

Digital Forensics & Evidence Analysis

Systematically collecting, preserving, and analyzing digital evidence to reconstruct security incidents and support legal proceedings.

WARNING: Proper Evidence Handling Required. Digital forensics evidence is only admissible in legal proceedings if a strict chain of custody is maintained from the moment of collection. Never work directly on original evidence media -- always create verified forensic images first. Use write-blockers when connecting suspect drives. Document every action you take, including timestamps, tools used, and hash values. Failure to follow proper procedures can render evidence inadmissible and compromise an entire investigation. If you are unsure whether an investigation may lead to legal action, assume it will and follow forensic best practices from the start.

What Is Digital Forensics?

Digital forensics is the application of scientific methods to the identification, collection, preservation, examination, analysis, and reporting of digital evidence. It encompasses evidence found on computers, mobile devices, network traffic, cloud services, and any system that stores or transmits data electronically. Unlike general incident response -- which focuses on containment and recovery -- forensics prioritizes the integrity of evidence so findings can withstand legal scrutiny.

For blue teams, digital forensics serves several critical purposes. It answers the fundamental questions of an incident: what happened, when it happened, how it happened, and who was responsible. It provides the evidentiary foundation for disciplinary action, civil litigation, or criminal prosecution. It also feeds lessons learned back into the organization's defenses, revealing gaps that allowed the incident to occur in the first place.

The discipline demands a careful balance between speed and thoroughness. Evidence is volatile -- a running system's memory contents change every millisecond, log files

rotate, and attackers may actively destroy traces of their activity. The forensic examiner must act quickly to preserve evidence while meticulously documenting every step to maintain its integrity.

The Forensic Investigation Workflow

The standard forensic investigation follows six phases, often described by the NIST SP 800-86 framework. Each phase builds on the previous one, and mistakes early in the process can invalidate everything that follows.

1. **Identify.** Determine which systems, devices, and data sources are relevant to the investigation. This includes live systems, backup media, cloud storage, network devices, mobile phones, and IoT devices. Identify the scope of the incident and prioritize evidence sources based on volatility -- RAM disappears first, then running processes, then network connections, then disk contents.
2. **Preserve.** Protect evidence from modification or destruction. Isolate affected systems from the network (but do not power them off if memory forensics is needed). Apply write-blockers to storage media. Capture volatile evidence (memory dumps, running processes) before non-volatile evidence. Calculate cryptographic hashes of all evidence to establish a baseline for integrity verification.
3. **Collect.** Create forensic images of storage media and memory. Use validated tools to produce bit-for-bit copies. Record the chain of custody for every piece of evidence: who collected it, when, where it was stored, and who has had access. Every transfer of evidence must be documented.
4. **Examine.** Process the collected evidence to extract relevant data. This includes recovering deleted files, parsing filesystem metadata, extracting log entries, carving files from unallocated space, and decrypting encrypted volumes where keys are available.
5. **Analyze.** Interpret the extracted data to reconstruct the timeline of events. Correlate findings across multiple evidence sources. Identify indicators of compromise, attacker techniques, lateral movement, data exfiltration, and persistence mechanisms.
6. **Report.** Document all findings in a clear, structured report suitable for the intended audience -- whether that is management, legal counsel, law enforcement, or a court

of law. Include methodology, tools used, evidence hashes, timeline of events, and conclusions supported by evidence.

Chain of Custody and Legal Admissibility

The chain of custody is an unbroken record that documents who had possession of evidence, when, and what they did with it. Every forensic image, memory dump, and log file must have an associated chain-of-custody form. The form records: the item description, serial numbers and identifying marks, the date and time of collection, the name and role of the person who collected it, every subsequent transfer between individuals, how and where the evidence was stored, and what actions were performed on it.

Legal admissibility requires demonstrating that evidence has not been tampered with. This is why forensic examiners calculate SHA-256 hashes at collection time and verify them before analysis. If the hash of a disk image does not match the hash recorded at collection, the evidence may be challenged in court. Work only on copies, never on original media, and maintain hash verification at every stage.

Disk Forensics

Disk forensics involves creating bit-for-bit images of storage media and analyzing their contents. The imaging process captures everything on the drive, including deleted files, unallocated space, slack space, and hidden partitions.

Forensic Imaging

```
# Create a forensic image using dd (basic)
# /dev/sdb is the suspect drive (connected via write-blocker)
dd if=/dev/sdb of=/evidence/case001/disk.img bs=4M status=progress

# Better: use dcfldd for hashing during acquisition
dcfldd if=/dev/sdb of=/evidence/case001/disk.img \
  bs=4M hash=sha256 hashlog=/evidence/case001/disk.hash \
  hashwindow=1G

# Best: use dc3dd with verification
dc3dd if=/dev/sdb of=/evidence/case001/disk.img \
```

```
hof=/evidence/case001/disk.img.sha256 hash=sha256 \  
log=/evidence/case001/acquisition.log  
  
# Verify image integrity  
sha256sum /evidence/case001/disk.img  
# Compare output against the hash recorded during acquisition
```

Filesystem Analysis

Once an image is acquired, tools like Autopsy, The Sleuth Kit (TSK), and FTK Imager allow examiners to explore the filesystem without modifying the image. Common tasks include recovering deleted files from unallocated space, examining filesystem timestamps (created, modified, accessed), parsing the Master File Table (MFT) on NTFS volumes, and identifying file type mismatches where an attacker has renamed a malicious executable to look like a document.

```
# The Sleuth Kit -- list files including deleted entries  
fls -r -p /evidence/case001/disk.img  
  
# Extract a specific file by inode number  
icat /evidence/case001/disk.img 12345 > /evidence/case001/recovered_file.dat  
  
# Parse the NTFS MFT for timeline data  
fls -r -m "/" /evidence/case001/disk.img > /evidence/case001/bodyfile.txt  
mactime -b /evidence/case001/bodyfile.txt -d > /evidence/case001/timeline.csv  
  
# Carve files from unallocated space using Scalpel  
scalpel /evidence/case001/disk.img -o /evidence/case001/carved/
```

Memory Forensics

Memory forensics analyzes the contents of a system's RAM to reveal running processes, open network connections, loaded DLLs, injected code, encryption keys, and other volatile artifacts that disappear when a system is powered off. The Volatility framework is the de facto standard tool for memory analysis.

Memory Acquisition

```
# Linux: capture memory using /proc/kcore or LiME  
# LiME (Linux Memory Extractor) kernel module  
sudo insmod lime.ko "path=/evidence/case001/memory.lime format=lime"
```

```
# Windows: use winpmem
winpmem_mini_x64.exe /evidence/case001/memory.raw

# Verify the memory dump
sha256sum /evidence/case001/memory.lime
```

Volatility Framework Analysis

```
# Identify the operating system profile
vol.py -f memory.raw imageinfo

# Volatility 3 syntax (current version)
vol3 -f memory.raw windows.info

# List running processes -- look for suspicious names or parent relationships
vol3 -f memory.raw windows.pslist
vol3 -f memory.raw windows.pstree # Tree view shows parent-child

# Detect hidden processes (rootkits)
vol3 -f memory.raw windows.psscan # Scans for EPROCESS structures

# Examine network connections
vol3 -f memory.raw windows.netscan

# Detect injected code in processes
vol3 -f memory.raw windows.malfind

# Dump a suspicious process for further analysis
vol3 -f memory.raw windows.dumpfiles --pid 4892 --dump-dir ./dumped/

# Extract command line arguments
vol3 -f memory.raw windows.cmdline

# List loaded DLLs for a process
```

Log Forensics and Timeline Reconstruction

Log files are often the richest source of forensic evidence. They record authentication events, process execution, network connections, file access, and system changes. The challenge is that logs come from dozens of sources in different formats -- Windows

Event Logs, syslog, application logs, web server access logs, firewall logs, and cloud audit trails -- and must be correlated into a single coherent timeline.

```
#!/usr/bin/env python3
"""Forensic timeline builder -- merge multiple log sources."""
import csv
import json
from datetime import datetime
from pathlib import Path

def parse_syslog(filepath):
    """Parse standard syslog format."""
    events = []
    with open(filepath) as f:
        for line in f:
            # Example: Jan 15 03:22:41 webserver sshd[12345]: Failed password
            parts = line.strip().split(None, 5)
            if len(parts) >= 6:
                timestamp_str = f"{parts[0]} {parts[1]} {parts[2]}"
                ts = datetime.strptime(
                    f"2025 {timestamp_str}", "%Y %b %d %H:%M:%S"
                )
                events.append({
                    "timestamp": ts.isoformat(),
                    "source": filepath,
                    "host": parts[3],
                    "process": parts[4],
                    "message": parts[5]
                })
```

Network Forensics

Network forensics involves capturing and analyzing network traffic to identify malicious activity, reconstruct communications, and trace data exfiltration. Packet capture (pcap) files are the primary evidence format, typically collected from network taps, span ports, or host-based tools.

```
# Capture traffic from a specific interface
tcpdump -i eth0 -w /evidence/case001/capture.pcap -c 1000000

# Filter pcap for specific indicators
```

```
# Extract all DNS queries (common C2 channel)
tshark -r capture.pcap -Y "dns.qr == 0" -T fields -e dns.qry.name | sort -u

# Extract HTTP objects (downloaded files)
tshark -r capture.pcap --export-objects http,/evidence/case001/http_objects/

# Reconstruct TCP sessions
tshark -r capture.pcap -z "follow,tcp,ascii,0" > session_0.txt

# Find connections to known bad IPs
tshark -r capture.pcap -Y "ip.addr == 198.51.100.23" -T fields \
  -e frame.time -e ip.src -e ip.dst -e tcp.dstport
```

AI-Assisted Forensics

Large language models like Claude can significantly accelerate forensic investigations by analyzing large volumes of data, correlating events across sources, and drafting structured reports. AI does not replace the examiner's judgment, but it handles the tedious pattern-matching work that consumes hours of manual effort.

Analyzing Memory Dumps with AI

Feed Volatility output directly to Claude for rapid triage. The AI can identify suspicious processes, flag anomalous parent-child relationships, and highlight indicators of compromise that might be missed in thousands of lines of output.

```
"""Send Volatility output to Claude for forensic triage."""
import anthropic
import subprocess

client = anthropic.Anthropic()

def analyze_memory_with_ai(memory_dump_path):
    """Run Volatility plugins and send output to Claude for analysis."""
    # Collect output from multiple Volatility plugins
    plugins = ["windows.pslist", "windows.netscan", "windows.malfind",
               "windows.cmdline"]

    vol_output = {}
    for plugin in plugins:
        result = subprocess.run(
            ["vol3", "-f", memory_dump_path, plugin],
```

```
capture_output=True, text=True
)
vol_output[plugin] = result.stdout

# Build the prompt with all forensic data
analysis_prompt = f"""You are a digital forensics analyst examining a memory dump from a potentially compromised Windows system. Analyze the following Volatility output and identify:

1. Suspicious processes (unusual names, unexpected parent processes, processes running from temp directories)
2. Signs of malware injection (malicious results)
```

Automated Report Generation

After completing the technical analysis, use Claude to draft a structured forensic report. Provide it with your timeline, findings, and evidence hashes. The AI produces a professional report that you then review, verify, and sign. This reduces report-writing time from hours to minutes while ensuring consistent formatting and completeness.

Anti-Forensics: Techniques and Detection

Sophisticated attackers use anti-forensics techniques to hinder investigation. Understanding these methods is essential for detecting them.

Technique	Description	Detection Method
Timestamp manipulation	Altering file MAC times to hide activity	Compare filesystem timestamps against log entries and MFT records; look for timestamps that predate the OS installation
Log clearing	Deleting or truncating system logs	Gaps in log sequences, Event ID 1102 (Windows audit log cleared), missing log files with recent modification times
Secure deletion	Overwriting file contents before deletion	Tools leave characteristic patterns; journal entries may still reference the file; memory may contain remnants
Steganography	Hiding data inside images or other files	Statistical analysis of image files, unusual file sizes, tools like StegDetect

Technique	Description	Detection Method
In-memory only malware	Malware that never touches disk	Memory forensics with Volatility; EDR telemetry; network traffic analysis
Encrypted volumes	Using BitLocker, VeraCrypt to encrypt evidence	Memory forensics may recover encryption keys; check for key escrow; hibernation files may contain decrypted data
Process hollowing	Replacing a legitimate process's memory with malicious code	Volatility malfind; compare process memory against on-disk executable

Tip: The best defense against anti-forensics is collecting evidence from multiple independent sources. An attacker can wipe disk logs but cannot easily alter network tap captures, cloud audit trails, and EDR telemetry simultaneously. Correlating across sources reveals gaps and inconsistencies that indicate anti-forensic activity.

Further Reading

- [NIST SP 800-86: Guide to Integrating Forensic Techniques into Incident Response](#) - the foundational framework for digital forensics
- [Volatility Foundation](#) -- documentation and plugin references for memory forensics
- [The Sleuth Kit & Autopsy](#) -- open-source disk forensics toolkit
- [SANS Windows Forensic Analysis Poster](#) -- quick reference for Windows artifact locations
- [DFIR Training](#) -- community resource for digital forensics tools and techniques

Download Files

These files contain synthetic forensic artifacts for the challenges. All files are plain text (.txt) and safe to open.

↓ [event-log-extract.txt](#) ~14 KB

↓ [filesystem-timeline.txt](#) ~7 KB

↓ [volatility-output.txt](#) ~11 KB

↓ [network-forensics.txt](#) ~13 KB