

Memory Forensics

Live Data, Hidden Threats: Extracting Truth from Memory

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License. You are free to share and adapt this work for non-commercial purposes, as long as appropriate credit is provided. For more details, visit <https://creativecommons.org/licenses/by-nc/4.0/>.

Agenda

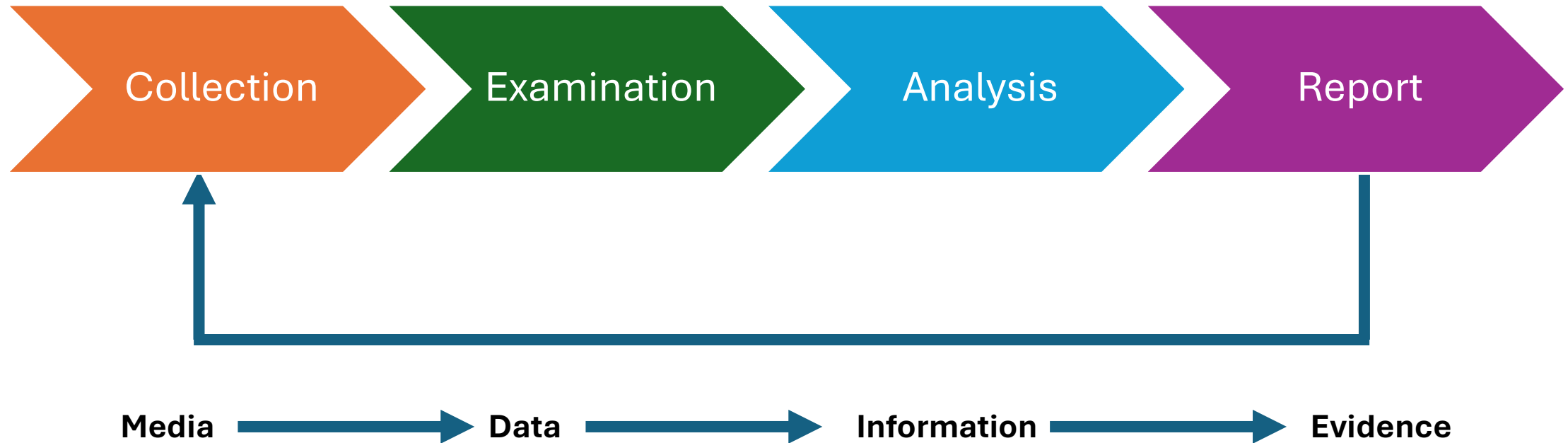
- Introduction to Memory Forensics
- Conducting Live Forensics
- Introduction to the Volatility Framework
- Practical Application of Volatility
 - Advanced Volatility Usage *
 - Analysing Windows memory dumps
- Q&A session
- Lab 1 Live Forensics- Investigating a Compromised Linux System
- Lab 2 Introduction to Volatility - Analysing a Linux memory dump
- Lab 3 Introduction to Volatility plugins - Analysing a Linux memory dump
- Lab 4 Analysing a Windows memory dump

<https://github.com/waz-here/labs/tree/main/volatility>

Introduction to Memory Forensics

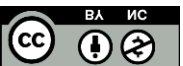
- Memory forensics is the process of extracting and analysing a system's **volatile memory (RAM)** to uncover hidden threats, investigate cyber incidents, and recover forensic artefacts.
- It helps IT professionals detect malware, advanced persistent threats (APTs), and fileless attacks that don't leave traces on disk.
- NIST 800-86 Guide to Integrating Forensic Techniques into Incident Response (2006)
 - Defines forensic acquisition, examination, and analysis best practices.
 - Emphasises chain of custody, data integrity, and structured methodologies.

Performing the Forensic Process



National Institute of Standards and Technology [NIST], SP 800-86 2006, p. 25)

<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-86.pdf#page=25>



Introduction to Memory Forensics

Types of Data Found in RAM

- Running Processes – Identify active programs, injected malware, or suspicious activities.
- Encryption Keys & Credentials – Recover passwords, SSH keys, and cryptographic secrets from RAM.
- Network Connections – Reveal exfiltration channels, backdoors, and C2 servers.
- Loaded DLLs & Kernel Modules – Detect rootkits and persistence techniques in the OS kernel.

Introduction to Memory Forensics

Active Process Information

- Running Processes – Programs and background services currently executing.
- Hidden or Suspended Processes – Malware and stealth processes that evade standard process listings.
- Parent-Child Relationships – Connections between processes (e.g., cmd.exe launching powershell.exe)

Why It Matters:

- Attackers often inject malicious code into running processes (e.g., DLL injection).
- Fileless malware exists only in running processes and disappears on reboot.

Introduction to Memory Forensics

Unencrypted Credentials & Authentication Tokens

- Windows LSASS (Local Security Authority Subsystem Service) – Stores plaintext passwords, NTLM hashes, and Kerberos tickets in RAM.
- SSH & RDP Session Keys – Active remote sessions store authentication keys in memory.
- Web Browser Session Tokens – Cookies and authentication tokens (e.g., logged-in Gmail or banking sessions).

Why It Matters:

- Mimikatz can extract passwords from LSASS memory.
- Attackers use Pass-the-Hash and Pass-the-Ticket techniques to move laterally.

Introduction to Memory Forensics

Active Network Connections & Encrypted Traffic

- Open TCP/UDP Sockets – Shows real-time command-and-control (C2) communications.
- DNS Cache & Recent Resolutions – Tracks resolved domain names that malware connects to.
- Decrypted HTTPS Sessions – Memory holds plaintext versions of encrypted communications.

Why It Matters:

- Malware like Cobalt Strike and Emotet establishes hidden C2 channels only visible in RAM.
- Attackers use DNS tunnelling for stealthy data exfiltration.

Introduction to Memory Forensics

In-Memory Code Execution (Fileless Malware)

- Injected Code – Malicious scripts loaded directly into RAM (e.g., Cobalt Strike beacons, Mimikatz).
- Reflective DLL Injection – Malicious DLLs loaded without touching disk.
- PowerShell & Living-Off-The-Land Binaries (LOLBins) – Attacks executed from memory-only scripts.

Example:

- Attackers use Metasploit Meterpreter, which runs entirely in RAM, leaving no trace on disk.

Introduction to Memory Forensics

Clipboard & Keystroke Data

- Copy/Paste History – Data copied to the clipboard remains in memory.
- Keylogging Data – Some malware captures typed passwords before encryption.
- Screenshots in Memory – Some malware stores captured screenshots before exfiltration.

Example:

- Redline Stealer and Agent Tesla extract clipboard data for stealing credentials.

Introduction to Memory Forensics

How Memory Forensics is Performed

- Capture Memory – Use tools like DumpIt (Windows), WinPmem (Windows), LiME (Linux), or macOS tools.
- Analyse Memory Dumps – Process memory images using Volatility, Rekall, MemProcFS.
- Correlate Findings – Cross-reference with disk forensics, logs, threat intelligence sources.

Conducting Live Forensics

- Live forensics aims to capture real-time information that would otherwise be lost when a system is shut down, enabling a more comprehensive investigation of active threats and incidents.
- Quickly assess system compromise, identify indicators of attack, and collect actionable forensic evidence.

RAM → Network Connections → Running Processes → Disk → Logs

Conducting Live Forensics

- Live Response Commands – Execute essential forensic commands to collect volatile system data.
- General System Information – Gather hostname, kernel version, and system uptime.
- Logon Activities – Examine authentication logs for suspicious logins and privilege escalations.
- Process Review – Identify active processes, hidden or rogue binaries, and memory-resident threats.
- Deleted Process Recovery – Retrieve deleted process executables from disk or memory.
- Network Analysis – Inspect active connections, open ports, and unauthorised remote access.
- User and System Activity Review – Analyse command history, cron jobs, and file access logs.
- File System Investigation – Detect unusual files, unauthorised modifications, and hidden directories.
- Installed Programs & Persistence Mechanisms – Audit installed packages, startup scripts, and backdoor persistence.
- Unusual System Resources – Identify anomalies in CPU, memory, and disk usage indicative of malware.

Conducting Live Forensics

Description/Task	Windows Command	Linux Command
Check currently logged-in users	query user	who
List all running processes	tasklist	ps aux
Identify suspicious processes by network activity	netstat -ano findstr ESTABLISHED	lsof -i -n -P grep ESTABLISHED
Check for recently created or modified files	dir /S /T:W C:\path\to\check	find /path/to/check -type f -printf '%TY-%Tm-%Td %TT %p\n' sort
Analyse scheduled tasks or cron jobs	schtasks /query /fo LIST	crontab -l; ls /etc/cron*
Identify installed keylogger tools	powershell "Get-WmiObject Win32_Product Where-Object { \$_.Name -like '*keylogger*' }"	grep -i 'keylogger' /var/log/syslog
Review system logs for unusual activity	eventvwr.msc	journalctl -xe
Check recent user logins	whoami /logonid	last
Dump network connections for inspection	netstat -bano	ss -tulnp
Verify integrity of critical system files	sfc /scannow	rpm -Va; debsums -s

Lab 1 – Live Forensics

<https://academy.apnic.net/virtual-labs?labId=151590>

APNIC Academy

APNIC

LOG IN

Academy

Courses ▾

Events ▾

Community Experts ▾

About ▾

My Account ▾



Virtual Labs

Introducing a new way of hands-on eLearning

Try out your skills using multiple cloud-based instances of virtual machines and network topologies.

To access Hands-on Virtual Labs, you need to [Login](#) or [Register](#) to APNIC Academy.

[Register now](#)

[Login](#)

114 hours of hands-on virtual labs



Memory Forensics (Linux)

English 3h 00m

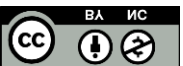
Learn step-by-step how to use the Volatility Framework (an open-source tool) to complete memory analysis and forensics of a Linux system. This virtual lab topology has been set up with one Linux machine.

<https://academy.apnic.net/virtual-labs?labId=151590>

Lab 1 - Live Forensics

- **Simulate an Attack** – Use Netcat to open a bind shell backdoor.
- **Identify & Monitor Suspicious Processes** – Detect abnormal activity.
- **Explore Process Information in /proc** – Analyse running processes stored in the Linux /proc directory.
- **Recover Deleted Executable** – Retrieve binary files of suspicious processes that were removed.
- **Validate Binary Integrity** – Compare recovered binaries against known hashes for tampering.
- **Investigate Process Names & Parameters** – Examine command-line arguments, environment variables, and execution details.
- **Analyse File Descriptors & Memory Mappings** – Inspect open files, process memory maps, and stack contents for forensic clues.
- **Review Process Status** – Extract overall process details using /proc/\$PID/status.

<https://academy.apnic.net/virtual-labs?labId=151590>



Lab 1 - Live Forensics

Command	Description
<code>cd /tmp</code>	Changes the current working directory to /tmp.
<code>cp /bin/nc /tmp/x7</code>	Copies the Netcat binary to /tmp as x7 to simulate malware or a backdoor.
<code>./x7 -vv -k -w 1 -l 31337 > /dev/null &</code>	Starts a verbose Netcat listener on port 31337, simulating a bind shell.
<code>rm x7</code>	Deletes the x7 binary after execution, simulating attacker clean-up.
<code>ps aux grep x7</code>	Searches for running processes named x7 to verify it is still in memory.
<code>netstat -punat</code>	Displays active network connections and listening ports.
<code>PROCESS_ID=\$(lsof -i tail -1 awk -F' ' '{print \$2}' xargs)</code>	Extracts the process ID (PID) of the last network connection shown by lsof.
<code>ls -al /proc/\$PROCESS_ID</code>	Lists all files and directories for the given process in /proc.
<code>ls -al /proc/\$PROCESS_ID grep -E "tmp x7"</code>	Filters the above output to show links to /tmp or the x7 binary.
<code>cp /proc/\$PROCESS_ID/exe /tmp/recovered_bin</code>	Copies the executable in memory to a file, recovering the deleted binary.

Lab 1 - Live Forensics

Command	Description
<code>md5sum recovered_bin /bin/nc</code>	Compares MD5 checksums to check if the recovered binary matches Netcat.
<code>sha256sum recovered_bin /bin/nc</code>	Compares SHA-256 checksums for more secure binary integrity verification.
<code>cat /proc/\$PROCESS_ID/comm</code>	Prints the command name of the process.
<code>cat /proc/\$PROCESS_ID/cmdline && echo "\n"</code>	Shows the full command line used to start the process.
<code>strings /proc/\$PROCESS_ID/environ</code>	Displays environment variables used by the process.
<code>ls -al /proc/\$PROCESS_ID/fd</code>	Lists open file descriptors for the process.
<code>cat /proc/\$PROCESS_ID/maps</code>	Shows the memory map of the process.
<code>sudo cat /proc/\$PROCESS_ID/stack</code>	Displays the current stack trace of the process (requires root).
<code>cat /proc/\$PROCESS_ID/status</code>	Provides detailed process status information (e.g., UID, memory usage).

Lab 1 - Live Forensics

File/Directory	Purpose
/proc/meminfo	Check overall memory usage
/proc/[PID]/maps	Inspect memory mappings for a process
/proc/[PID]/mem	Dump process memory (requires root)
/proc/kcore	Capture a full RAM snapshot
/proc/slabinfo	Detect rootkits and memory leaks
/proc/net/tcp	Analyse network activity in memory

The **/proc** directory in Linux is a virtual filesystem (procfs) that provides a real-time, in-memory representation of system and process information. While it does not store RAM content on disk, it acts as an interface to access live memory structures, making it highly useful for live forensics and troubleshooting.

Lab 1 - Live Forensics

Technique	Command	Purpose
List memory-heavy processes	<code>ps aux --sort=-%mem</code>	Identify unusual memory usage
Inspect memory mappings	<code>cat /proc/[PID]/maps</code>	Find hidden/injected code
Dump process memory	<code>sudo gcore -o dumpfile /proc/[PID]/mem</code>	Extract secrets/malware
Detect hidden processes	<code>ls -l /proc grep -E '[0-9]+'</code>	Find rootkit-hidden processes
Check active TCP connections	<code>cat /proc/net/tcp</code>	Detect backdoors
Find process using a socket	<code>ls -l /proc/*/fd grep socket</code>	Link connections to processes
Monitor system calls	<code>strace -p [PID] -e open,write,execve</code>	Track malicious file modifications
Dump kernel memory	<code>dd if=/proc/kcore of=/tmp/memory.dump bs=1M count=100</code>	Extract rootkits from memory

Discussion / Let's Recap

What is the value of Live Forensics?

- A. Gathering volatile information that would be lost if the computer is turned off or rebooted.
- B. Identifying and analyzing malicious processes and activities as they happen.
- C. Collecting evidence that can be used for incident response and further investigation.
- D. All of the above.

What is the recommended action when dealing with a suspicious process during live forensics?

- A. Investigating the process and gathering information.
- B. Immediately terminating the suspicious process.
- C. Rebooting the compromised system.
- D. Ignoring the suspicious process.

Discussion / Let's Recap

What is the value of Live Forensics?

- A. Gathering volatile information that would be lost if the computer is turned off or rebooted.
- B. Identifying and analyzing malicious processes and activities as they happen.
- C. Collecting evidence that can be used for incident response and further investigation.
- D. All of the above.

What is the recommended action when dealing with a suspicious process during live forensics?

- A. Investigating the process and gathering information.
- B. Immediately terminating the suspicious process.
- C. Rebooting the compromised system.
- D. Ignoring the suspicious process.

Introduction to the Volatility Framework

- The Volatility Framework is an open-source memory forensics tool, written in Python.
 - **Volatility 2** was primarily written in Python 2.7, which became a limitation as Python 2 reached end-of-life in 2020.
 - **Volatility 3** is written in Python 3, making it more modern, efficient, and compatible with current Python libraries.
- Designed to analyse volatile memory (RAM) dumps from various operating systems, including Linux, Windows, macOS, and Android.
- Works on a memory snapshot, avoiding forensic contamination.
- Identifies hidden processes and memory-resident threats.
- Can be extended with plugins for custom analysis.

Introduction to the Volatility Framework

Volatility 2 laid the foundation for memory forensics but had certain limitations:

- **Address Spaces:** It utilised a linear stacking model for address spaces, which limited flexibility in handling complex memory translations.
- **Profiles:** Analysis relied on profiles, which were collections of symbols and types specific to operating system versions. This approach required manual specification and could lead to symbol name collisions due to a single namespace for all symbols.
- **Object Model:** Objects were complex proxy constructs that attempted to emulate host object methods. However, they differed in type and couldn't always be used interchangeably with native Python types, potentially causing inconsistencies during analysis.

Introduction to the Volatility Framework

Volatility 3 is designed as a comprehensive library for memory analysis, focusing on modularity and efficiency. Its core components include:

- **Memory Layers:** These represent bodies of data accessible by specific addresses. Modern systems use paged memory models, and Volatility 3 can handle these by understanding the mappings between virtual and physical addresses. It constructs a directed graph of layers, combining raw memory images and page files to form a cohesive virtual memory layer.
- **Templates and Objects:** Templates define the structure of data without populating it, specifying sizes and member offsets. When applied to a memory layer at a specific offset, they instantiate Objects, which allow interrogation of their members and facilitate pointer traversal. Notably, in Volatility 3, once an Object is created, its data remains static, enhancing analysis consistency.
- **Symbol Tables:** Compiled programs define structures known as Symbols, detailing the structure and location within the program. Volatility 3 accesses these through SymbolTables, organised within a SymbolSpace in a Context. This organisation aids in accurately identifying structures within an operating system by referencing known offsets provided by official debugging information.

Introduction to the Volatility Framework

`volatility --plugins=. -f "memory-dump" --profiles="profile of OS kernel" OScommands`

Command Part	Description
volatility	Main Volatility command for memory analysis.
--plugins=.	Specifies the directory for plugins (current directory in this case).
-f	Specifies the RAM image to analyse.
--profile=	Selects the OS kernel profile (e.g., LinuxUbuntu_4_4_0-21x64)
OScommands	The Volatility plugin used to extract information about the system and memory dump.

Note:

- Volatility 2.x requires an OS profile (e.g., Win7SP1x64, LinuxUbuntu_4_4_0-21x64).
- Volatility 3.x does NOT require profiles because it detects system structures dynamically.

Introduction to the Volatility Framework

```
volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_banner
```

Command Part	Description
volatility	Main Volatility command for memory analysis.
--plugins=.	Specifies the directory for plugins (current directory in this case).
-f memUbuntu.raw	Indicates the input file, which is "memUbuntu.raw," the memory dump for analysis.
--profile=LinuxUbuntu_4_4_0-21x64	Specifies the profile for analysis, tailored to the Linux system with a specific kernel version.
linux_banner	The Volatility plugin used to extract information about the Linux system, including the kernel version and build details.

Introduction to the Volatility Framework

volatility --info | grep -i Linux

linux_apihooks	- Checks for userland apihooks
linux_arp	- Print the ARP table
linux_aslr_shift	- Automatically detect the Linux ASLR shift
linux_banner	- Prints the Linux banner information
linux_bash	- Recover bash history from bash process memory
linux_bash_env	- Recover a process' dynamic environment variables
linux_bash_hash	- Recover bash hash table from bash process memory
linux_check_afinfo	- Verifies the operation function pointers of network protocols
linux_check_creds	- Checks if any processes are sharing credential structures
linux_check_evt_arm	- Checks the Exception Vector Table to look for syscall table hooking
linux_check_fop	- Check file operation structures for rootkit modifications
linux_check_idt	- Checks if the IDT has been altered
linux_check_inline_kernel	- Check for inline kernel hooks
linux_check_modules	- Compares module list to sysfs info, if available
linux_check_syscall	- Checks if the system call table has been altered
linux_check_syscall_arm	- Checks if the system call table has been altered
linux_check_tty	- Checks tty devices for hooks
linux_cpuinfo	- Prints info about each active processor
linux_dentry_cache	- Gather files from the dentry cache
linux_dmesg	- Gather dmesg buffer
linux_dump_map	- Writes selected memory mappings to disk
linux_dynamic_env	- Recover a process' dynamic environment variables
linux_elfs	- Find ELF binaries in process mappings
linux_enumerate_files	- Lists files referenced by the filesystem cache
linux_find_file	- Lists and recovers files from memory
linux_getcwd	- Lists current working directory of each process
linux_hidden_modules	- Carves memory to find hidden kernel modules
linux_ifconfig	- Gathers active interfaces
linux_info_regs	- It's like 'info registers' in GDB. It prints out all the

linux_hidden_modules	- Carves memory to find hidden kernel modules
linux_ifconfig	- Gathers active interfaces
linux_info_regs	- It's like 'info registers' in GDB. It prints out all the
linux_iomem	- Provides output similar to /proc/iomem
linux_kernel_opened_files	- Lists files that are opened from within the kernel
linux_keyboard_notifiers	- Parses the keyboard notifier call chain
linux_ldrmodules	- Compares the output of proc maps with the list of libraries from libdl
linux_library_list	- Lists libraries loaded into a process
linux_librarydump	- Dumps shared libraries in process memory to disk
linux_list_raw	- List applications with promiscuous sockets
linux_lsmmod	- Gather loaded kernel modules
linux_lsof	- Lists file descriptors and their path
linux_malfind	- Looks for suspicious process mappings
linux_mmap	- Dumps the memory map for linux tasks
linux_moddump	- Extract loaded kernel modules
linux_mount	- Gather mounted fs/devices
linux_mount_cache	- Gather mounted fs/devices from kmem_cache
linux_netfilter	- Lists Netfilter hooks
linux_netscan	- Carves for network connection structures
linux_netstat	- Lists open sockets
linux_pidhashtable	- Enumerates processes through the PID hash table
linux_pkt_queues	- Writes per-process packet queues out to disk
linux_plthook	- Scan ELF binaries' PLT for hooks to non-NEEDED images
linux_proc_maps	- Gathers process memory maps
linux_proc_maps_rb	- Gathers process maps for linux through the mappings red-black tree
linux_procdump	- Dumps a process's executable image to disk
linux_process_hollow	- Checks for signs of process hollowing
linux_psaux	- Gathers processes along with full command line and start time
linux_psenv	- Gathers processes along with their static environment variables
linux_pslist	- Gather active tasks by walking the task_struct->task list
linux_pslist_cache	- Gather tasks from the kmem_cache
linux_psscan	- Scan physical memory for processes
linux_pstree	- Shows the parent/child relationship between processes
linux_psvview	- Find hidden processes with various process listings
linux_recover_filesystem	- Recovers the entire cached file system from memory
linux_route_cache	- Recovers the routing cache from memory
linux_sk_buff_cache	- Recovers packets from the sk_buff kmem_cache
linux_slabinf	- Mimics /proc/slabinfo on a running machine
linux_strings	- Match physical offsets to virtual addresses (may take a while, VERY verbose)
linux_threads	- Prints threads of processes
linux_tmpfs	- Recovers tmpfs filesystems from memory
linux_truecrypt_passphrase	- Recovers cached Truecrypt passphrases
linux_vma_cache	- Gather VMAs from the vm_area_struct cache
linux_volshell	- Shell in the memory image
linux_yarascan	- A shell in the Linux memory image

Introduction to the Volatility Framework

Task – Volatility 2	Windows Command	Linux Command
Identify OS Profile	imageinfo	
Check OS Version & Kernel		linux_banner
List Running Processes	pslist	linux_pslist
Process Tree View	pstree	linux_pstree
List Network Interfaces		linux_ifconfig
Extract Bash History		linux_bash
Enumerate Files Open in Memory		linux_enumerate_files
List Process IDs & Hashes		linux_pidhashtable
Show Process Memory Mappings		linux_proc_maps
Find Specific Files in Memory		linux_find_file
Dump Process Memory	procdump	linux_procdump
List Active Network Connections		linux_netstat
Scan Memory Using YARA Rules	yarascan	linux_yarascan
Detect Injected Malware	malfind	

Windows - <https://github.com/volatilityfoundation/volatility/wiki/Command-Reference>

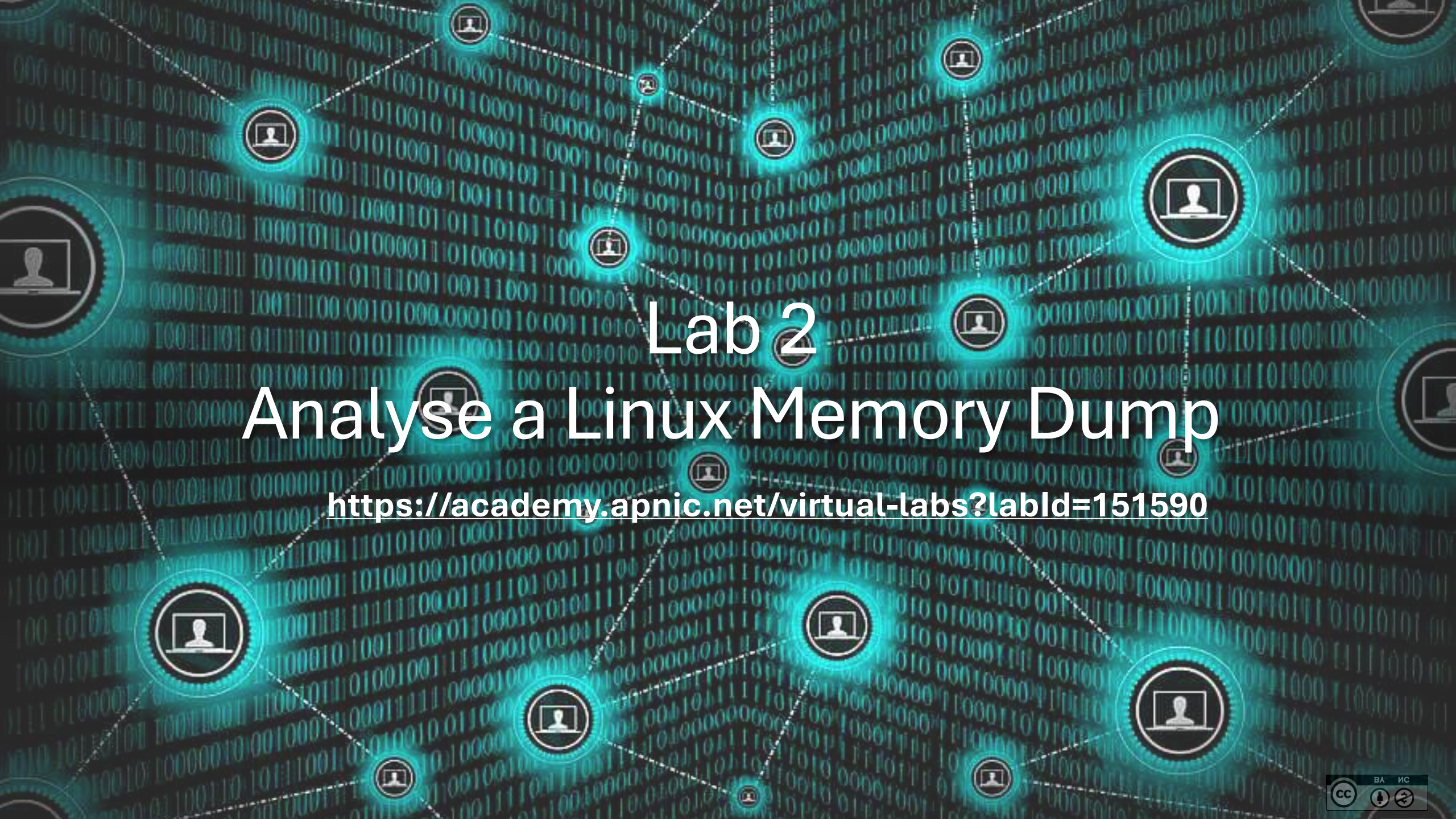
Linux - <https://github.com/volatilityfoundation/volatility/wiki/Linux-Command-Reference>

Introduction to the Volatility Framework

Task – Volatility 3	Windows Command	Linux Command
Identify OS Profile	windows.info.Info	
Check OS Version & Kernel		banners.Banners
List Running Processes	windows.pslist.PsList	linux.pslist.PsList
Process Tree View	windows.pstree.PsTree	linux.pstree.PsTree
List Network Interfaces		✗ No direct equivalent
Extract Bash History		linux.bash.Bash
Enumerate Files Open in Memory		linux.lsof.Lsof *
List Process IDs & Hashes		✗ No direct equivalent
Show Process Memory Mappings		linux.proc.Maps
Find Specific Files in Memory		✗ No direct equivalent
Dump Process Memory	windows.dumpfiles.DumpFiles *	linux.dumpfiles.DumpFiles * linux.proc.Maps
List Active Network Connections		✗ No direct equivalent
Scan Memory Using YARA Rules	yarascan.YaraScan windows.vadyarascan.VadYaraScan	yarascan.YaraScan
Detect Injected Malware	windows.malfind.Malfind	linux.malfind.Malfind

Windows - <https://github.com/volatilityfoundation/volatility/wiki/Command-Reference>

Linux - <https://github.com/volatilityfoundation/volatility/wiki/Linux-Command-Reference>

The background is a dark green field filled with a pattern of binary code (0s and 1s). Overlaid on this are several circular icons, each containing a white silhouette of a person at a computer. These icons are connected by a network of thin, light green lines, creating a web-like structure across the entire image.

Lab 2

Analyse a Linux Memory Dump

<https://academy.apnic.net/virtual-labs?labId=151590>

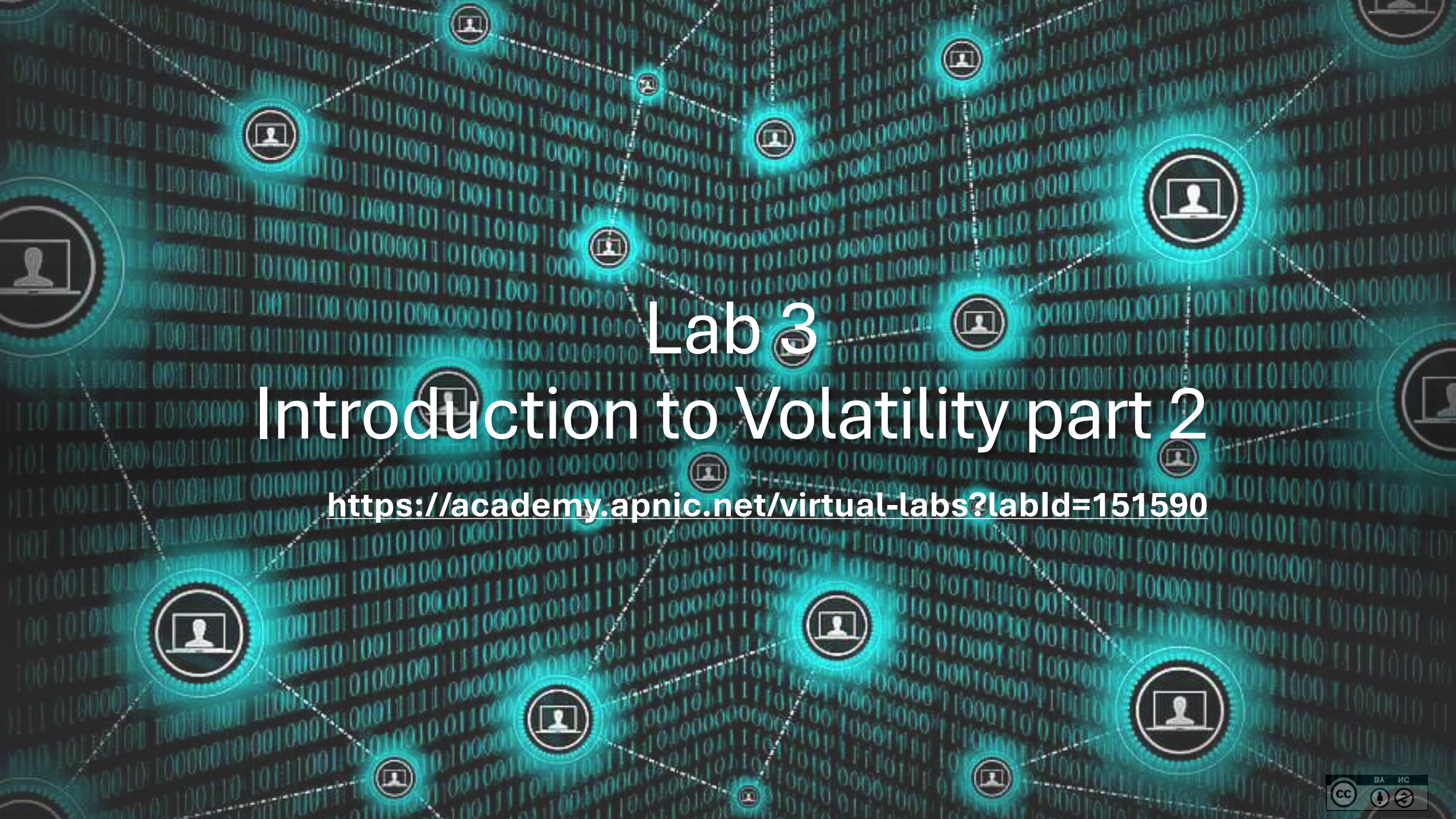
Lab 2 – Analyse a Linux memory dump

The steps to complete this section are:

- Analysis with Volatility
- Extract information about the system build and kernel version
- Explore Available Linux Commands
- Enumerate Active Processes
- Gather Network Interface Information
- Retrieve Command History
- List and Extract Open Files

Lab 2 – Analyse a Linux memory dump

Command	Description
<code>cd ~/memoryanalysis</code>	Changes to the memory analysis working directory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_banner</code>	Extracts system build and kernel version from memory.
<code>volatility --info grep -i Linux</code>	Lists available Linux plugins supported by Volatility.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_pslist</code>	Lists active processes found in memory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_pstree</code>	Displays the process tree to show parent-child relationships.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_ifconfig</code>	Shows network interface configuration information.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_bash</code>	Retrieves command history from memory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_enumerate_files</code>	Lists and extracts open file handles from memory.

The background is a dark teal color with a pattern of binary code (0s and 1s) in a lighter teal shade. Overlaid on this are several circular icons, each containing a white silhouette of a person at a computer, connected by a network of thin white lines. The icons and lines are also in a light teal color, creating a digital network effect.

Lab 3

Introduction to Volatility part 2

<https://academy.apnic.net/virtual-labs?labId=151590>

Lab 3 - Introduction to Volatility part 2

Analysis begins with questions. For example:

- What is this malware doing?
- What is it modifying?
- Are there any network activities?
- Did it create more process?
- Where is it hiding (if it has been deleted)?

Lab 3 - Introduction to Volatility part 2

The steps to complete this lab are:

- Find the process ID for krn.
- Locate how it was executed.
- Extract the file from the memory dump.
- Analyse network connections.
 - Use virustotal and check if the IP address is malicious
- Scan the memory dump for signs of malware.

Lab 3 - Introduction to Volatility part 2

Command	Description
<code>cd ~/memoryanalysis</code>	Changes to the memory analysis working directory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_pidhashtable</code>	Searches memory for process IDs via the PID hash table.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_proc_maps -p 1351</code>	Shows memory mappings of process with PID 1351 to see how it was executed.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_find_file -F "/home/analyst/workspace/malware/._lul/krn"</code>	Searches for the specified file path in memory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_find_file -F "/lib64/lib64/home/analyst/workspace/malware/._lul/krn"</code>	Attempts alternative file path search for the same file.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_find_file -i 0xffff88003d1004c8 -O dump.elf</code>	Extracts the memory-mapped file at the specified inode to disk.
<code>mkdir dump</code>	Creates a directory called 'dump' to store extracted files.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_procdump -D dump -p 1351</code>	Dumps the memory image of process 1351 to the 'dump' directory.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_netstat -p 1351</code>	Displays network connections associated with process 1351.
<code>head malware_rules.yar</code>	Displays the first few lines of the YARA rules file.
<code>volatility --plugins=. -f memUbuntu.raw --profile=LinuxUbuntu_4_4_0-21x64 linux_yarascan -y malware_rules.yar</code>	Scans the memory dump using the provided YARA rules to detect malware.

Lab 4 - Windows Memory Dump

<https://academy.apnic.net/virtual-labs?labId=151590>

Lab 4 - Windows Memory Dump

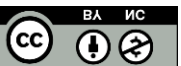
The steps to complete this lab are:

- Identify the profile to use.
- Find the suspicious process IDs.
- Scan the memory dump for signs of malware.
- Identify hidden and injected code in processes.
- Extract the file from the memory dump.
- Use virustotal to confirm if the file is malicious.

<https://en.wikipedia.org/wiki/Stuxnet>

<https://spectrum.ieee.org/the-real-story-of-stuxnet>

<https://css.csail.mit.edu/6.566/2018/readings/stuxnet.pdf>



Lab 4 - Windows Memory Dump

Command	Description
<code>cd ~/memoryanalysis</code>	Changes to the memory analysis working directory.
<code>volatility -f stuxnet.vmem imageinfo</code>	Identifies the recommended profile for the memory image.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 pstree</code>	Displays the process tree to show parent-child relationships.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 pstree grep -E "Name lsass"</code>	Filters the pstree output to locate the lsass process.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 pslist more</code>	Lists all active processes in a paginated view.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 pslist grep -E "Name 680"</code>	Searches for a specific process ID (680) in the process list.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 yarascan -y malware_rules.yar</code>	Scans memory using YARA rules to identify malware patterns.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 yarascan -y malware_rules.yar grep Rule -A 1</code>	Filters YARA scan results to show matched rules and context.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 malfind -p 1928</code>	Checks for injected or hidden code in process 1928.

Lab 4 - Windows Memory Dump

Command	Description
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 malfind -p 1928 grep -i write -B 1 -A 1</code>	Searches malfind output for writable regions of memory.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 malfind -p 1928,868,680 grep -i write -B 1 -A 1</code>	Searches for hidden/injected memory in multiple processes.
<code>mkdir stux_dump</code>	Creates a directory named 'stux_dump' for dumping memory contents.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 procdump -D stux_dump -p 1928,868</code>	Dumps the memory contents of processes 1928 and 868.
<code>file stux_dump/*</code>	Identifies file types of dumped memory segments.
<code>md5sum stux_dump/*</code>	Generates MD5 checksums for dumped files for integrity and malware scanning.
<code>sha256sum stux_dump/*</code>	Generates SHA-256 hashes for the dumped files.
<code>rm stux_dump/*</code>	Deletes all files in the 'stux_dump' directory.
<code>volatility -f stuxnet.vmem --profile=WinXPSP2x86 malfind -D stux_dump -p 1928,868</code>	Dumps suspicious memory regions detected by malfind into the stux_dump directory.

Lab 5 - Analysing hiberfit.sys and pagefile.sys

Draft – Feel free to have a go in your own time

Lab 5 - Analysing hiberfil.sys and pagefile.sys

The steps to complete this lab are:

- Convert and analyse hiberfil.sys .
- Scan pagefile.sys using external tools.
- Detect hidden malware with YARA rules.
- Extract and investigate relevant artefacts.

Lab 5 - Analysing hiberfil.sys and pagefile.sys

Command	Description
<code>volatility -f hiberfil.sys imagecopy -O mem_hiber.raw</code>	Converts `hiberfil.sys` to a raw memory image
<code>volatility -f mem_hiber.raw imageinfo</code>	Identifies the correct profile for analysis
<code>volatility -f mem_hiber.raw --profile=<profile> pslist</code>	Lists active processes
<code>volatility -f mem_hiber.raw --profile=<profile> pstree</code>	Shows the process tree of the system
<code>volatility -f mem_hiber.raw --profile=<profile> netscan</code>	Displays network connections from the memory image
<code>volatility -f mem_hiber.raw --profile=<profile> yarascan -y malware_rules.yar</code>	Applies YARA rules to scan for indicators of malware
<code>yarascan -r malware_rules.yar pagefile.sys</code>	Applies YARA rules to detect patterns in `pagefile.sys`
<code>bulk_extractor -o pagefile_output/ pagefile.sys</code>	Extracts data like URLs, emails, and keywords from the pagefile
<code>volatility -f full_mem_with_pagefile.raw --profile=Win10x64_18362 pslist</code>	Analyses a full memory dump that includes the pagefile

Memory Samples

Memory Samples

gleeda edited this page on Mar 23, 2019 · [8 revisions](#)

This is a list of publicly available memory samples for testing purposes.

Description	OS
Art of Memory Forensics Images	Assorted Windows, Linux, and Mac
Mac OSX 10.8.3 x64	Mac Mountain Lion 10.8.3 x64
Jackcr's forensic challenge	Windows XP x86 and Windows 2003 SP0 x86 (4 images)
GrrCon forensic challenge ISO (also see PDF questions)	Windows XP x86
Malware Cookbook DVD	Black Energy, CoreFlood, Laqma, Prolaco, Sality, Silent Banker, Tigger, Zeus, etc
Malware - Cridex	Windows XP SP2 x86
Malware - Shylock	Windows XP SP3 x86
Malware - R2D2 (pw: infected)	Windows XP SP2 x86
Windows 7 x64	Windows 7 SP1 x64
NIST (5 samples)	Windows XP SP2, 2003 SP0, and Vista Beta 2 (all x86)
Hogfly's skydrive (13 samples)	Assorted (mostly Windows XP x86)
Moyix's Fuzzy Hidden Process Sample	Windows XP SP3 x86
HoneyNet Banking Troubles Image	Windows XP SP2 x86
NPS 2009-M57 (~70 samples)	Various XP / Vista x86
Dougee's comparison samples	Windows XP x86
DFRWS 2008 Forensic Challenge	CentOS
HoneyNet Compromised Server Challenge	Linux Debian 2.6.26-26 x86
Pikeworks Linux Samples	Linux CentOS and Ubuntu (x86/x64)
DFRWS 2011 Forensics Challenge	Android
DFRWS 2012 Rodeo	Android

<https://github.com/volatilityfoundation/volatility/wiki/Memory-Samples>



Memory & Data-set Samples

Newest Data-Sets		
NEW	2025 MVS CTF	02/17/2025 at 22:20 Hexordia
NEW	Defaced web server (Ubuntu 22.04) (simulation)	02/16/2025 at 00:13 Benjamin Donnachie
NEW	Compromised Windows Server 2022 (simulation)	02/16/2025 at 00:10 Benjamin Donnachie
NEW	Reddit Cross-Topic Authorship Verification Corpus	02/11/2025 at 01:22 Oren Halvani
NEW	Large dataset of Sequential mobile App usage	02/11/2025 at 01:03 Mohammad Aliennejadi

NIST Computer Forensic Reference DataSet Portal
<https://cfreds.nist.gov/>

Summary

- What is Memory Forensics?
 - Examine volatile memory (RAM) to uncover processes, network connections, and system activity.
- The Volatility Framework
 - Open-source tool for memory analysis in incident response and malware investigations.
- Key Analysis Techniques
 - Identify active/hidden processes (pslist, psscan), process trees (pstree), and malware, suspicious files, or specific patterns in memory (yarascan).
- Real-World Applications
 - Used in cybersecurity investigations, CTF challenges, and malware detection.
- Why Use Volatility?
 - Detects threats missed by traditional methods, supports various memory dump formats, and provides extensive forensic insights.

References

- <https://github.com/Digitalisx/awesome-memory-forensics>
- <https://github.com/mesquidar/ForensicsTools>
- <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-86.pdf#page=25>
- <https://academy.apnic.net/virtual-labs?labId=151590>
- <https://tryhackme.com/room/btwindowsinternals>
- <https://tryhackme.com/room/volatility>
- <https://hadess.io/memory-forensic-a-comprehensive-technical-guide/>
- <https://fareedfauzi.github.io/2024/03/29/Linux-Forensics-cheatsheet.html>
- <https://blog.onfvp.com/post/volatility-cheatsheet/>