

MODULO 04 • S1.1 • AGOSTO 2026

Memory Forensics

La memoria volatile come fonte di prova:
processi, connessioni, chiavi e malware che il disco non mostra

`vol.py -f memdump.raw windows.pslist • windows.netscan • windows.malfind`

Corso di Laurea Magistrale in Sicurezza Informatica

Al termine del modulo lo studente sarà in grado di

01 • ACQUISIRE

Un dump di memoria con strumenti idonei (WinPmem, LiME), minimizzando e documentando l'impronta dell'operazione

02 • ANALIZZARE

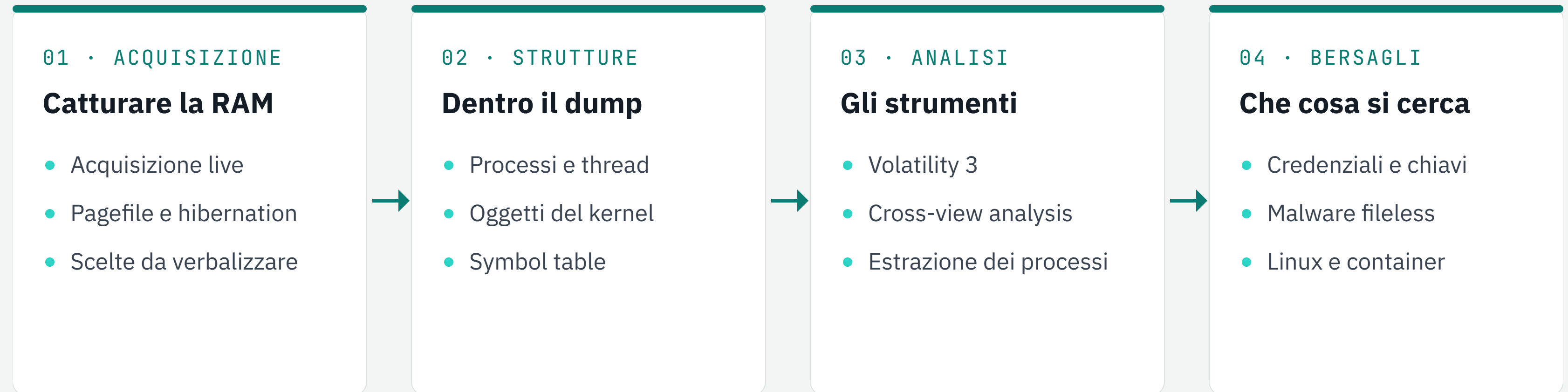
Processi, connessioni di rete, handle e moduli con Volatility 3 e MemProcFS, interpretando le strutture del kernel

03 • RILEVARE

Indicatori di compromissione in memoria: code injection, processi anomali, rootkit e credenziali esposte

Collocazione processuale. L'acquisizione della memoria è tipicamente un accertamento **irripetibile**, perché lo stato osservato non si può ricreare: solo la documentazione contestuale ne consente la verifica successiva.

Il flusso della memory forensics



Le quattro aree si susseguono nelle slide del modulo nello stesso ordine dello schema: ciascuna presuppone la precedente.

Osservare un sistema lo modifica: il paradosso da governare

Lo strumento di acquisizione viene caricato in memoria e ne altera una porzione: l'impronta è **inevitabile ma quantificabile**. Il perito la dichiara, scegliendo strumenti minimali e documentando versione, hash e momento dell'esecuzione.

Strumento	Ambito
WinPmem	Windows, output RAW/AFF4, open source
LiME	Linux, modulo kernel, dump locale o via rete
avml	Linux user-space, nessun modulo da caricare
Hibernation/pagefile	Fonti di memoria «fredde» su disco

Cosa vive solo in RAM

- Processi e thread in esecuzione
- Connessioni di rete attive
- Chiavi di cifratura dei volumi montati
- Comandi, appunti e credenziali in chiaro
- Malware fileless, privo di traccia su disco

Pagefile e hibernation: la RAM che resta sul disco

Parte della memoria finisce su disco. Il **pagefile.sys** contiene le pagine scaricate su disco dal gestore della memoria virtuale: frammenti di processi, stringhe, chiavi, anche di processi terminati da tempo. Il **hiberfil.sys** è un'istantanea compressa dell'intera RAM al momento dell'ibernazione, convertibile in formato analizzabile.

Conseguenza metodologica: anche quando la macchina è spenta e l'acquisizione live è impossibile, **una parte della memoria è recuperabile** dalla copia forense del disco. Il pagefile però è frammentario e privo di struttura, e si analizza per ricerca di pattern (carving, stringhe, regex).

Windows 10/11: anche swapfile.sys (app UWP) e la compressione della memoria complicano il quadro e forniscono altri dati.

ESEMPIO • LA CHIAVE NEL HIBERFIL.SYS

Un portatile sequestrato spento contiene un volume cifrato. Nessuna acquisizione live è stata possibile.

Nel hiberfil.sys, la RAM dell'ultima ibernazione, l'analista individua le strutture del driver di cifratura con la chiave ancora in memoria: il volume viene aperto senza conoscere la password. L'intera operazione avviene sulla copia di lavoro ed è ripetibile.

Strutture del kernel e symbol table

Un dump di memoria è una sequenza di byte senza indice. Volatility la interpreta perché conosce, per ciascuna versione del sistema, la **forma delle strutture del kernel**: dove inizia un oggetto processo, a quale offset sta il suo nome, dove il puntatore al successivo.

- **EPROCESS** (Windows): l'oggetto che rappresenta un processo: PID, nome, momento di avvio, collegamenti alla lista attiva
- **Symbol table (ISF)**: la «mappa» degli offset per quella precisa build del kernel; sbagliare mappa significa leggere strutture inesistenti
- **Traduzione degli indirizzi**: dalle pagine virtuali dei processi agli offset fisici del dump

ESEMPIO • PERCHÉ RILEVA IN RELAZIONE

Se il perito scrive «il processo X era in esecuzione», sta in realtà affermando: «l'interpretazione del dump secondo la symbol table della build Y mostra un oggetto EPROCESS coerente con X».

Dichiarare la versione del kernel e la symbol table usata serve a rendere l'affermazione verificabile da un terzo con gli stessi strumenti.

Conseguenza pratica: dump e metadati del sistema (versione OS, build) si acquisiscono insieme: un dump «anonimo» è interpretabile solo per tentativi.

Dal dump grezzo alle strutture del kernel

Volatility interpreta il dump attraverso i **symbol table** del sistema operativo, ricostruendo le strutture del kernel: lista dei processi, tabelle di rete, handle, moduli caricati.

```
windows.pslist / psscan → processi (anche nascosti)
windows.netscan → connessioni e socket
windows.cmdline → argomenti di avvio
windows.dlllist / handles → risorse aperte
windows.hashdump → credenziali locali
```

Metodo di lettura

- **pslist vs psscan**: la divergenza rivela processi scollegati dalle liste del kernel
- Gerarchie anomale: chi ha generato cosa, e da quale percorso
- Correlazione con la timeline del modulo 3: la RAM data gli eventi che il disco non registra

MemProcFS presenta gli stessi dati come file system navigabile.

Il confronto tra pslist e psscan

pslist • la vista «ufficiale»

Percorre la **lista doppiamente collegata** dei processi attivi. È la fotografia che il sistema mostra di sé, e che un rootkit può manipolare scollegando il proprio processo dalla catena (DKOM).

psscan • la vista «archeologica»

Scandisce **tutta la memoria fisica** cercando pattern di oggetti EPROCESS, collegati o no. Trova anche processi scollegati dalla lista e processi terminati i cui oggetti non sono ancora stati sovrascritti.

ESEMPIO • psscan elenca 87 processi; pslist ne mostra 86. Il processo mancante (PID 4211, nome quasi identico a un servizio di sistema) esiste in memoria ma il kernel «non lo vede»: pattern compatibile con un rootkit DKOM. Anche un processo *terminato da poco*, però, compare solo in psscan: l'ipotesi va verificata con malfind, handle e connessioni.

Corroborazione. Su un sistema potenzialmente compromesso si confrontano viste indipendenti, come per gli artefatti del modulo 3.

Estrarre un processo dal dump

1 • Individuare

pslist / psscan / pstree: nome, PID, genitore, orari di avvio. Le anomalie tipiche: nomi quasi legittimi (svch0st), genitori sbagliati, orari incoerenti con il logon.

2 • Contestualizzare

cmdline, dlllist, handles, netscan: con quali argomenti è partito, quali librerie e file teneva aperti, con chi comunicava.

3 • Estrarre

dumpfiles / pslist --dump: eseguibile e memoria del processo diventano file con hash propri, da analizzare staticamente o in sandbox. L'estrazione va verbalizzata come ogni derivazione.

ESEMPIO • Un «explorer.exe» ha una connessione attiva verso un IP estero sulla porta 443. dlllist rivela una DLL caricata da %TEMP%; il binario estratto ha un hash assente da NSRL e segnalato da più motori antimalware. In relazione: comandi eseguiti, hash del dump e dell'estratto, versione di Volatility, perché un altro consulente possa ripercorrere ogni passo.

Chiavi e credenziali in chiaro nella RAM

Per lavorare, il sistema deve tenere alcuni segreti in forma usabile. Finché il sistema è acceso, in memoria risiedono tipicamente:

- **Chiavi dei volumi cifrati montati** (BitLocker, VeraCrypt, LUKS): recuperabili dal dump con strumenti dedicati: è la ragione per cui la RAM si acquisisce *prima* di spegnere
- **Materiale di autenticazione** nel processo LSASS (Windows): hash e, in configurazioni datate, credenziali riutilizzabili
- **Contenuti effimeri**: appunti, campi compilati, chat decifrate dall'applicazione

ESEMPIO • IL VOLUME CIFRATO

Perquisizione: il laptop è acceso, un volume VeraCrypt montato. Si acquisisce la RAM; il sistema viene poi spento e il disco copiato. Dal dump si estrae la master key del volume, che consente di decifrare la copia forense del disco, operazione impossibile se si fosse spento per primo.

L'acquisizione si documenta ex art. 360 c.p.p.: lo stato della RAM non è riproducibile.

La difesa può chiedere da quale dump e con quali hash è stata ottenuta la chiave: il perito deve poterlo documentare.

Memory forensics su Linux e container

Gran parte degli incidenti aziendali avviene su server Linux.

L'acquisizione usa moduli kernel come **LiME** o strumenti come AVML: ma il modulo va **compilato per il kernel esatto** della macchina, e caricarlo modifica lo stato del sistema: una scelta da motivare e verbalizzare, come ogni operazione live.

L'analisi con Volatility 3 richiede un **symbol table** coerente con il kernel (l'equivalente dei profili): senza, il dump è illeggibile. I plugin ricostruiscono processi, connessioni, moduli caricati, bash history in memoria. Nei **container** i processi appaiono nel kernel dell'host: un solo dump copre tutti i container in esecuzione.

Nel cloud lo snapshot della VM offerto dal provider è spesso la via più pulita: evita di caricare moduli e richiede di documentare la catena di custodia con il provider.

ESEMPIO • IL ROOTKIT NEL MODULO KERNEL

Su un web server compromesso, ps e netstat non mostrano nulla di anomalo: il rootkit filtra le risposte del kernel.

Il dump LiME, letto da Volatility, elenca un modulo kernel assente dalla distribuzione e un processo nascosto con connessione persistente. Il dump, letto «dall'esterno», non dipende dalle API manipulate; il confronto con l'output di ps e netstat applica a Linux la cross-view analysis.

Dove il malware non può mentire

Code injection

Codice eseguibile in regioni di memoria anomale di processi legittimi. **windows.malfind** individua pagine eseguibili non riconducibili a moduli su disco.

Process masquerading

Un processo che usa il nome di un processo di sistema si riconosce dal genitore sbagliato, dal percorso anomalo o da un'istanza duplicata.

Rootkit e hooking

Manipolazione delle strutture del kernel per occultare processi e connessioni: il confronto tra viste indipendenti (cross-view) li fa riemergere.

Integrazione con YARA. Le regole YARA eseguite sul dump ([vol.py yarascan](https://github.com/VolatilityFoundation/volatility3)) individuano pattern noti direttamente in memoria; i segmenti sospetti estratti si analizzano poi con Ghidra, ponte verso l'analisi malware.

Una compromissione senza eseguibili su disco

Il malware **fileless** evita di scrivere eseguibili su disco: vive in memoria, si inietta in processi legittimi e persiste con mezzi «leciti» (attività pianificate, chiavi di avvio, sottoscrizioni WMI) che richiamano codice da remoto o da blob offuscati nel registry.

- **Vettore tipico:** script (PowerShell, mshta) lanciato da documento o link
- **Esecuzione:** payload decodificato e iniettato direttamente in RAM
- **Persistenza:** script e comandi nel registry o nel task scheduler

Conseguenza forense: l'antivirus su disco e la scansione dei file possono non trovare nulla; **la memoria è l'unico luogo dove il malware esiste per intero.**

ESEMPIO • LA CATENA RICOSTRUITA

Dal dump: **cmdline** mostra PowerShell avviato con flag di esecuzione nascosta e un argomento codificato; **malfind** trova pagine eseguibili anomale in un processo di sistema; **netscan** rivela la connessione periodica verso l'esterno.

Sul disco non risulta alcun file malevolo: la relazione documenta la compromissione sulla base delle tre evidenze di memoria.

Analisi di un dump compromesso

Laboratorio

- Dump pubblico in **Volatility 3**
- Processo malevolo e connessioni
- Estrazione del payload e YARA

DELIVERABLE: scheda IoC
motivata

Lecture obbligatorie

- Ligh et al., *The Art of Memory Forensics* (capitoli su Windows)
- Documentazione Volatility 3
- Scheda tecnica MemProcFS

Verifica

Si valuta la catena inferenziale: dall'anomalia osservata all'attribuzione, ogni passaggio esplicito e falsificabile.

Prossimo modulo: S1.2 Incident response e intrusioni su host.

L'acquisizione della memoria è irripetibile e la qualità del dump condiziona ogni analisi successiva.



Grazie per l'attenzione

Vincenzo Calabrò

Domande, materiali del corso e contatti
sul sito del docente



vincenzocalabro.it