

Digital Forensics & Evidence Analysis

From disk images to courtroom evidence — reconstructing the truth from digital artifacts.

What You'll Learn

Disk, memory, network, and log forensics techniques used in incident response and CTF competitions.

AI Advantage

How to leverage LLMs to accelerate artifact analysis, timeline reconstruction, and pattern recognition.

Disk Forensics

Memory Analysis

Network Forensics

Log Correlation

Chain of Custody

AI-Assisted Analysis

Before We Begin

⚠ Educational Use Only

Digital forensics techniques are for authorized investigations and CTF competitions. Improper evidence handling can destroy admissibility. Always follow legal procedures in real investigations.

In this lesson, all examples use purpose-built CTF disk images and synthetic data. Never perform forensic acquisition on systems you do not own or have written authorization to examine.

What Is Digital Forensics?

The scientific process of identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally admissible.

The Four Pillars

- **Acquisition** — Creating a forensically sound copy of digital evidence
- **Preservation** — Maintaining integrity through hashing and chain of custody
- **Analysis** — Extracting artifacts, building timelines, correlating events
- **Presentation** — Documenting findings for stakeholders, legal teams, or courts

Notes: Emphasize that forensics is a discipline, not just a toolset. Methodology matters more than any individual tool.

Forensic Investigation Workflow

Phase	Activities	Key Tools
Identification	Determine scope, identify evidence sources	Inventory, interviews
Collection	Bit-for-bit imaging, RAM capture, log export	dd, FTK Imager, DumpIt
Examination	Extract artifacts, recover deleted files	Autopsy, Volatility, Wireshark
Analysis	Correlate timelines, attribute actions	log2timeline, AI/LLM assistance
Reporting	Document findings, maintain chain of custody	Case notes, hash verification

Notes: Each phase feeds into the next. Skipping or rushing a phase compromises the entire investigation.

Chain of Custody & Legal Admissibility

Evidence is only useful if it can be proven authentic and unaltered. Chain of custody documents every person who handled the evidence, when, and why.

Chain of Custody Requirements

- **Who** collected the evidence
- **When** it was collected (timestamps)
- **How** it was stored and transferred
- **Hash verification** at every handoff

Hash Integrity

```
# Create forensic image with hash
dc3dd if=/dev/sda of=evidence.dd \
    hash=sha256 log=acquisition.log

# Verify later
sha256sum evidence.dd
# a3f2b8c1...must match original
```

If the hash changes at any point, the evidence is considered **tainted** and may be inadmissible in court.

Disk Forensics: Imaging

A forensic image is a bit-for-bit copy of a storage device, including unallocated space, slack space, and deleted data.

Common Imaging Formats

Format	Extension	Features
Raw / dd	.dd, .raw, .img	Exact bit copy, universally supported
E01 (EnCase)	.E01	Compression, built-in hashing, metadata
AFF4	.aff4	Open format, compression, signing

```
# Create raw image with dd (read-only mount first!)
dd if=/dev/sdb of=suspect_drive.dd bs=4096 conv=noerror,sync status=progress

# Or use dc3dd for built-in hashing
dc3dd if=/dev/sdb of=suspect_drive.dd hash=md5 hash=sha256 log=imaging.log
```

Notes: Always use a hardware write-blocker when imaging physical drives. In CTFs, you'll typically receive a pre-made image file.

File System Analysis

Understanding file system structures reveals where data hides — even after deletion.

Key Concepts

- **MFT (NTFS)** — Master File Table tracks every file's metadata
- **Inodes (ext4)** — Index nodes store file metadata on Linux
- **Slack space** — Unused bytes at end of last cluster
- **Unallocated space** — Clusters not assigned to any file

Sleuth Kit Commands

```
# List file system info
fsstat evidence.dd

# List files including deleted
fls -r -d evidence.dd

# Read file by inode
icat evidence.dd 12345 > recovered.doc

# Get file metadata
istat evidence.dd 12345
```

File Carving & Deleted File Recovery

File carving recovers files based on their headers and footers, ignoring file system metadata entirely. This works even when the file system is damaged or wiped.

Common File Signatures

File Type	Header (Hex)	Footer (Hex)
JPEG	FF D8 FF E0	FF D9
PNG	89 50 4E 47	AE 42 60 82
PDF	25 50 44 46	25 25 45 4F 46
ZIP/DOCX	50 4B 03 04	50 4B 05 06
ELF	7F 45 4C 46	Variable

```
# Carve files with Scalpel
scalpel -c /etc/scalpel/scalpel.conf -o carved_output/ evidence.dd
```

```
# Or use foremost
foremost -t jpeg,png,pdf -i evidence.dd -o carved_output/

# Results:
# carved_output/jpg/00001234.jpg  (156 KB)
# carved_output/pdf/00005678.pdf  (2.3 MB)
# carved_output/png/00009012.png  (89 KB)
```

Timeline Analysis

Timelines correlate file system timestamps (MACB) across an entire disk image to reconstruct what happened and when.

MACB Timestamps

- **M** — Modified (content changed)
- **A** — Accessed (file read)
- **C** — Changed (metadata modified, or Created on NTFS)
- **B** — Born / Created (NTFS \$STANDARD_INFORMATION)

```
# Generate timeline with Sleuth Kit + log2timeline
fls -r -m "/" evidence.dd > bodyfile.txt
mactime -b bodyfile.txt -d > timeline.csv

# Sample output:
# Date,Size,Type,Mode,UID,GID,Meta,File Name
# 2024-03-15T02:14:33,4096,d/d,drwxr-xr-x,0,0,1234,/tmp/.hidden_dir
# 2024-03-15T02:14:45,18432,.a.b,rrwxr-xr-x,0,0,5678,/tmp/.hidden_dir/payload.elf
# 2024-03-15T02:15:01,0,m...,rrw-r--r--,0,0,9012,/var/log/auth.log
```

Notes: The timeline above tells a story: a hidden directory was created, a payload was dropped, and then auth.log was modified (possibly to cover tracks).

Autopsy: GUI Forensic Platform

Autopsy is the open-source digital forensics platform built on The Sleuth Kit. It provides a graphical interface for disk analysis, keyword search, and artifact extraction.

Key Features

- **Ingest modules** — Hash lookup, keyword search, web artifacts
- **Timeline view** — Visual MACB timeline
- **Deleted files** — Recovery with visual indicators
- **Tagging** — Mark items as notable or suspicious
- **Reporting** — Generate HTML/PDF case reports

Browser Artifacts Found

Web History:

```
2024-03-14 23:41  pastebin.com/raw/x8k2mN
2024-03-15 01:12  github.com/exploit-db/...
2024-03-15 01:58  mega.nz/file/download
```

Web Downloads:

```
nc.exe           (2024-03-15 02:01)
mimikatz.zip     (2024-03-15 02:03)
```

Web Cookies:

```
c2-server.evil.com  session=a4f2b8...
```

Memory Forensics: Why RAM Matters

RAM contains evidence that never touches disk: encryption keys, running processes, network connections, injected code, and command history.

What Lives Only in Memory

- **Encryption keys** — TrueCrypt, BitLocker, LUKS master keys
- **Running malware** — Fileless attacks exist only in memory
- **Network connections** — Active C2 sessions and socket state
- **Decrypted data** — Passwords, tokens, session cookies
- **Process injection** — Malicious code hiding inside legitimate processes
- **Command history** — CMD/PowerShell commands not yet flushed to logs

```
# Capture memory on Windows  
winpmem_mini_x64.exe memdump.raw
```

```
# Capture memory on Linux
```

```
sudo avml memory.lime
```

```
# Capture with DumpIt (simplest for Windows)
```

```
DumpIt.exe    # Creates HOSTNAME-YYYYMMDD-HHMMSS.dmp
```

Volatility Framework

Volatility is the gold standard for memory forensics. Version 3 uses a plugin-based architecture with automatic OS profile detection.

```
# Identify the memory image profile
vol3 -f memdump.raw windows.info

# Output:
# Variable      Value
# Kernel Base   0xf8000261a000
# DTB           0x187000
# Symbols        ntkrnlmp.pdb
# Is64Bit        True
# IsPAE          False
# primary layer  Intel32e
# OS             Windows 10 19041
```

Essential Volatility Plugins

Plugin	Purpose
<code>windows.pslist</code>	List running processes

Plugin	Purpose
<code>windows.pstree</code>	Process tree (parent-child relationships)
<code>windows.netscan</code>	Network connections and listening ports
<code>windows.cmdline</code>	Command-line arguments for each process
<code>windows.dlllist</code>	Loaded DLLs per process
<code>windows.malfind</code>	Detect injected/hidden code in processes
<code>windows.filescan</code>	Scan for file objects in memory

Process Analysis with Volatility

Process trees reveal attacker activity. Look for unusual parent-child relationships and suspicious process names.

```
$ vol3 -f memdump.raw windows.pstree
```

PID	PPID	ImageFileName	CreateTime
4	0	System	2024-03-15 00:00:12
88	4	Registry	2024-03-15 00:00:12
344	4	smss.exe	2024-03-15 00:00:15
432	344	csrss.exe	2024-03-15 00:00:18
508	344	wininit.exe	2024-03-15 00:00:18
* 596	508	services.exe	2024-03-15 00:00:19
** 1204	596	svchost.exe	2024-03-15 00:00:22
** 2048	596	svchost.exe	2024-03-15 00:00:22
* 604	508	lsass.exe	2024-03-15 00:00:19
3712	3456	explorer.exe	2024-03-15 01:12:33
* 4088	3712	chrome.exe	2024-03-15 01:15:01
* 5124	3712	cmd.exe	2024-03-15 02:14:08 <-- suspicious

Red Flags in Process Trees

- **svchost.exe** not spawned by `services.exe` (PID 596)
 - **cmd.exe / powershell.exe** spawned from user apps at unusual hours
 - **Misspelled names:** `scvhost.exe` , `svch0st.exe`
 - **Processes running from** `%TEMP%` or `%APPDATA%`
-

Detecting Malware in Memory

The `malfind` plugin detects process injection by looking for memory regions with suspicious permissions (RWX) and PE headers.

```
$ vol3 -f memdump.raw windows.malfind
```

PID	Process	Start VPN	End VPN	Tag	Protection
5344	svchost.exe	0x2b40000	0x2b80000	VadS	PAGE_EXECUTE_READWRITE

Disasm:

```
0x2b40000: 4d 5a 90 00 03 00 00 00    MZ.....
0x2b40008: 04 00 00 00 ff ff 00 00    .....
0x2b40010: b8 00 00 00 00 00 00 00    .....
```

Injected PE detected at 0x2b40000 in svchost.exe (PID 5344)

```
# Dump the suspicious process for further analysis
```

```
vol3 -f memdump.raw windows.dumpfiles --pid 5344
```

```
# Check network connections from this process
```

```
vol3 -f memdump.raw windows.netscan | grep 5344
```

```
# 5344 svchost.exe TCPv4 192.168.1.50:49721 -> 45.33.32.156:443 ESTABLISHED
```

```
# Extract command lines
```

```
vol3 -f memdump.raw windows.cmdline --pid 5200  
# 5200 powershell.exe "powershell.exe -enc aQB1AHgAIAAoAE4AZQB3AC0ATwBiAGoA..."
```

Notes: The base64-encoded PowerShell command is a classic indicator. Decode it to reveal the actual payload.

Network Forensics: PCAP Analysis

Packet captures (PCAPs) record every byte that traversed the network. They are often the most objective evidence source.

What PCAPs Reveal

- **C2 communication** — Beaconsing patterns, data exfiltration
- **Lateral movement** — SMB, RDP, SSH connections
- **Data exfiltration** — DNS tunneling, HTTP POST uploads
- **Exploit delivery** — Malicious payloads in transit

Capture Sources

- **tcpdump / Wireshark** — Host-based capture
- **Network TAPs** — Passive hardware capture
- **SPAN ports** — Switch port mirroring
- **IDS/IPS** — Zeek, Suricata full PCAP

```
# Quick triage with tshark
tshark -r capture.pcap -q -z conv,tcp
```

```
# Filter for suspicious traffic
```

```
tshark -r capture.pcap -Y "ip.dst == 45.33.32.156" -T fields -e frame.time -e tcp.len
```

Session Reconstruction & Artifact Extraction

Reconstruct full application-layer sessions from raw packets to see exactly what the attacker sent and received.

```
# Follow a TCP stream in tshark
tshark -r capture.pcap -q -z follow,tcp,ascii,0

# Extract HTTP objects (files downloaded)
tshark -r capture.pcap --export-objects http,exported_files/

# Extract DNS queries for tunneling detection
tshark -r capture.pcap -Y "dns.qr == 0" -T fields \
  -e frame.time -e dns.qry.name | sort | uniq -c | sort -rn

# Results showing DNS tunneling:
#   847  data.evil-c2.com          <-- excessive queries!
#    23  google.com
#    12  microsoft.com
```

```
POST /upload HTTP/1.1
Host: 45.33.32.156
Content-Type: application/octet-stream
Content-Length: 4521984
```

```
[... binary data: exfiltrated database dump ...]
```

```
HTTP/1.1 200 OK
```

```
{"status": "received", "bytes": 4521984}
```

Log Forensics: Timeline Correlation

No single log source tells the whole story. Correlating timestamps across multiple logs reveals the full attack chain.

Time (UTC)	Source	Event
02:13:45	Apache access.log	POST /wp-admin/upload.php — webshell uploaded
02:14:02	Apache access.log	GET /uploads/cmd.php?c=whoami
02:14:08	Windows Security	Event 4688: cmd.exe created by w3wp.exe
02:14:12	Sysmon Event 1	powershell.exe -enc aQB1AHgA...
02:14:33	Sysmon Event 1	svchost.exe spawned from PowerShell
02:15:01	auth.log	File modification detected — log tampering
02:15:44	Firewall	Outbound TCP 443 to 45.33.32.156 (C2)
02:18:22	Sysmon Event 3	Network connection from fake svchost.exe

Notes: Walk through this timeline step by step. Each log source fills in a piece of the puzzle.

Windows Event Log Forensics

Windows Event Logs (`.evtx`) are one of the richest forensic sources on Windows systems.

Critical Event IDs

Event ID	Log	Meaning
4624	Security	Successful logon (check logon type)
4625	Security	Failed logon (brute force indicator)
4688	Security	New process created (with command line)
4720	Security	User account created
7045	System	New service installed (persistence)
1102	Security	Audit log cleared (anti-forensics!)

```
# Parse EVTX with python-evtx
python -c "
import Evtx.Evtx as evtx
```

```
with evtx.Evtx('Security.evtx') as log:
    for record in log.records():
        xml = record.xml()
        if 'EventID>4688' in xml:
            print(record.timestamp(), xml[:200])
"
```

Windows Registry Forensics

The Windows Registry is a treasure trove of forensic artifacts — it records user activity, installed software, USB history, and persistence mechanisms.

Key Forensic Registry Locations

```
# Persistence mechanisms
```

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

```
"WindowsUpdate" = "C:\Users\admin\AppData\Local\Temp\svchost.exe"  <-- SUSPICIOUS
```

```
# Recently executed programs (UserAssist)
```

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist
```

```
{CEBFF5CD}\Count\P:\Hfref\nqzva\Qbjaybnqf\zvzvxngm.rkr    (ROT13 encoded!)
```

```
# Decoded: C:\Users\admin\Downloads\mimikatz.exe
```

```
# Run count: 3      Last run: 2024-03-15 02:16:44
```

```
# USB Device History
```

```
HKLM\SYSTEM\CurrentControlSet\Enum\USB
```

```
VID_0781&PID_5567  SanDisk Cruzer Blade  Serial: 4C530001...
```

```
# First connected: 2024-03-15 02:20:11
```

Email Forensics: Header Analysis

Email headers trace the message path from sender to recipient and reveal spoofing attempts, relay servers, and authentication results.

```
Received: from mail-gateway.company.com (10.0.0.5)
  by exchange01.company.com (10.0.0.10) with ESMTTP;
  Fri, 15 Mar 2024 02:10:33 +0000
Received: from smtp.evill-domain.com (45.33.32.156)          <-- origin IP
  by mail-gateway.company.com with ESMTTP;
  Fri, 15 Mar 2024 02:10:31 +0000
From: "IT Support" <itsupport@company.com>                  <-- spoofed!
Reply-To: helpdesk@evill-domain.com                          <-- real address
X-Mailer: Microsoft Outlook 16.0
Authentication-Results: spf=fail smtp.mailfrom=evill-domain.com;
  dkim=none; dmarc=fail action=none                          <-- auth failed!
```

Header Red Flags

- **SPF/DKIM/DMARC failures**
- **Reply-To** differs from **From**
- **Origin IP** doesn't match claimed domain

Attachment Analysis

- Check for **double extensions**:
`invoice.pdf.exe`
- Extract and hash attachments

- **Unusual X-Mailer** strings

- Look for **macro-enabled** Office docs
 - Scan with YARA rules or VirusTotal
-

AI-Assisted Email Analysis

LLMs excel at parsing complex email headers and identifying anomalies that manual review might miss.

```
import anthropic

client = anthropic.Anthropic()

email_headers = open("suspicious_email.eml").read()

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2000,
    messages=[{
        "role": "user",
        "content": f"""Analyze these email headers for forensic investigation.
Identify:
1. The true origin IP and sender
2. Any spoofing indicators (SPF/DKIM/DMARC results)"""
    }])
```

Sample AI Analysis Output

Origin: 45.33.32.156 (evil-domain.com) — **SPF FAIL** for company.com domain.

Spoofing: From header claims itsupport@company.com but originated from evil-domain.com.

Reply-To redirect: Replies would go to helpdesk@evil-domain.com, not the claimed sender.

Verdict: High-confidence phishing email. Recommend blocking 45.33.32.156 and alerting all recipients.

AI-Assisted Hex Dump Analysis

LLMs can rapidly identify file types, embedded strings, and anomalies in hex dumps that would take a human analyst much longer to spot.

```
import subprocess, anthropic

client = anthropic.Anthropic()

# Extract hex dump of suspicious file
hex_dump = subprocess.run(
    ["xxd", "-l", "512", "carved_file.bin"],
    capture_output=True, text=True
).stdout

response = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=2000,
    messages=[{
        "role": "user",
```

```
# Example hex dump the AI might analyze:
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ.....
00000010: b800 0000 0000 0000 4000 0000 0000 0000  .....@.....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....
```

```
00000030: 0000 0000 0000 0000 0000 0000 f000 0000 .....
```

```
# AI identifies: PE executable, likely Windows malware
```

AI-Assisted Process Tree Analysis

Feed Volatility output directly to an LLM to identify suspicious process relationships, injection indicators, and attack patterns.

```
import anthropic, subprocess

client = anthropic.Anthropic()

# Capture Volatility output
pstree = subprocess.run(
    ["vol3", "-f", "memdump.raw", "windows.pstree"],
    capture_output=True, text=True
).stdout

netscan = subprocess.run(
    ["vol3", "-f", "memdump.raw", "windows.netscan"],
    capture_output=True, text=True
).stdout
```

AI-Assisted Registry Analysis

Registry hives contain thousands of keys. LLMs can rapidly scan exported registry data to flag persistence mechanisms, anti-forensics, and user activity.

```
import anthropic

client = anthropic.Anthropic()

# Registry export from RegRipper or regipy
registry_data = """
[HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run]
"SecurityHealth" = "C:\\Windows\\System32\\SecurityHealthSystray.exe"
"WindowsDefender" = "C:\\ProgramData\\Microsoft\\wduupdate.exe"
"OneDriveSync" = "C:\\Users\\admin\\AppData\\Roaming\\ODSync\\sync.exe"

[HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce]
"Cleanup" = "cmd.exe /c del /f /q C:\\Users\\admin\\AppData\\Local\\Temp\\*. * & wevtutil cl Security"
"""
```

AI Would Flag

- **"WindowsDefender"** in `ProgramData` — legitimate Defender doesn't run from there
 - **"Cleanup" RunOnce** — deletes temp files AND clears the Security event log (anti-forensics)
 - **"OneDriveSync"** in Roaming — suspicious path, not official OneDrive location
-

Anti-Forensics Awareness

Sophisticated attackers actively work to destroy, alter, or hide evidence. Knowing anti-forensics techniques helps you detect them.

Technique	Description	Detection Method
Timestomping	Altering file timestamps to blend in	Compare \$STANDARD_INFO vs \$FILE_NAME timestamps in MFT
Log wiping	Clearing event logs or deleting log files	Event ID 1102 (audit cleared), gaps in log sequences
Secure deletion	Overwriting file content before deletion	Look for clusters with repeating patterns (0x00, 0xFF)
Encryption	Encrypting payloads or exfiltrated data	High entropy analysis on files/network traffic
Steganography	Hiding data inside images or audio	Statistical analysis, file size anomalies
Fileless malware	Running entirely in memory	Memory forensics (Volatility malfind)

Technique	Description	Detection Method
Log injection	Adding fake log entries to mislead	Format inconsistencies, impossible timestamps

Detecting Timestomping

NTFS stores two sets of timestamps: `$STANDARD_INFORMATION` (easily modified) and `$FILE_NAME` (harder to fake). Comparing them reveals manipulation.

```
# analyzeMFT output showing timestomping
File: C:\Windows\System32\svchost_update.exe

$STANDARD_INFORMATION:
  Created:    2023-08-15 10:30:00    <-- Claims to be old
  Modified:   2023-08-15 10:30:00
  Accessed:   2024-03-10 09:00:00

$FILE_NAME:
  Created:    2024-03-15 02:14:33    <-- Actually very recent!
  Modified:   2024-03-15 02:14:33

ALERT: $SI Created (2023-08-15) is BEFORE $FN Created (2024-03-15)
       This is physically impossible -- TIMESTOMPING DETECTED!
```

```
# Detect with analyzeMFT
analyzeMFT.py -f MFT -o mft_analysis.csv
```

```
# Or use Volatility for in-memory MFT analysis  
vol3 -f memdump.raw windows.mftscan.MFTScan
```

Detecting Log Wiping

Attackers frequently clear logs to cover their tracks. Multiple indicators can reveal this activity.

Indicators of Log Wiping

- **Event ID 1102** — "The audit log was cleared" (can't delete this one!)
- **Event ID 104** — System log was cleared
- **Time gaps** in sequential log entries
- **Log files smaller** than expected for uptime
- **Missing .evtx files** in

```
C:\Windows\System32\winevt\Logs\
```

Evidence in Registry

```
# RunOnce anti-forensics entry
[HKCU\SOFTWARE\...\RunOnce]
"Cleanup" = "cmd.exe /c wevtutil cl
Security & wevtutil cl System &
wevtutil cl Application &
del /f C:\Windows\System32\
winevt\Logs\*.evtx"

# PowerShell event log clearing
Clear-EventLog -LogName Security
# or
wevtutil cl Security
```

Notes: Even when logs are cleared, Event ID 1102 records who cleared them and when. This is often the most

damning evidence.

Browser Artifact Forensics

Web browsers store a wealth of forensic data in SQLite databases and JSON files.

Chrome Forensic Artifacts

Artifact	Location	Database/Table
History	AppData/Local/Google/Chrome/User Data/Default/	History → urls, visits
Downloads	Same directory	History → downloads
Cookies	Same directory	Cookies → cookies
Cached pages	Cache/Cache_Data/	Binary cache files
Saved passwords	Same directory	Login Data → logins

```
# Query Chrome history with sqlite3
sqlite3 "History" "SELECT datetime(last_visit_time/1000000-11644473600,'unixepoch'),
url, title FROM urls ORDER BY last_visit_time DESC LIMIT 20;"

# Output:
```

```
# 2024-03-15 02:03:11 | https://mega.nz/file/download?id=... | MEGA - Download
# 2024-03-15 01:58:22 | https://github.com/gentilkiwi/mimikatz | mimikatz
# 2024-03-15 01:12:05 | https://pastebin.com/raw/x8k2mN | Pastebin.com - raw
```

Super Timelines with Plaso

Plaso (log2timeline) creates a unified "super timeline" from hundreds of artifact sources — file system, event logs, browser history, registry, prefetch, and more.

```
# Step 1: Parse all artifacts into a Plaso storage file
log2timeline.py --storage-file case.plaso evidence.dd

# Step 2: Filter and export to CSV
psort.py -o l2tcsv -w timeline.csv case.plaso \
    "date > '2024-03-15 00:00:00' AND date < '2024-03-16 00:00:00'"

# Step 3: Review in Timeline Explorer or spreadsheet
# Columns: datetime, timestamp_desc, source, sourcetype, type, user, host, message
```

Super Timeline Power

A single Plaso run on a disk image can extract **millions of events** from 100+ parsers: MFT, EVTX, Prefetch, Chrome, Firefox, Outlook, USBStor, registry hives, LNK files, Jump Lists, and more. This is the definitive timeline for any investigation.

Notes: Plaso can take hours to run on large images. Plan accordingly. Use filters aggressively when analyzing output.

AI-Assisted Timeline Analysis

Super timelines can contain millions of entries. Use an LLM to identify the critical events and reconstruct the attack narrative.

```
import anthropic, csv

client = anthropic.Anthropic()

# Read a filtered portion of the super timeline
with open("timeline_filtered.csv") as f:
    reader = csv.DictReader(f)
    events = [row for row in reader][:500] # First 500 events in window

timeline_text = "\n".join(
    f"{e['datetime']} | {e['source']} | {e['sourcetype']} | {e['message'][:150]}"
    for e in events
)

response = client.messages.create(
```

Forensic Toolkit Reference

Category	Tool	Purpose
Disk Imaging	dd, dc3dd, FTK Imager	Bit-for-bit forensic copies
Disk Analysis	Autopsy, Sleuth Kit	File system analysis, file recovery
File Carving	Scalpel, Foremost, PhotoRec	Recover files from raw data
Memory Capture	WinPMEM, AVML, DumpIt	RAM acquisition
Memory Analysis	Volatility 3	Process, network, malware analysis
Network	Wireshark, tshark, NetworkMiner	PCAP analysis, session reconstruction
Timeline	Plaso (log2timeline)	Super timeline from all artifact sources
Registry	RegRipper, regipy	Registry hive parsing
EVTX	python-evtx, EvtxECmd	Windows Event Log parsing

Category	Tool	Purpose
----------	------	---------

AI	Claude API	Artifact interpretation, timeline analysis
----	------------	--

CTF Forensics Methodology

A systematic approach to forensic CTF challenges ensures you don't miss hidden evidence.

1. **Identify the image type** — `file image.dd` to determine disk, memory, or PCAP
2. **Hash the evidence** — `sha256sum image.dd` before any analysis
3. **Mount / extract** — Mount disk images read-only; load memory in Volatility
4. **Low-hanging fruit** — `strings`, `grep` for flag patterns, file carving
5. **Structured analysis** — File system timeline, process trees, network connections
6. **Deep dive** — Registry, browser artifacts, deleted files, slack space
7. **AI assistance** — Feed complex artifacts (hex dumps, process trees, logs) to Claude
8. **Document findings** — Build the attack timeline and reconstruct the narrative

```
# Quick forensic triage script
file evidence.dd
sha256sum evidence.dd
strings -a evidence.dd | grep -iE "flag\{|ctf\{|password|secret" | head -20
```

```
mmls evidence.dd      # partition layout  
fls -r evidence.dd    # file listing
```

CTF Challenge: Reconstruct the Attack

Scenario

A company's web server was compromised. You've been given a forensic disk image (`webserver.dd`) and a memory dump (`memdump.raw`). Reconstruct the complete attack timeline.

Your Mission

1. **Initial Access** — How did the attacker get in? Find the webshell or exploit evidence.
2. **Execution** — What commands did the attacker run? Check process trees and command history.
3. **Persistence** — What persistence mechanisms were installed? Check registry, services, scheduled tasks.
4. **Exfiltration** — What data was stolen? Analyze network connections and file access patterns.
5. **Anti-forensics** — What did the attacker try to hide? Detect timestomping, log wiping, deleted files.

6. **IOCs** — Extract all indicators of compromise: IPs, domains, file hashes, user accounts.

Deliverable

A forensic report with a complete attack timeline (UTC), supported by evidence from at least three different forensic sources (disk, memory, logs). Use AI to help analyze complex artifacts and generate the narrative.

Challenge Approach Hints

```
# 1. Triage the disk image
mmls webserver.dd
fls -r -o 2048 webserver.dd | grep -i "\.php\|\.exe\|\.ps1"

# 2. Build a timeline of the attack window
fls -r -m "/" -o 2048 webserver.dd > bodyfile.txt
mactime -b bodyfile.txt -d 2024-03-15 > timeline_0315.csv

# 3. Analyze memory for running processes
vol3 -f memdump.raw windows.pstree
vol3 -f memdump.raw windows.netscan
vol3 -f memdump.raw windows.cmdline
vol3 -f memdump.raw windows.malfind

# 4. Check for anti-forensics
```

Notes: Encourage students to use multiple forensic sources to corroborate each finding. A single source is circumstantial; multiple sources build a strong case.

Key Takeaways

Forensic Fundamentals

- Always preserve evidence integrity with **hashing and chain of custody**
- **Disk forensics**: imaging, file carving, timeline analysis, registry
- **Memory forensics**: Volatility for processes, network, malware detection
- **Network forensics**: PCAP analysis, session reconstruction

AI-Powered Analysis

- Feed **hex dumps, process trees, registry exports** to LLMs for rapid interpretation
- Use AI to **correlate super timelines** and build attack narratives
- AI excels at spotting **anomalies in structured data**
- Always **verify AI findings** with tool-based evidence

Remember

No single artifact tells the whole story. The best forensic investigations **corroborate findings across disk, memory, network, and log evidence**. AI accelerates analysis but never replaces

methodical investigation.

Up Next: Lesson 10

Continue building your blue team skills with the next lesson in the CTF Blue Team course.

Prepare By

- Install **Autopsy** and **Volatility 3** on your analysis workstation
- Practice with the **NIST CFReDS** sample disk images
- Download **Wireshark** sample captures from the wiki
- Complete the CTF challenge and document your forensic report

