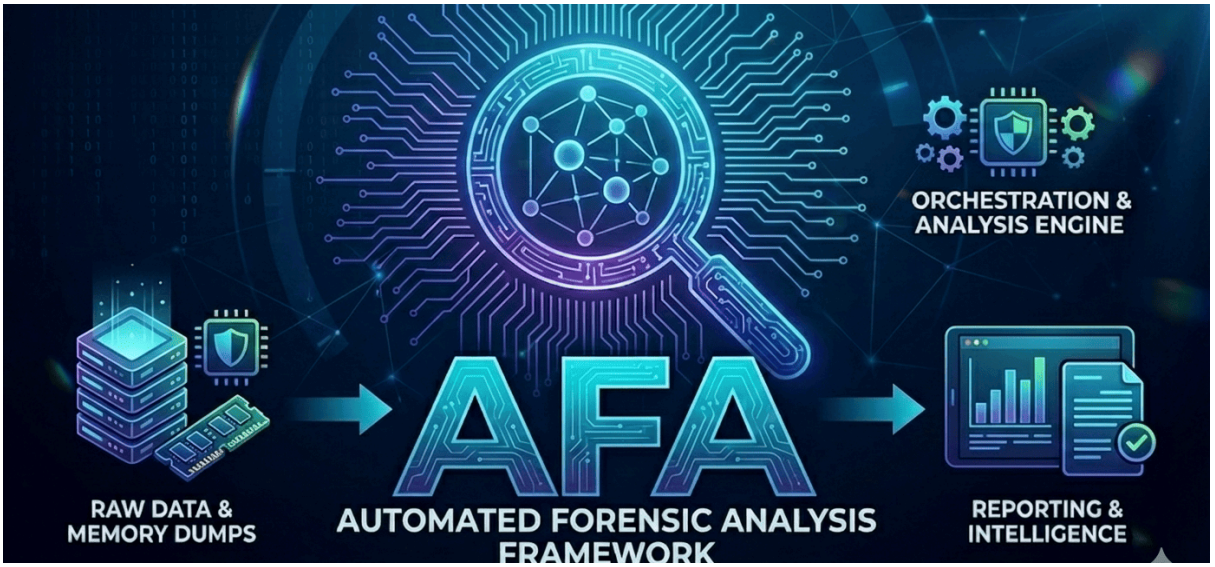
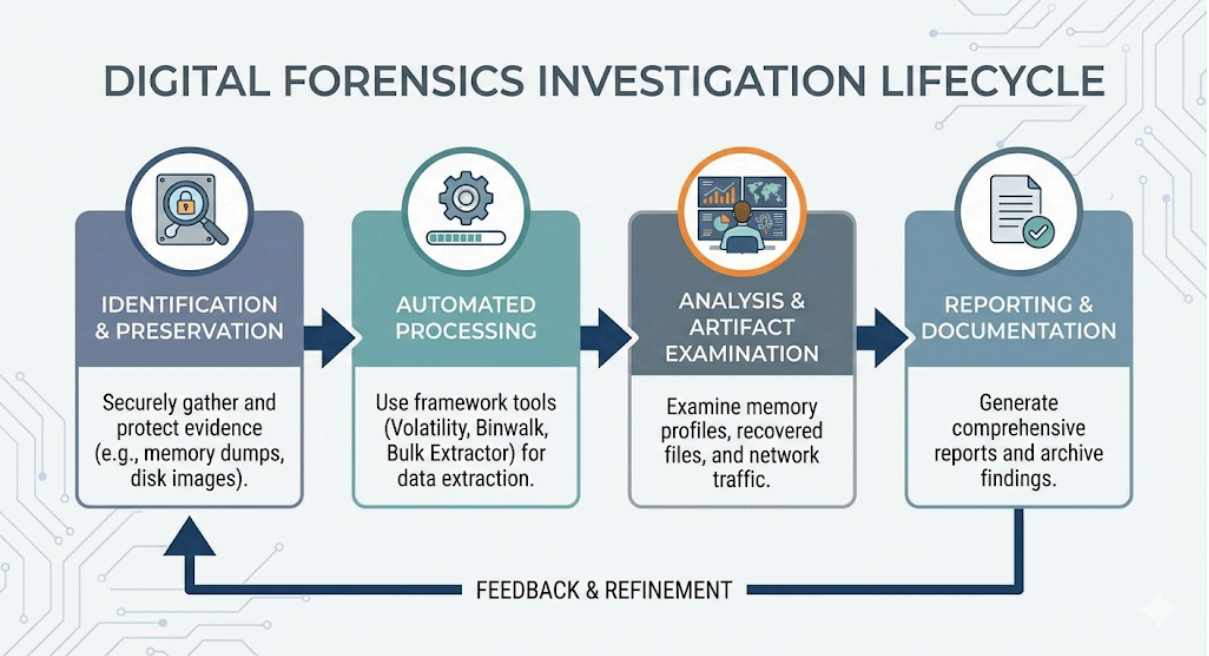


Project Documentation: Automated Forensic Analysis Framework (AFA)

1. Executive Summary

The **Automated Forensic Analysis Framework (AFA)** is a specialized Bash-based orchestration tool designed to accelerate the digital forensics lifecycle. By abstracting the complexity of command-line tools such as Volatility, Bulk Extractor, and Binwalk, the framework provides a unified interface for analyzing memory dumps and disk images.



The primary objective of this project is to reduce the "Time to Evidence" (TTE) by automating dependency management, standardizing output directory structures, and executing multi-stage analysis workflows—ranging from memory profiling to heuristic file carving and credential harvesting.

2. Technical Architecture & Capabilities

The framework operates on a modular architecture, distinct functions handling initialization, dependency resolution, user interaction, and core analysis tasks.

- **Memory Forensics Engine:**
  - Integrates **Volatility 2.5** to perform deep memory analysis.
  - Automates the `imageinfo` profile detection step to ensure compatibility before executing plugins.
  - Supports granular plugin selection (e.g., `pslist`, `netscan`, `hivelist`) or batch execution for comprehensive auditing.
- **Artifact Carving & Extraction:**
  - Orchestrates **Bulk Extractor**, **Binwalk**, and **Foremost** to recover deleted files and extract embedded artifacts from raw images.
  - Implements logic to detect extracted `.pcap` files automatically, streamlining the transition from disk forensics to network traffic analysis.

- **Static Analysis & Credential Hunting:**
  - Utilizes the `strings` utility combined with Regular Expressions to filter memory dumps for human-readable text.
  - Applies heuristic pattern matching to identify potential sensitive data, such as passwords, tokens, and authentication keys.
- **Self-Healing Environment:**
  - Includes a robust initialization routine (`CHECK_APPS`) that verifies the presence of required binaries.
  - Capable of auto-resolving missing dependencies via `apt-get` and downloading legacy tools (e.g., Volatility 2.5) directly from external sources if not present locally.

---

3. Core Logic & Implementation

The following sections detail the implementation of critical system functions.

A. Interactive Command & Control Interface

The `MENU` function serves as the central control loop, offering a dynamic command-line interface (CLI). It supports multi-argument input, allowing analysts to chain multiple analysis vectors (e.g., "Run Binwalk and Volatility") in a single session.

```
# Function: MENU
# Purpose: Main event loop handling user input and tool selection.
function MENU() {
    while true; do
        # ... [Header Display Logic] ...

        # Capture multi-select input (e.g., "1 3 5")
        echo -e -n "${YELLOW}Enter your choice(s): ${RESET}${BLUE}"
        read -r OPTIONS_ALL
        OPTIONS_ALL=($(echo "$OPTIONS_ALL" | tr ',' ' '))

        for OPTIONS in "${OPTIONS_ALL[@]}"; do
            case "$OPTIONS" in
                1)
                    # Module: Bulk Extractor
```

```
                bulk_extractor "$path" -o "$OUTPUT_DIR/bulk_extractor_outpu
t" > /dev/null 2>&1
                TOOLS_USED+=("bulk_extractor")
                ;;
            4)
                # Module: Volatility Integration
                RUN_VOLATILITY
                TOOLS_USED+=("Volatility")
                ;;
            6)
                # Module: Full Audit (Batch Processing)
                echo -e "${BRIGHT_YELLOW}Running all forensic tools...${RES
ET}"
                # ... [Sequential Execution Logic] ...
                break
                ;;
        esac
    done
  done
}
```

B. Automated Memory Profiling (Volatility Wrapper)

This function encapsulates the complexity of Volatility. It programmatically determines the correct memory profile using `imageinfo` and validates it before attempting to run analysis plugins, preventing common execution errors.

```
# Function: RUN_VOLATILITY
# Purpose: Automates profile detection and plugin execution for Volatility 2.
5.
function RUN_VOLATILITY() {
  echo -e "${YELLOW}[*] Determining memory profile...${RESET}"

  # Execute imageinfo and parse output for the suggested profile
  PROFILE_OUTPUT=$(sudo ./vol -f "$path" imageinfo 2>/dev/null)
  PROFILE=$(echo "$PROFILE_OUTPUT" | grep "Suggested Profile" | cut -
d ":" -f2 | cut -d "," -f1 | xargs)
```

```
    if [ -z "$PROFILE" ]; then
        echo -e "${RED}Error: Failed to determine memory profile.${RESET}"
        return
    fi

    echo -e "${GREEN}Profile Detected: ${BLUE}$PROFILE${RESET}"

    # Plugin Execution Loop
    for plugin in "${PLUGINS[@]}"; do
        # Execute plugin and redirect output to standardized directory
        sudo ./vol -f "$path" --profile="$PROFILE" "$plugin" --output-file
="$VOL_DIR/${plugin}.txt" > /dev/null 2>&1

        # Logic for Hive Dumping
        if [ "$plugin" == "hivelist" ]; then
            # ... [Registry Extraction Logic] ...
        fi
    done
}
```

C. Network Artifact Heuristics

The framework bridges the gap between file and network forensics. If the file carving modules produce a packet capture (`.pcap`) file, the system detects it and prompts the analyst to launch Wireshark immediately.

```
# Function: EXTRACT_NETWORK_TRAFFIC
# Purpose: Detects PCAP artifacts and integrates with Wireshark.
function EXTRACT_NETWORK_TRAFFIC() {
    # Scan output directories for PCAP files
    pcap_file=$(find "$OUTPUT_DIR/bulk_extractor_output" -type f -iname
"*.pcap" 2>/dev/null | head -n 1)

    if [ -n "$pcap_file" ]; then
        filesize=$(du -h "$pcap_file" | cut -f1)
        echo -e "${YELLOW}Network Artifact Detected: ${BLUE}$pcap_file
($filesize)${RESET}"
```

```
        # Interactive prompt for immediate analysis
        echo -en "${YELLOW}Open in Wireshark? (y/n): ${RESET}"
        read -t 15 -r open_choice

        if [[ "$open_choice" =~ ^[Yy]$ ]]; then
            wireshark "$pcap_file" &
        fi
    fi
}
```

D. Heuristic Credential Analysis

This module performs static analysis on the raw image file. It extracts ASCII strings and filters them against a predefined dictionary of high-value keywords (e.g., "password", "token"), isolating potential credentials into separate report files.

```
# Function: CHECK_HUMAN_READABLE
# Purpose: Static string analysis and credential pattern matching.
function CHECK_HUMAN_READABLE() {
    # Dictionary of target keywords
    GREP_FLAG=("password" "passwd" "login" "username" "user" "credenti
al" "token" "auth" "secret" "key")

    echo -e "${YELLOW}[*] Performing static string extraction...${RESET}"
    strings -n 6 "$path" > "$STRINGS_FILE"

    echo -e "${YELLOW}[*] Filtering for credential patterns...${RESET}"
    for flag in "${GREP_FLAG[@]}"; do
        grep -i "$flag" "$STRINGS_FILE" > "$CREDS_DIR/$flag.txt" 2>/dev/nul
l
    done
}
```

4. Reporting & Archival

To ensure data integrity and ease of transfer, the system concludes operations by generating a statistical summary (`statistics.txt`) and compressing the entire session directory into a standardized ZIP archive.

```
function STATISTICS() {
    # ... [Log Generation] ...

    # Archival Routine
    echo -e "${YELLOW}[*] Archiving session data...${RESET}"
    zip -r "${OUTPUT_DIR}.zip" "$OUTPUT_DIR" > /dev/null 2>&1
    echo -e "${YELLOW}Archive Created: ${BLUE}${OUTPUT_DIR}.zip${RE
SET}"
}
```