CW2 — Exercise 2: Automating Volatility2 (Memory Triage Script)

1. **Purpose and scope**

This repository contains a small command-line automation script that wraps Volatility 2 to perform repeatable triage of a Windows memory dump (e.g., win.mem).

The objective is to automatically extract the artefacts required for CW2 Exercise 2, specifically:

- The operating system version (via Volatility profile inference)
- Running processes, including detection of potentially hidden processes
- Open network connections/socket artefacts
- Logged-on users
- Evidence from console/terminal windows (command history and console buffers)

The script is designed for an academic forensic workflow, meaning it:
1) produces raw plugin outputs for reproducibility, and
2) generates a single consolidated report for quick review.

2. **What the script collects (Volatility2 plugins)**

The tool executes the following Volatility 2 plugins:

2.1 OS identification

- imageinfo -> Used to infer the most likely Volatility profile (and therefore OS family/architecture).

2.2 Process listing + hidden process detection

- pslist (active process list)
- psscan (memory scan for process objects)

The script compares pslist vs psscan and highlights processes that appear in psscan but not in pslist, as these can indicate:

- terminated-but-still-resident process structures, or
- stealthy/hidden processes (depending on context).

2.3 Network artefacts

- connscan
- sockets

These plugins provide evidence of network connections and socket structures present in memory.

2.4 Logged-on users

- loggedin-> This is used to identify active / recently active user sessions present in the memory image.

2.5 Console and command artefacts

- consoles
- cmdscan

These plugins provide:

- console screen buffers (what was visible in console windows), and
- command history or command processor artefacts.

3. **Repository structure**

Typical structure (your submission may include additional demo artefacts):

- Volatility.py - main automation script (CLI wrapper)
- demo-task2/ - example outputs from a demonstration run
- output/<run>/ - run directory created by the tool

4. Requirements

This tool is intended to run in the lab VM environment.

4.1 System requirements

- Python 3
- Volatility 2 installed and callable (e.g., `volatility2` available in PATH)
- A Windows memory image (e.g., `win.mem`)

4.2 Python dependencies

No third-party Python packages are required. The script uses only the Python standard library.

5. How to run

5.1 Basic usage

Run the script by providing:

- the memory image path, and

- an output directory where results will be written.

Example:
bash
python3 volatality.py \
--mem /home/user/Downloads/cw2-espionage/win.mem \
--outdir demo-task2/output/runfile

6. What happens during execution

The script will:

1. Run the required Volatility 2 plugins,
2. store each plugin's output in a raw/ folder, and
3. Build a consolidated report and structured summary.

7. Output (what you should expect)

After a successful run, the output directory will contain:

- report.txt
  A consolidated human-readable report containing:
  - inferred OS/profile information
  - process listing and hidden-process comparison
  - network artefacts summary
  - logged-on users
  - console/cmd artefacts
- summary.json
  A structured machine-readable summary of key findings (useful for quick checks or extension work).

- raw/
  Raw outputs for each Volatility plugin (evidence preservation/reproducibility), e.g.:
  - imageinfo.txt
  - pslist.txt
  - psscan.txt
  - connscan.txt
  - sockets.txt
  - loggedin.txt
  - cmdscan.txt
  - consoles.txt

This separation (raw artefacts vs consolidated report) aligns with standard forensic practice: raw outputs support transparency and verification, while the report supports interpretation.

8. Processes and Artefacts

8.1 “Hidden” processes

A process present in psscan but missing from pslist is **not automatically malicious**.

It may represent:

- a legitimately terminated process whose artefacts remain in memory, or
- process manipulation/stealth techniques (if supported by surrounding evidence).

8.2 Network artefacts

Memory-based network artefacts may include:

- stale connections,

- partial socket structures,
- incomplete metadata (depending on capture timing and OS).

Therefore, results should be interpreted as “observed in memory at acquisition time”, not as a definitive network ground truth.