckoo (via web UI or API). Set analysis options: timeout, package type (generic/office/android), enable network capture, enable memory dump.
- Monitor the run: watch process tree, API calls, file system and registry changes, network traffic (PCAP), mutexes, service activity, and persistence attempts.
- After completion, collect the report (JSON + HTML) and export artifacts: dropped files, memory dump, PCAPs, screenshots, and behavioral logs.

3. Triage & Quick Indicators

- Compute hashes (MD5, SHA256), run YARA rules, and submit samples to VirusTotal or private sandbox (if policy allows).
- Extract strings, imports, version info, and cryptographic constants to quickly find URLs, domains, IPs, or C2 clues.

4. Static Reverse Engineering (IDA Pro / Ghidra)

- Load the original binary and any dropped artifacts into IDA/Ghidra; let auto-analysis run.
- Identify entry point, unpacking routines, and anti-analysis checks (VM/sandbox detection). If the binary is packed, use the memory dump from Cuckoo or manual unpacking techniques.
- Analyze imports and dynamically resolved APIs (networking, process injection, crypto, persistence). Cross-reference strings and XREFs to uncover command handling, URLs, or protocol details.
- Use decompiler output and control-flow analysis to recover protocol handlers, encryption routines, and payload logic. Focus on functions implementing persistence, C2 communication, credential theft, lateral movement, and data exfiltration.
- Document function names, offsets, and add meaningful comments. Emulate or script suspected routines where safe (do not execute malware on host).

5. Correlation & Classification

- Map observed behaviors to **MITRE ATT&CK** techniques (for example, T1059, T1071, T1543).
- Prioritize IOCs and behaviors based on impact: persistence + credential theft + active C2 >> simple info leak.

6. Reporting & Remediation Guidance

- Produce an executive summary, technical findings, YARA rules (if safe), IOCs (hashes, domains, IPs, filenames, registry keys), and recommended mitigations (patching, domain/IP blocking, credential resets).
- Revert the analysis VM to the clean snapshot and store artifacts in secure, access-controlled storage.

OUTPUT ANALYSIS (Cuckoo + IDA/Ghidra):

- **Behavior:** Sample creates persistence mechanisms, spawns processes, performs process injection (e.g., into explorer.exe), and communicates with C2 domains.
- **Artifacts:** Dropped files, created registry keys, new services or scheduled tasks, and spawned processes.
- **Network:** Observed domains, IPs, URLs, DNS queries, and TLS-encrypted C2 traffic; beaconing frequency noted.
- **Memory/Static:** Injected modules discovered in memory dumps, decrypted strings exposed at runtime, suspicious functions and crypto routines, evidence of packing/obfuscation.
- **IOCs:** Provide SHA256/MD5 hashes, C2 domains/IPs, dropper filenames, registry keys—classified by criticality (critical / medium / low).
- **MITRE & Risk Mapping:** Techniques observed (e.g., persistence T1543, command-and-control T1071) and risk ranking based on impact and exploitability.
- **Mitigation:** Immediate host isolation, blocklisted C2 domains/IPs at perimeter, revoke potentially compromised credentials, apply security patches, enable/strengthen EDR and monitoring, and schedule rescans.

CONCLUSION:

Combining **Cuckoo Sandbox** dynamic analysis with static reverse engineering in **IDA Pro** or **Ghidra** provides a comprehensive understanding of malware behavior, implementation, and IOCs. Dynamic analysis reveals runtime behavior and network activity while static analysis exposes code-level implementation details and protocol logic. Together they enable accurate detection, containment, and remediation recommendations. Always perform malware analysis in an isolated, snapshot-backed environment and document findings clearly for both technical teams and stakeholders.

EXPERIMENT – 10

AIM:

To perform logical imaging of a USB drive using FTK Imager and recover deleted files from it.

DESCRIPTION:

This experiment demonstrates how a digital forensic examiner acquires and analyzes data from a USB drive without altering the original evidence.
Using **FTK Imager**, a logical image of a USB drive containing sample files (some intentionally deleted) is created.
The tool computes hash values (MD5/SHA1) to ensure forensic integrity.
The image is then examined in FTK Imager to locate and recover deleted files. The recovered files are exported and verified for data accuracy.
This process replicates a real-world forensic workflow for evidence acquisition, preservation, and recovery from removable media.

REQUIREMENTS:

- A Windows PC with **FTK Imager** installed
- A **USB drive** containing test files (some should be deleted beforehand)
- A **secondary storage location** (e.g., external HDD or local folder) to store the forensic image
- **Administrator privileges** on the system
- **FTK Imager download link:** https://accessdata.com/product-download/ftk-imager

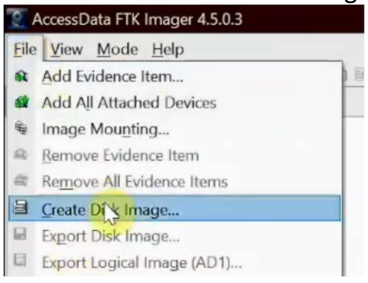
PROCEDURE:

Step 1: Launch FTK Imager

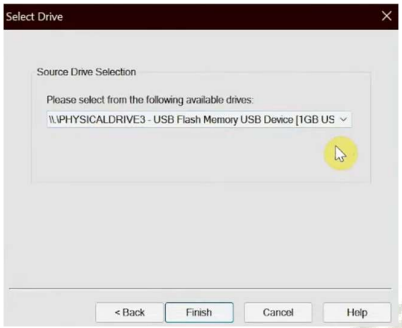
1. Open FTK Imager with administrative rights.
2. Prepare a folder or USB with test files, then delete a few files from it.
3. You'll attempt to recover those deleted files from the image later.

Step 2: Create a Logical Image

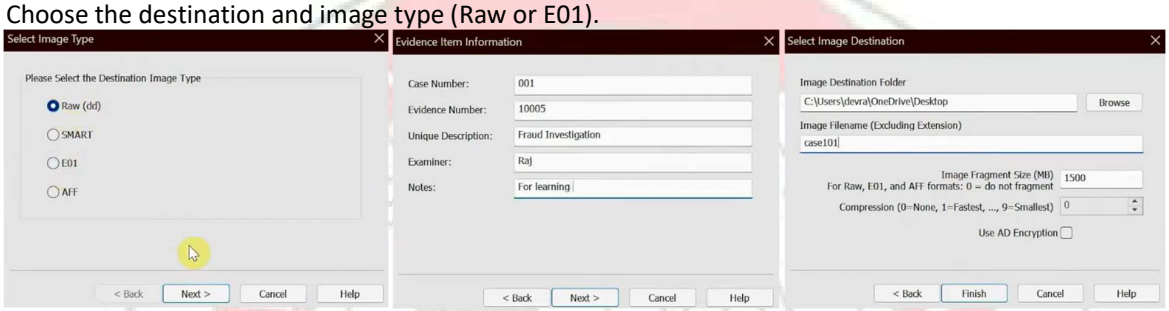
1. Go to 'File' → 'Create Disk Image'.



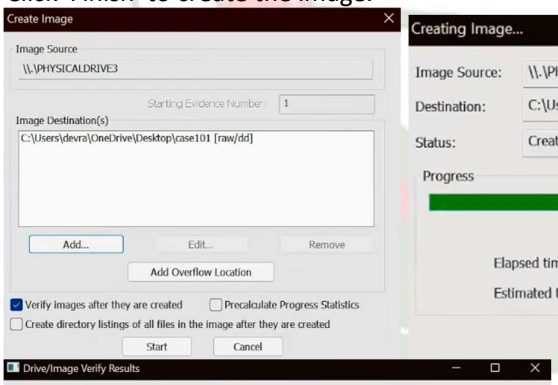
2. Select 'Contents of a Folder'.
3. Browse and choose the folder or USB drive.

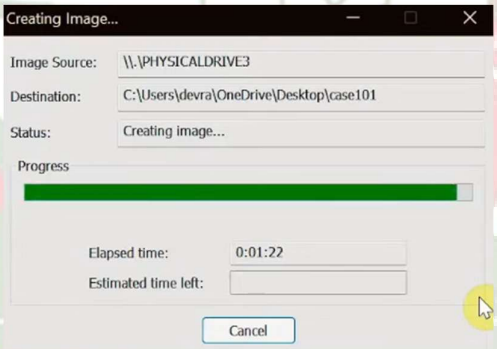


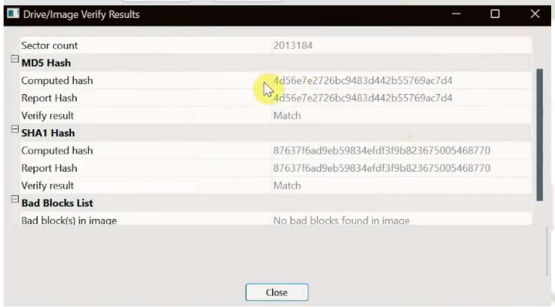
4. Choose the destination and image type (Raw or E01).



5. Enter optional case metadata.
6. Click 'Finish' to create the image.

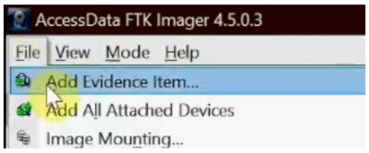




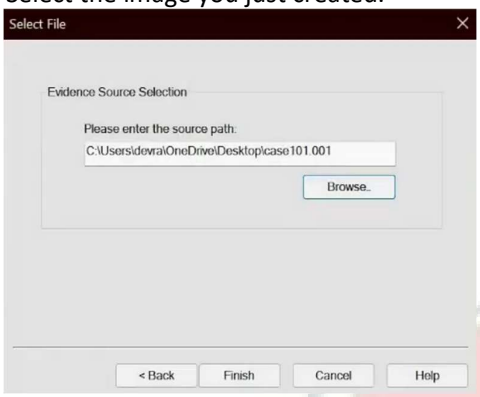


Step 3: View and Analyze Image Contents

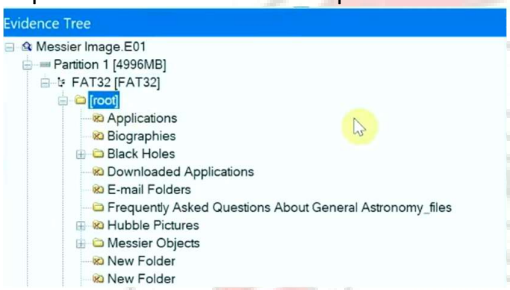
1. Go to 'File' → 'Add Evidence Item'.



2. Select the image you just created.



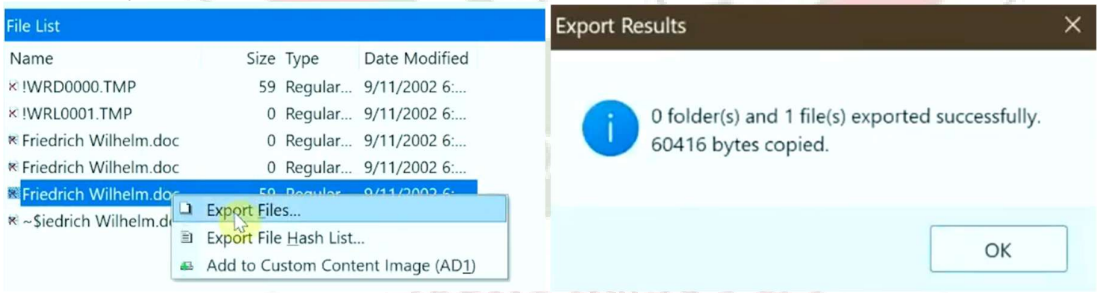
3. Expand the folders in the left pane.



4. Look for deleted files (shown in red or marked as deleted).

Step 4: Export Recovered Files

1. Right-click on any deleted file(s).
2. Choose 'Export Files' and save them to a secure folder.



3. Verify if file content is intact and usable.

OUTPUT ANALYSIS

- A **forensic image** of the USB drive (e.g., USB_Logical.E01) was successfully created.
- FTK Imager displayed **deleted files in red** within the directory tree.
- Deleted files were **successfully recovered** and could be opened without corruption.
- **Hash verification** confirmed the image's integrity — no alterations occurred during the process.

CONCLUSION:

The experiment demonstrates that **FTK Imager** is a reliable tool for performing **logical imaging** and **deleted file recovery** from USB drives. The process maintains forensic soundness by generating hash values that validate image authenticity.

All deleted files that had not been overwritten were successfully recovered, showcasing the potential of forensic tools to restore evidence even after user deletion.

Through this experiment, the importance of imaging (instead of direct analysis) and hash verification in maintaining the **chain of custody** and **data integrity** was reinforced.

This exercise effectively simulates how digital forensic investigators collect, preserve, and analyze evidence from removable storage devices in real-world investigations.

EXPERIMENT – 11

AIM:

To Authenticate Multimedia Files Using Metadata Analysis (EXIFTool)

DESCRIPTION:

This experiment demonstrates the theoretical and practical aspects of using **ExifTool**—a command-line metadata extraction utility developed by Phil Harvey—to perform digital forensic analysis on multimedia files. The focus is on:

- Extracting and analyzing EXIF, IPTC, and XMP metadata.
- Identifying device and software information used to capture or modify media.
- Editing metadata fields to simulate tampering.
- Comparing original and modified files using hash values and metadata reports to detect manipulation.

By using ExifTool on Windows 10/11, students can understand how metadata supports digital evidence authentication in forensic investigations.

PROCEDURE:

Part A: Installation and Basic Metadata Extraction

Procedure

1. Download **ExifTool** from the official website: https://exiftool.org.
2. Extract the downloaded ZIP file and rename exiftool(-k).exe to **exiftool.exe**.
3. Create a new folder named **EXIFTOOL EXP** inside your working directory (C:\SEM VII\CS LAB\).
4. Place at least **five sample image files** (e.g., 1.jpg, 2.jpg, 3.jpg, 4.jpg, 5.jpg) along with:
   - exiftool.exe
   - exiftool_files\ folder
   - README.html file
5. Open **Windows PowerShell or Command Prompt** and navigate to the folder:
   cd "C:\SEM VII\CS LAB\EXIFTOOL EXP\"
6. Run the following command to extract metadata from an image file: .\exiftool.exe 1.jpg

```
PS C:\SEM VII\CS LAB\EXIFTOOL EXP> .\exiftool.exe 1.jpg
ExifTool Version Number         : 13.39
File Name                       : 1.jpg
Directory                       : .
File Size                       : 212 kB
Zone Identifier                 : Exists
File Modification Date/Time     : 2025:10:24 19:03:36+05:30
File Access Date/Time           : 2025:10:24 19:03:59+05:30
File Creation Date/Time         : 2025:10:24 19:03:35+05:30
File Permissions                : -rw-rw-rw-
```

   (You can replace "1.jpg" with any other file name or extension.)
7. To extract metadata from all images in bulk: .\exiftool.exe *.jpg
8. To save metadata output to a text report: .\exiftool.exe 1.jpg > metadata_1.txt

Part B: Metadata Editing and Tampering Demonstration

Removing Metadata

Removing all metadata is essential for privacy or resetting image data.

- **For a single file:** .\exiftool.exe -all= -overwrite_original file.ext

```
PS C:\SEM VII\CS LAB\EXIFTOOL EXP> .\exiftool.exe -all= -overwrite_original 1.jpg
Warning: ICC_Profile deleted. Image colors may be affected - 1.jpg
    1 image files updated
```

- **For multiple files:** .\exiftool.exe -all= -overwrite_original *.ext

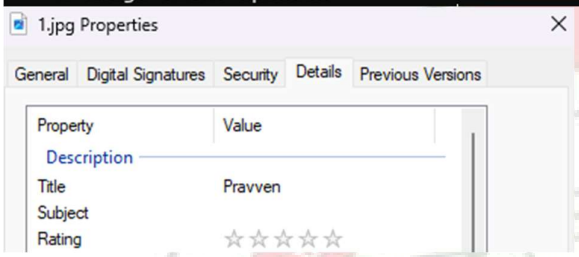
Changing Metadata

Editing metadata allows altering fields such as title, subject, author, and creation date for testing or SEO optimization.

Single File Metadata Editing

.\exiftool.exe -Title="New Title" file.ext

```
PS C:\SEM VII\CS LAB\EXIFTOOL EXP> .\exiftool.exe -Title="Pravven" 1.jpg
    1 image files updated
```

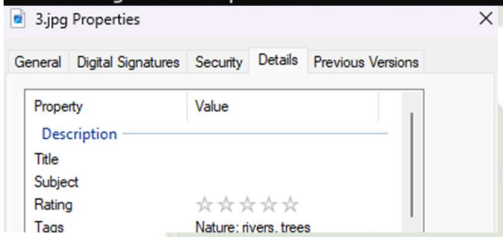


.\exiftool.exe -Subject="New Subject" file.ext

```
PS C:\SEM VII\CS LAB\EXIFTOOL EXP> .\exiftool.exe -Subject="Nature" 3.jpg
    1 image files updated
```

.\exiftool.exe -Keywords="keyword1, keyword2" file.ext

```
PS C:\SEM VII\CS LAB\EXIFTOOL EXP> .\exiftool.exe -Keywords="rivers, trees" 3.jpg
    1 image files updated
```



Bulk Metadata Editing

.\exiftool.exe -Title="New Title" *.ext
.\exiftool.exe -Subject="New Subject" *.ext
.\exiftool.exe -Keywords="keyword1, keyword2" *.ext

Common Metadata Fields and Example Commands

Metadata Field	Description	Example Command
Title	Title of the file	.\exiftool.exe -Title="New Title" *.ext
Subject	Short description of the file	.\exiftool.exe -Subject="New Subject" *.ext
Keywords	Tags or keywords for search	.\exiftool.exe -Keywords="keyword1, keyword2" *.ext
Copyright	Copyright details	.\exiftool.exe -Copyright="Copyright Owner" *.ext
Create Date	Creation date of file	.\exiftool.exe -CreateDate="2024:02:15 10:00:00" *.ext
Modify Date	Date when file was last modified	.\exiftool.exe -ModifyDate="2024:02:16 10:00:00" *.ext
GPS Latitude +	Embeds location	.\exiftool.exe -GPSLatitude="40.7128" -GPSLongitude="-

Longitude	coordinates	74.0060" *.ext
Camera Model	Device used to capture the image	.\exiftool.exe -Model="Nikon D850" *.ext
Artist	Creator or photographer	.\exiftool.exe -Artist="John Doe" *.ext
Album	Album title for audio files	.\exiftool.exe -Album="Summer Hits" *.ext
Author	Author or creator of document	.\exiftool.exe -Author="Jane Smith" *.ext

Hash Value Comparison
To verify integrity and detect tampering, compute SHA-256 hash values **before and after modification** using PowerShell:
Get-FileHash photo_original.jpg -Algorithm SHA256
Get-FileHash photo_edit.jpg -Algorithm SHA256
Compare metadata reports and hash results using **WinMerge** or **Notepad++ Compare Plugin**.

OUTPUT ANALYSIS

- Metadata extraction displays all EXIF, IPTC, and XMP fields such as camera make, model, date, software used, and GPS data.
- After editing, modified tags (Title, Artist, Date, etc.) differ from the original output, confirming metadata manipulation.
- Hash mismatches verify that file integrity has been altered.

CONCLUSION:

The experiment successfully demonstrated metadata extraction, editing, and tampering detection using ExifTool.
Through controlled modifications and hash comparisons, it was observed that even minor metadata changes can alter the file integrity hash, confirming the importance of metadata authentication in digital forensics.

EXPERIMENT – 12

AIM:

To write a Python program using the Scapy library that captures network packets from the interface 'Ethernet/Wifi', analyzes TCP, UDP, and ICMP protocols, and saves the captured packets into a file.

DESCRIPTION:

Packet sniffing is a technique used to monitor and capture network traffic passing through an interface. It allows the observation of data exchange between devices for analysis, debugging, and network security monitoring.

Scapy is a powerful Python-based interactive packet manipulation tool that can capture, dissect, forge, and save network packets. It provides detailed insights into network layers and supports protocols such as **Ethernet, IP, TCP, UDP, and ICMP**.
This experiment demonstrates how to use Python and Scapy to capture and analyze network packets in real-time, extract protocol details, and store them for further analysis using tools like **Wireshark**.

PROCEDURE:

Step 1: Environment Setup

1. Install Python 3.
    - Download from: https://www.python.org/downloads
2. Install Scapy: >>> pip install scapy
3. Install **Npcap** (for Windows):
    - Download from https://nmap.org/npcap/
    - During installation, enable:
        - "Install Npcap in WinPcap API-compatible mode"
        - "Support loopback traffic (optional)"

Step 2: Verify Installation
Open PowerShell (Admin) and run:
>>> Get-Service -Name npcap
If the service is Running, Scapy can perform packet sniffing successfully.

```
PS C:\SEM VII\CS LAB\Packet Sniffer> Get-Service -Name npcap

Status   Name               DisplayName
------   ----               -----------
Running  npcap              Npcap Packet Driver (NPCAP)
```

Step 3: Python Script Implementation- sniffer.py

```
from scapy.all import sniff, wrpcap, Ether, IP, TCP, UDP, ICMP
from datetime import datetime

def packet_callback(pkt):
    print("\n=== Packet Captured ===")
    print("Timestamp:", datetime.now())

    # Ethernet layer
```

```
    if Ether in pkt:
        print("Source MAC:", pkt[Ether].src)
        print("Destination MAC:", pkt[Ether].dst)

    # IP layer
    if IP in pkt:
        print("Source IP:", pkt[IP].src)
        print("Destination IP:", pkt[IP].dst)
        proto = pkt[IP].proto

        # TCP packets
        if proto == 6 and TCP in pkt:
            print("Protocol: TCP")
            print("Source Port:", pkt[TCP].sport)
            print("Destination Port:", pkt[TCP].dport)
        # UDP packets
        elif proto == 17 and UDP in pkt:
            print("Protocol: UDP")
            print("Source Port:", pkt[UDP].sport)
            print("Destination Port:", pkt[UDP].dport)
        # ICMP packets
        elif proto == 1 and ICMP in pkt:
            print("Protocol: ICMP")
            print("Type:", pkt[ICMP].type)
            print("Code:", pkt[ICMP].code)

# Change iface to “Wi-Fi” or “Ethernet” based on your network
packets = sniff(iface="Wi-Fi", count=10, prn=packet_callback)
wrpcap("sniffer_output.pcap", packets)
print("\nAll packets saved to sniffer_output.pcap")
```

Step 4: Execute the Program

Run the script in terminal: python sniffer.py

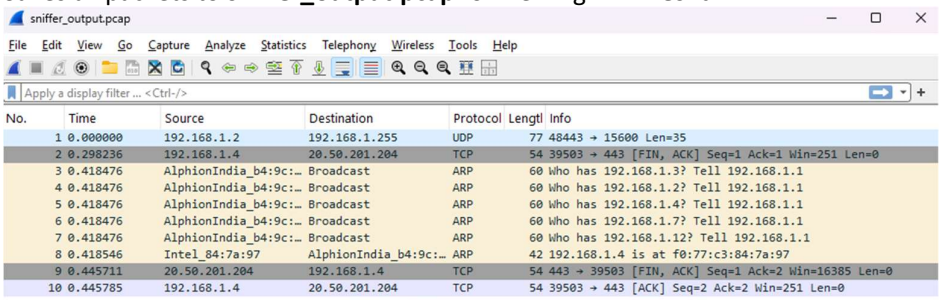
- The program captures 10 packets from the specified network interface.

```
=== Packet Captured ===
Timestamp: 2025-10-25 17:45:59.981251
Source MAC: 1c:af:4a:0d:57:92
Destination MAC: ff:ff:ff:ff:ff:ff
Source IP: 192.168.1.2
Destination IP: 192.168.1.255
Protocol: UDP
Source Port: 48443
Destination Port: 15600

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.286823
Source MAC: f0:77:c3:84:7a:97
Destination MAC: 90:67:17:b4:9c:e3
Source IP: 192.168.1.4
Destination IP: 20.50.201.204
Protocol: TCP
Source Port: 39503
Destination Port: 443

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.399651
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: ff:ff:ff:ff:ff:ff
```

```
=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.399651
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: ff:ff:ff:ff:ff:ff

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.405026
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: ff:ff:ff:ff:ff:ff

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.412597
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: ff:ff:ff:ff:ff:ff

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.416874
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: ff:ff:ff:ff:ff:ff

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.423054
Source MAC: f0:77:c3:84:7a:97
Destination MAC: 90:67:17:b4:9c:e3
```

```
=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.430408
Source MAC: 90:67:17:b4:9c:e3
Destination MAC: f0:77:c3:84:7a:97
Source IP: 20.50.201.204
Destination IP: 192.168.1.4
Protocol: TCP
Source Port: 443
Destination Port: 39503

=== Packet Captured ===
Timestamp: 2025-10-25 17:46:00.439480
Source MAC: f0:77:c3:84:7a:97
Destination MAC: 90:67:17:b4:9c:e3
Source IP: 192.168.1.4
Destination IP: 20.50.201.204
Protocol: TCP
Source Port: 39503
Destination Port: 443

All packets saved to sniffer_output.pcap
PS C:\SEM VII\CS LAB\Packet Sniffer>
```

- Displays protocol details on screen.
- Saves all packets to **sniffer_output.pcap** for viewing in **Wireshark**.



OUTPUT ANALYSIS

After executing the Python Scapy packet sniffer script on the **Wi-Fi/Ethernet interface**, the system successfully captured **10 network packets**, which were stored in the file **sniffer_output.pcap**. The captured data was later opened in **Wireshark** for verification and analysis.

Summary of Captured Packets
The captured packets include multiple network protocol types such as **UDP**, **TCP**, and **ARP**.
Each packet represents communication between devices on the same local network (192.168.1.x) or external internet destinations (e.g., 20.50.201.204).

No.	Source IP / MAC	Destination IP / MAC	Protocol	Description
1	192.168.1.2	192.168.1.255	UDP	Broadcast packet sent to all hosts in LAN.
2	192.168.1.4	20.50.201.204	TCP	Outgoing TCP connection to external web service (Port 443 – HTTPS).
3–7	AlphionIndia_b4:9c:...	Broadcast	ARP	ARP requests like “Who has 192.168.1.x? Tell 192.168.1.1”. Used for IP–MAC resolution in local LAN.
8	Intel_84:7a:97	AlphionIndia_b4:9c:...	ARP	ARP reply confirming MAC for the requested IP.
9	20.50.201.204	192.168.1.4	TCP	Incoming TCP acknowledgment from remote web server.
10	192.168.1.4	20.50.201.204	TCP	TCP FIN/ACK packet — connection termination.

CONCLUSION:

The experiment successfully demonstrated packet sniffing using Python’s Scapy library on both Wi-Fi and Ethernet interfaces.
It captured and analyzed TCP, UDP, and ARP packets in real time, displaying key network details.
All captured packets were saved in a .pcap file and verified through Wireshark.
This experiment enhanced understanding of network communication and protocol behavior.
Overall, it proved Scapy’s effectiveness for network monitoring and cybersecurity analysis.

EXPERIMENT – 13

AIM:

To analyze and retrieve digital evidence such as web browser history and email artifacts from a forensic image using the **Autopsy** digital forensics tool.

DESCRIPTION:

Autopsy is a powerful open-source digital forensics platform used to investigate disk images and recover deleted, hidden, or modified data. It provides modules for analyzing browser artifacts (like history, cookies, bookmarks) and email archives from evidence drives.
This experiment focuses on:

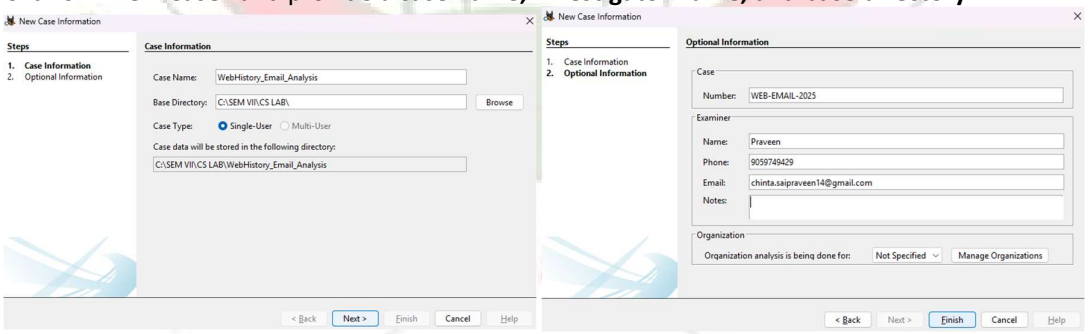
- Extracting and analyzing **web browser history**, cache, and cookies.
- Retrieving **email artifacts** to identify communication patterns, sender/receiver details, and timestamps.

By analyzing these data sources, investigators can reconstruct user activity, online behavior, and potential malicious communication.

PROCEDURE:

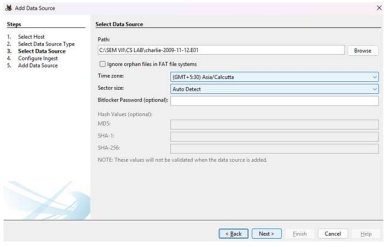
Step 1: Launch Autopsy

- Download **Autopsy** from the official website: https://www.autopsy.com/download/
- Open **Autopsy**.
- Click on **"New Case"** and provide a **case name, investigator name, and base directory**.



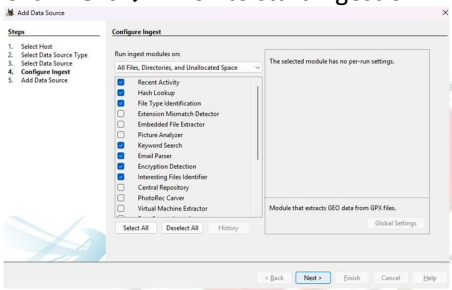
Step 2: Add Data Source

- Choose **"Add Data Source"** and select the evidence type:
  - *Disk Image or VM File* if you have a forensic image (.E01, .dd, .img).
  - *Local Drive* if analyzing a connected disk.
- Browse and select the forensic image.



Step 3: Configure Ingest Modules

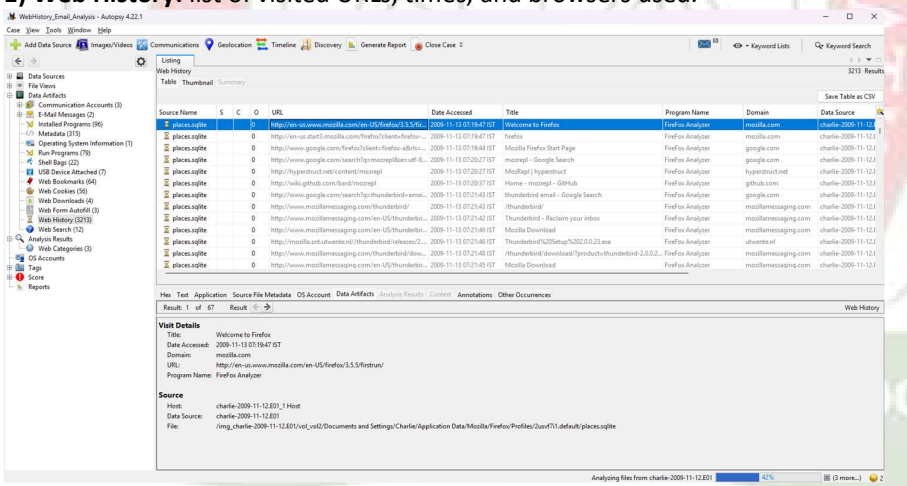
- Enable modules such as:
  - **Web Artifacts** (for browser history, cookies, downloads)
  - **Email Parser** (for email databases)
  - **Recent Activity** (for system activity timeline)
  - **Keyword Search** (for specific queries like domain names or email addresses)
- Click **Next → Finish** to start ingestion.



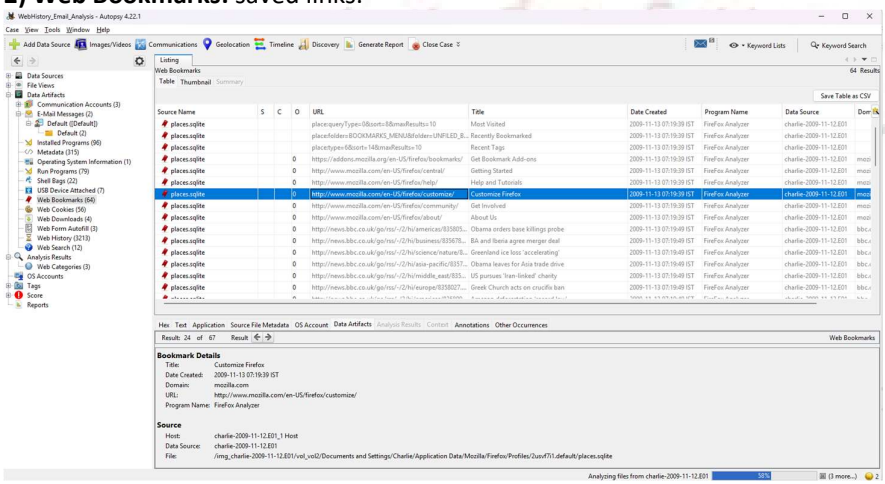
Step 4: Web Browser History Analysis

- Navigate to: **Data Sources → Extracted Content → Data Artifacts**
- Click on **Data Artifacts → Web Artifacts**
- You'll find:

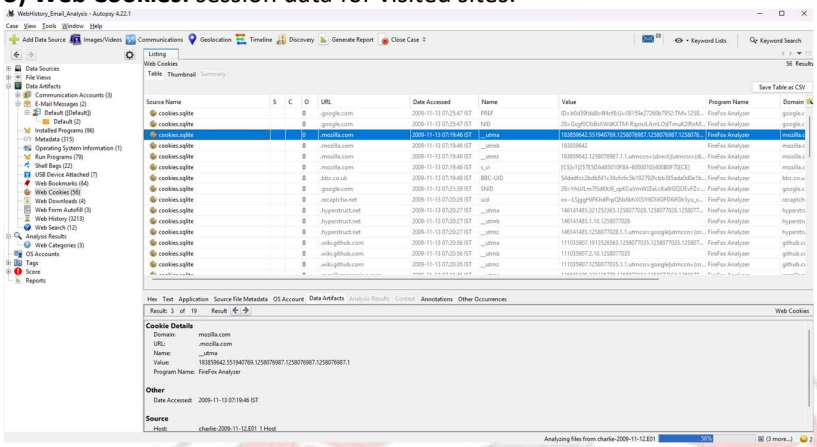
1) Web History: list of visited URLs, times, and browsers used.



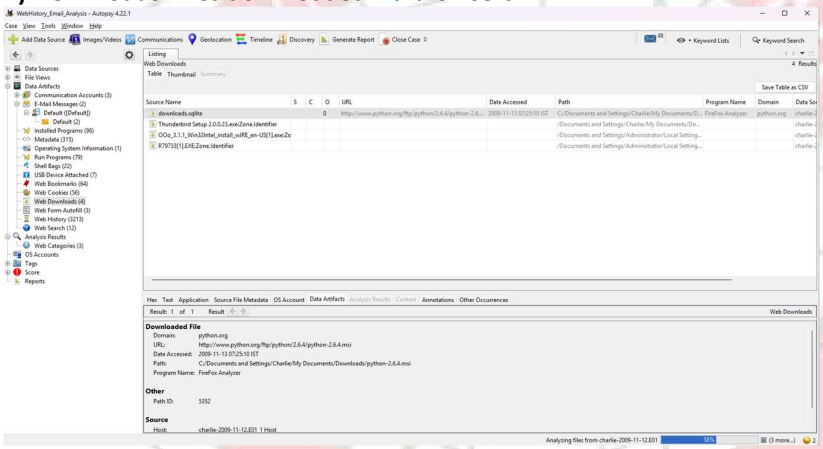
2) Web Bookmarks: saved links.



3) Web Cookies: session data for visited sites.

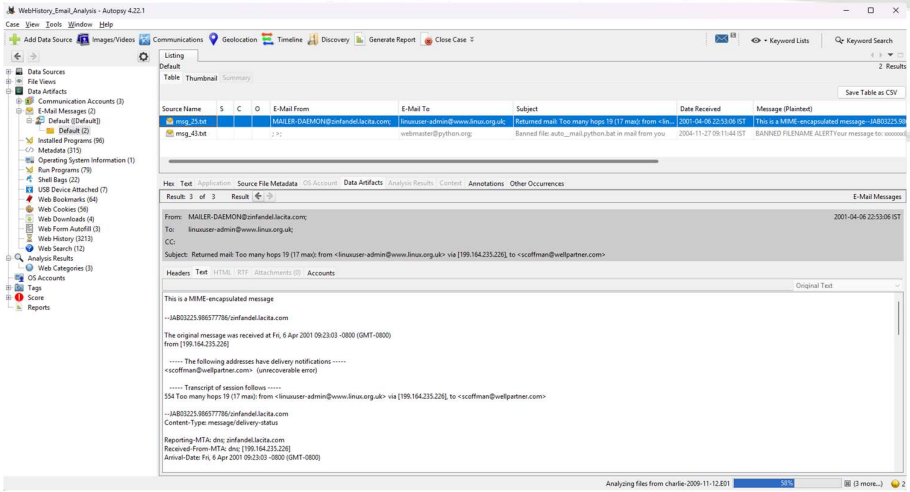


4) Downloads: files downloaded via browsers.



Step 5: Email Analysis

- Go to: **Data Artifacts → Email Messages or Analysis Results → Email Parser**
- View details such as:
  - Sender and recipient addresses
  - Subject lines
  - Message timestamps
  - Attachments



Step 6: Generate Report

- After completing the analysis, go to **"Generate Report"**.
- Choose report formats (HTML, CSV, Excel, or PDF).
- Export the findings for documentation or presentation.

OUTPUT ANALYSIS

Autopsy successfully extracted digital artifacts from the forensic image.
The Web Artifacts module revealed detailed browser activity, including visited URLs, bookmarks, cookies, and downloaded files from Mozilla Firefox. The user frequently accessed sites like *mozilla.com*, *google.com*, and *github.com*, and downloaded files such as the *Python 2.4.4 installer* and *Thunderbird setup*.

The Email Parser module recovered email messages showing sender and recipient details, timestamps, and subjects, confirming active use of an email client.

Overall, the analysis proved that Autopsy effectively retrieves browsing history, login cookies, download records, and email communications, providing a complete overview of user activity for forensic investigation.

CONCLUSION:

Using **Autopsy**, it is possible to successfully retrieve and analyze web browser history and email data from digital evidence.
This analysis helps:

- Establish user intent and timeline of activities.
- Correlate communication records with internet usage.
- Identify potential evidence in cybercrime or digital misconduct cases.

Autopsy's integrated modules and reporting capabilities make it a powerful tool for digital forensic investigations involving browser and email data recovery.

EXPERIMENT – 14

AIM:

To hide secret messages inside digital images using **OpenStego** and later extract them, demonstrating how steganography ensures secure and invisible data transmission.

DESCRIPTION:

OpenStego is an open-source steganography tool used for hiding confidential messages or files within image files without noticeably changing the image.
It uses **Least Significant Bit (LSB)** substitution techniques to embed the data into the pixels of an image (cover file).

The goal is to transmit sensitive information secretly over a public medium such as email or web uploads without arousing suspicion.
Unlike cryptography, which hides the *content*, steganography hides the *existence* of the message.

Key Features of OpenStego:

- Simple GUI interface (no coding required).
- Supports both *data hiding* and *watermarking* modes.
- Uses strong encryption for additional security (optional).
- Supports BMP and PNG image formats.

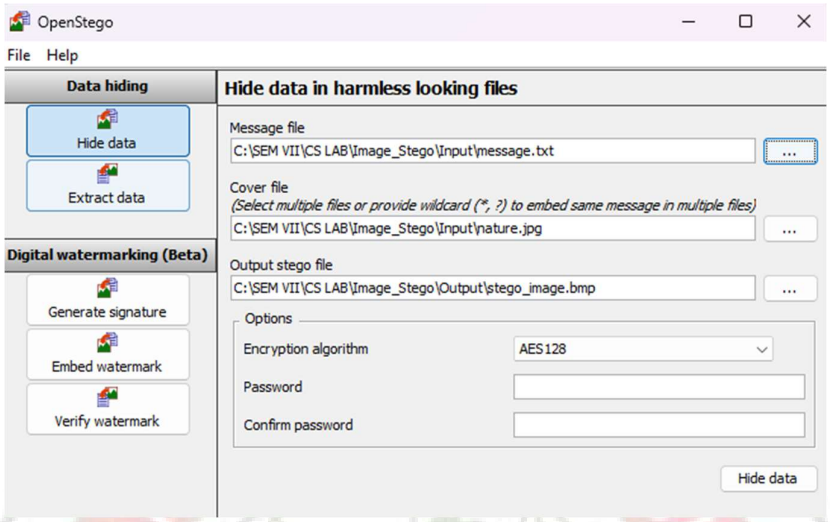
PROCEDURE:

1. Requirements:

- Software: **OpenStego** (download from https://www.openstego.com)
- Operating System: Windows / Linux
- Files required:
  - A **cover image** (e.g., nature.jpg)
  - A **secret message file** (e.g., message.txt)

2. Steps for Hiding (Embedding) a Message:

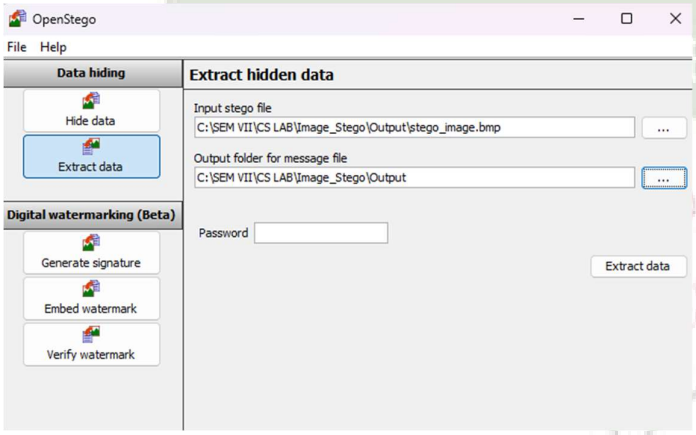
1. **Install and Launch OpenStego**
   Open the application after downloading and installing it.
2. **Select Operation Mode**
   Choose **"Data Hiding"** mode on the main interface.
3. **Select Files**
   - **Message File:** Select the text file (message.txt) you want to hide.
   - **Cover File:** Choose the image file (nature.jpg) in which the message will be hidden.
   - **Output File:** Specify the name and location of the output file (e.g., stego_image.png).

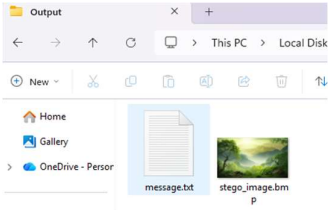


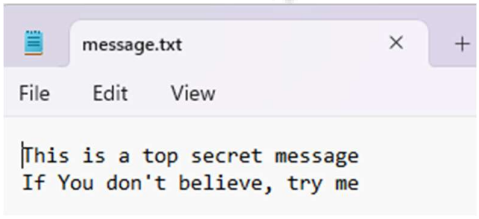
4. **(Optional) Set Password for Encryption**
   Enter a password to encrypt the hidden message for extra protection.
5. **Embed Message**
   Click **"Hide Data"**.
   OpenStego will generate the stego image (stego_image.png) which looks identical to the original cover image.

3. Steps for Extracting (Decoding) the Hidden Message:

1. Open **OpenStego** and choose **"Extract Data"** mode.
2. **Select Stego Image:** Browse and select the file stego_image.png.
3. **Output Folder:** Choose where to save the extracted message.
4. **Enter Password (if used)** to decrypt.
5. Click **"Extract Data"**.
6. The hidden message (message.txt) will be successfully extracted and saved in the chosen folder.







OUTPUT ANALYSIS

Step	Input	Output	Observation
1	Cover image (nature.jpg)	—	Original image used for hiding data
2	Secret message file (message.txt)	—	File containing hidden text
3	After embedding	stego_image.png	Visually identical to cover image
4	After extraction	message.txt	Message successfully retrieved

Result:
The hidden text was securely embedded and later retrieved with no visible distortion in the image.

CONCLUSION:

Using **OpenStego**, data can be effectively hidden inside digital images without altering their visual quality.
This ensures a safe and secret method of communication, making it suitable for digital watermarking, secure transmission, and confidential data sharing.
The experiment successfully demonstrates that steganography using OpenStego is a practical and simple way to protect sensitive information.