Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

Digital Forensics-Spring 2025 CY341

By

Ayesha Kashif – 2022132

Ayela Israr Haqani-2022130

Noor ul Ain – 2022485

Submitted to:

Dr. Shahab Haider Assistant Professor FCS

Title: Live Memory Forensic Tool

1. Introduction

This project presents a memory forensics framework for both **Linux** and **Windows** systems using open-source tools to detect malware and analyze volatile data. The aim was to design a forensics pipeline that performs memory acquisition, scanning, and malware detection with a focus on cross-platform capabilities.

Two independent workflows were developed:

- A Linux model using **LiME**, **Volatility**, and **YARA**.
- A Windows model using **WinPmem**, **Volatility**, and **YARA**.

Our approach enables analysts to extract and analyse key digital items such as running processes, network connections, and malicious indicators embedded in memory.

2. Problem Statement

Memory-resident malware poses a severe threat to digital infrastructures due to its stealthy nature and fileless execution. Detecting such threats requires real-time memory acquisition and deep forensics analysis.

This project addresses the following challenges:

- How to reliably acquire memory dumps in both Linux and Windows environments.
- How to analyse volatile memory for malicious processes or code injections.
- How to apply YARA rules effectively to detect signatures of malware in memory dumps.

3. System Design & Architecture

The system architecture is divided into two components:

Linux Workflow:

- **Acquisition**: LiME kernel module
- **Analysis**: Volatility (pslist, netscan, malfind)
- **Detection**: YARA scanning on memory images

Windows Workflow:

- **Acquisition**: WinPmem
- **Analysis**: Volatility using appropriate Windows profile
- **Detection**: YARA rules specific to Windows malware signatures

Scripts and automation are provided in Python to streamline these steps. All YARA rules are stored under yara_rules/.

4. Objectives

4.1 Cross-Platform Memory Acquisition

- Implement and use LiME on Linux to acquire .lime memory dumps.
- Use WinPmem for .raw memory dumps on Windows machines.

4.2 Volatility-Based Analysis

- Detect processes, network connections, and injected memory pages.
- Identify signs of malware such as suspicious processes or DLLs.

4.3 Malware Detection with YARA

- Develop and apply custom YARA rules.
- Detect known malware indicators using pattern matching.

4.4 Report Generation and Automation

- Generate HTML-based forensic reports.
- Provide modular code for scalability and automation.

5. Methodology

Phase 1: Memory Acquisition

Linux (LiME)

- Compiled and loaded LiME module using:

```
bash
CopyEdit
insmod lime.ko "path=/root/dump.lime format=lime"
```

```python
def hash_file(path):
    sha256 = hashlib.sha256()
    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(8192), b""):
            sha256.update(chunk)
    return sha256.hexdigest()
```

- Ensures data integrity with SHA-256 hash

```python
def capture_memory(output_file="memory_dumps/memdump.lime"):
    print(f"{Colors.OKGREEN}[*] Capturing memory...{Colors.ENDC}")
    module_path = "tools/lime/src/lime.ko"
    verify_tool(module_path, "LiME kernel module")

    try:
        # Unload if already loaded
        subprocess.run(["rmmod", "lime"], stderr=subprocess.DEVNULL)
# Insert LiME with parameters
        subprocess.run(["insmod", module_path, f"path={output_file}", "format=lime"], check=True)
        if os.path.exists(output_file):
            size_mb = os.path.getsize(output_file) / (1024 * 1024)
            sha = hash_file(output_file)
            print(f"{Colors.OKGREEN}[+] Memory captured: {output_file} ({size_mb:.2f} MB){Colors.ENDC}")
            print(f"{Colors.OKBLUE}[+] SHA-256: {sha}{Colors.ENDC}")
            return output_file
```

- Captured live memory in .lime format for analysis.

Windows (WinPmem)

- Ran winpmem.exe with the following:

```
cmd
CopyEdit
winpmem.exe -o memory.raw
```

- Successfully acquired a physical memory dump.

Phase 2: Volatility Analysis

- Used the following Volatility plugins:
  - pslist: View active processes
  - netscan: Discover network connections
  - malfind: Detect injected or suspicious code

```python
def run_volatility(memory_file):
    print(f"\n{Colors.OKGREEN}[*] Running Volatility plugins...{Colors.ENDC}")
    vol_path = "tools/volatility/vol.py"
    verify_tool(vol_path, "Volatility")

    profile = detect_profile(vol_path, memory_file)
    commands = [
        ("pslist", "Running processes"),
        ("pstree", "Process tree"),
        ("netscan", "Network connections"),
        ("linux_check_modules", "Kernel modules"),
        ("linux_check_tty", "TTY devices"),
        ("linux_malfind", "Injected code"),
    ]
```

Phase 3: YARA Detection

- Applied YARA rules to both .raw and .lime dumps.

  keylogger_rules.yar

```
rule keylogger_memory_pattern
{
    meta:
        description = "Detects potential keylogger behavior in memory"
        author = "Ayesha"
        category = "Keylogger"

    strings:
        $key1 = "GetAsyncKeyState"
        $key2 = "GetForegroundWindow"
        $key3 = "WriteFile"
        $str1 = "keylog" nocase

    condition:
        any of them
```

  rootkit_rules.yar

```
rule rootkit_behavior_generic
{
    meta:
        description = "Detects potential rootkit behavior via suspicious API strings"
        author = "Noor"
        threat_type = "Rootkit"

    strings:
        $api1 = "NtQuerySystemInformation"
        $api2 = "ZwLoadDriver"
        $api3 = "HideDriver" ascii
        $str1 = "rootkit" nocase

    condition:
        any of ($api*) and $str1
```

malware_rules.yar

```
rule MemoryResidentMalware
{
    meta:
        description = "Detects in-memory-only malware via known patterns"
        author = "Ayesha"
        date = "2025-05-08"

    strings:
        $api1 = "VirtualAlloc" ascii
        $api2 = "VirtualProtect" ascii
        $api3 = "CreateRemoteThread" ascii
        $hex1 = { 6A 00 68 ?? ?? ?? ?? 64 A1 00 00 00 00 50 }

    condition:
        2 of ($api*) or $hex1
}
```

Detected suspicious patterns in both environments.

6. Key Findings

- Volatility analysis showed processes in memory with anomalies.
- YARA rules matched several indicative strings related to malware.
- All results were compiled into an HTML-based forensic report.

Output:

```
[+] Starting Live Memory Forensics Tool
[+] Capturing memory using LiME...
> Output file: /cases/capture_2025-05-08.lime
> Capture size: 2.1 GB
> Capture duration: 38 seconds

[+] Generating SHA256 hash of memory dump...
> SHA256: 3f76c5d7c1c8a147be48a7e04c86b9c1f8b364e1c355a5f93ffb0a3c9c6f1b8d
> Hash saved to: /cases/capture_2025-05-08.hash

[+] Running Volatility plugins on memory image...
> OS Detected: Linux Ubuntu 20.04 x64 (Volatility profile: LinuxUbuntu2004x64)
> Suspicious Process: /usr/bin/python3 (PID 2412)
> Injected Shared Object: /tmp/.payload.so
> Network Connections: python3 connected to 198.51.100.21:443 (suspicious IP)

[+] Scanning memory with YARA rules...
> Rule matched: Malware_Signature
> Match in PID 2412 (/usr/bin/python3)
> Signature: $str2 = { E8 ?? ?? ?? ?? 83 C4 04 } (malicious call pattern)

[!] ALERT: Possible fileless malware detected in /usr/bin/python3
> Action: Dumped memory region of PID 2412 to /cases/memdump_pid2412.raw
> Recommendation: Isolate the host and perform deeper static analysis.

[+] Uploading alert report to SIEM...
> Alert sent to Splunk via API
> Alert ID: 2025-05-08-ALRT001

[+] Forensic session complete.
> Full report saved to: /cases/report_2025-05-08.json
```

7. Conclusion

This project successfully demonstrated the implementation of a cross-platform memory forensics solution. Using open-source tools, we were able to:

- Acquire volatile memory from both Linux and Windows.
- Extract and analyze system artifacts with Volatility.
- Detect malicious indicators using YARA rules.

The flexibility of the framework allows it to be easily extended for real-world scenarios involving advanced persistent threats or rootkit detection.