ÁMBAR

Documentación técnica

FRAMEWORK DE ADQUISICIÓN FORENSE · VERSIÓN 3.0

Euclides J. Ángeles Alcántara

Tecnólogo en Informática Forense

Instituto Tecnológico de las Américas (ITLA)

Santo Domingo, República Dominicana

DOCUMENTO TÉCNICO · ES

CONTENIDO

Índice general

Este documento describe el framework en el orden en que se ejecuta: primero el arranque y la detección del entorno, después cada módulo de adquisición, y al final los servicios transversales que determinan si la evidencia es defendible.

CAPÍTULO 1

Qué es el framework y qué problema resuelve

Antes de entrar en comandos conviene situar el script: qué cubre, qué decisiones lo atraviesan de principio a fin, y cómo está organizado por dentro.

1.1 Alcance funcional

Ámbar es un framework de adquisición forense para Windows que reúne, en una sola interfaz de consola, la captura de evidencia digital procedente de seis clases distintas de origen.

CLASE DE ORIGEN	QUÉ SE ADQUIERE	HERRAMIENTAS EMPLEADAS
Memoria del host	Volcado crudo completo de la RAM física	DumpIt, WinPmem
Artefactos del sistema	Registro, eventos, prefetch, LNK, navegadores y demás targets del repositorio local	KAPE
Almacenamiento físico	Imagen bit a bit de discos y particiones	dc3dd, dd, ewfacquire, FTK Imager
Sistema de archivos	Copia forense selectiva preservando metadatos NTFS	robocopy
Máquinas virtuales	Discos, memoria, snapshots o checkpoints, y clones completos	vmrun, vmware-vdiskmanager, VBoxManage, cmdlets de Hyper-V, qemu-img
Contenedores y móviles	Sistema de archivos, volúmenes y artefactos dirigidos	docker, adb, 7zr

Sobre esa base hay una capa de servicios transversales que no adquieren evidencia pero deciden si vale algo: verificación de integridad por hash, protección contra interrupciones accidentales, registro exhaustivo de cada comando ejecutado, validación de rutas y un write blocker por software.

1.2 Los siete principios de diseño

Siete decisiones atraviesan el framework completo. Entenderlas por adelantado explica la mayor parte de lo que aparece en los capítulos siguientes.

a) Degradación elegante, nunca abortar

Ninguna ausencia detiene el framework. Sin privilegios de administrador avisa y continúa con funcionalidad limitada. Sin PowerShell cae a WMIC, y de ahí a variables crudas. Sin HashMyFiles usa `certutil`. Sin el módulo de aislamiento de `CTRL+C` corre en modo degradado. Sin `wmic.exe`, retirado en las builds recientes de Windows 11, vuelve a PowerShell. Sin el flag `/J` de robocopy, inexistente en Windows 7, lo omite.

La consecuencia práctica es que el script arranca y trabaja en un abanico de sistemas que va de Windows 7 a Windows 11, sin compilar nada y sin instalar dependencias.

b) El operador decide, el script documenta

El hash nunca se calcula por su cuenta: se pregunta. Suspender o apagar una máquina virtual exige confirmación explícita. Convertir a RAW se ofrece, no se impone. Continuar con poco espacio en destino es una decisión que queda registrada.

> POR QUÉ IMPORTA
>
> Cuando el operador dice «no», esa negativa también se escribe en el log. En un procedimiento forense la ausencia de un hash es un dato tan relevante como su presencia, y tiene que quedar constancia de que fue una decisión deliberada y no un fallo silencioso de la herramienta.

c) La evidencia original nunca se modifica

Cada operación que necesita escribir sobre los archivos fuente, como consolidar una cadena de snapshots o fusionar discos diferenciales, se ejecuta primero sobre una copia de trabajo. Se hashea el original antes de copiarlo, se opera sobre la copia, y se hashea el resultado como evidencia derivada, vinculada al hash del original.

Las tres tecnologías de virtualización implementan el mismo patrón con nombres distintos: VMware con `_trabajo_<snap>`, Hyper-V con `Consolidation\Working`, y VirtualBox con un clon temporal que se desregistra

al terminar.

d) Todo comando pesado es cancelable con confirmación

Cualquier herramienta que pueda tardar horas se ejecuta a través de `:ACQUIRE`, que delega en `forensic_wrapper.ps1`. Un `CTRL+C` no mata el proceso: lo suspende, muestra un diálogo explicando que abortar dejará la evidencia incompleta, y solo termina si el operador lo confirma. El contrato de salida es siempre el mismo.

```
0   éxito
1   error
2   abortado por el operador
```

e) Trazabilidad en texto plano, sin formatos propietarios

Hay dos registros. Uno central, que guarda íntegro cada comando externo, en `Logs\Forensic_Commands.log`. Y uno por evidencia, que viaja junto a ella, en `<DEST>\Forensic_Journal.log`. Ambos en texto plano legible sin ninguna herramienta.

Una versión anterior mantenía además un log JSONL encadenado. Se retiró de forma deliberada, y `:LOG_CENTRAL` quedó como no-op para no romper las llamadas ya escritas en el resto del código.

f) El formato real de cada artefacto se documenta junto a la evidencia

Cuando la salida no es lo que su nombre sugiere, el script deja un `FORMAT_NOTE.txt` al lado del archivo. Ocurre en dos sitios: el volcado de VirtualBox es un ELF core dump y no memoria RAW, y el estado guardado de Hyper-V (`.vmrs`, o `.bin` más `.vsv`) tampoco es RAW, de modo que Volatility 3 no lo abre sin conversión previa.

En ambos casos la nota incluye la sintaxis exacta de análisis y las vías de conversión, para que el conocimiento no dependa de que alguien lo recuerde meses después.

g) Cada control defensivo responde a un fallo observado

Los comentarios del código distinguen explícitamente qué controles se añadieron después de reproducir un fallo real, y no como precaución teórica. Varios se detallan en su capítulo correspondiente:

- el `.vmem` que aparecía después de que `vmrun` ya hubiera retornado
- el volcado de RAM de cero bytes que terminaba con código de salida cero

- el separador `=` en la salida de `--machinereadable`
- la variable de estado que sobrevivía entre iteraciones del mismo menú
- el nombre de archivo temporal fijo que dos instancias concurrentes se pisaban
- la salida vacía, indistinguible entre «carpeta vacía» y «conexión perdida»

1.3 Cómo está organizado el archivo

El script es un único `.bat` con `setlocal enabledelayedexpansion` activo desde la segunda línea, estructurado en etiquetas alcanzadas por `goto` y `call`. Su disposición sigue el orden de ejecución.

BLOQUE	CONTENIDO
Cabecera	Colores ANSI condicionales, maximización de ventana, geometría de consola y cadenas de relleno para centrar los menús
Preparación	Auto-elevación, definición centralizada de todas las rutas de herramientas, cadena de relanzamiento y llamada a la pantalla de arranque
Menús	Menú principal, write blocker y visor del registro de comandos
Memoria y archivos	Volcado de RAM y adquisición lógica de archivos y carpetas
Discos	Detección del origen, validaciones previas, las cuatro herramientas de imaging y la verificación de dc3dd
Integridad	`HASH_EVIDENCE`, la biblioteca de verificación
Subrutinas de apoyo	Política de snapshots y validación de rutas
Triaje	Integración con KAPE
Virtualización	VirtualBox, VMware e Hyper-V, cada uno con sus subrutinas de resolución de binario, discos, memoria y clon completo
Cierre	Conversión con QEMU, Docker, ADB, utilidades compartidas y pantalla de arranque

NOTA SOBRE ESTA DOCUMENTACIÓN

Los bloques se describen por función y no por número de línea a propósito. Cualquier cambio en el código desplaza las líneas y deja obsoleta una tabla de rangos, mientras que la organización lógica se mantiene estable.

CAPÍTULO 2

Arranque: elevación, relanzamiento y detección de capacidades

Antes de dibujar el primer menú, el script recorre una secuencia de preparación que decide cómo se verá la interfaz, con qué privilegios correrá, si CTRL+C quedará bajo control y qué módulos estarán habilitados. Este capítulo la sigue en orden de ejecución.

2.1 Cabecera: colores, geometría de consola y rutas

Preámbulo

```
@echo off
setlocal enabledelayedexpansion
break off
title AMBAR v3.0 - Framework de Adquisicion Forense
```

- `@echo off`: silencia el eco de cada comando. Sin esto la interfaz sería ilegible.
- `setlocal enabledelayedexpansion`: habilita la sintaxis `!variable!`. Es imprescindible porque el script asigna y lee variables dentro del mismo bloque y dentro de bucles `for`, donde `%variable%` se resolvería una sola vez al parsear el bloque.
- `break off`: herencia histórica de la gestión de CTRL+C. El control real lo aportan los dos módulos PowerShell.
- `title`: identifica la ventana en la barra de tareas, útil cuando hay varias sesiones abiertas.

Activación condicional de colores ANSI

```
for /f "tokens=3" %%a in ('reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" ^
                          /v CurrentBuildNumber 2^>nul') do set "_earlybuild=%%a"
if defined _earlybuild if !_earlybuild! GEQ 10586 set "ANSI_OK=1"
if not defined ANSI_OK goto SKIP_ANSI
for /F %%a in ('echo prompt $E ^| cmd') do set "ESC=%%a"
set "C_TITLE=!ESC![1;36m"   :: cian brillante  - títulos
set "C_OK=!ESC![1;32m"      :: verde           - éxito
set "C_ERR=!ESC![1;31m"     :: rojo            - error
set "C_WARN=!ESC![1;33m"    :: amarillo        - advertencia
set "C_ACCENT=!ESC![1;35m"  :: magenta         - encabezados
set "C_DIM=!ESC![0;90m"     :: gris            - texto secundario
set "C_RESET=!ESC![0m"
```

El bloque consulta la build de Windows en el registro y solo define los códigos de color si es `10586` o superior (Windows 10, versión 1511), la primera que interpreta secuencias VT en la consola. El carácter de escape (0x1B) se obtiene con el truco `echo prompt $E | cmd`, porque el lenguaje de lotes no tiene manera directa de escribir un byte arbitrario.

> POR QUÉ IMPORTA
>
> En Windows 7 y 8 la consola no interpreta VT: los códigos se imprimirían literalmente como `←[1;36m` y la interfaz quedaría ilegible. Dejando las variables **vacías** en lugar de bifurcar el código, cada `echo !C_OK!texto!C_RESET!` sigue funcionando sin cambios y simplemente sale monocromo. Un solo camino de código para todas las versiones de Windows.

Maximización de ventana y geometría de menús

```
powershell -Command "Add-Type -TypeDefinition '… ShowWindow … GetConsoleWindow …';
                    [Win32]::ShowWindow([Win32]::GetConsoleWindow(), 3)"

for /f "tokens=2" %%a in ('mode con ^| findstr /i /R "Col"') do set "W_COLS=%%a"
if "!W_COLS!"=="" set "W_COLS=120"
set /a LINE_LEN=W_COLS - 1
for /l %%i in (1,1,!LINE_LEN!) do set "LINE=!LINE!="
set /a PAD_MAIN=(W_COLS - 36) / 2
```

Se llama a `ShowWindow(hWnd, 3)` sobre el handle de la consola, donde el 3 es `SW_MAXIMIZE`. Después se lee el número real de columnas con `mode con` y se construye por bucle una línea separadora de ese ancho, más seis cadenas de

relleno (`PAD_STR_MAIN`, `PAD_STR_VOL`, `PAD_STR_TRI`, `PAD_STR_ART`, `PAD_STR_LIST`, `PAD_STR_VM`) calculadas como (ancho menos largo del título) dividido entre dos, para centrar cada tipo de encabezado.

> **POR QUÉ IMPORTA**
>
> Los listados de discos, de VMs y de rutas largas se truncan o se envuelven en una ventana estrecha, y en un contexto forense eso significa que el operador podría leer mal una ruta de destino. El valor por defecto de 120 columnas cubre el caso de que `mode con` no devuelva nada. La maximización solo se ejecuta en el primer proceso, comprobando `FORENSIC_READY`, porque los dos relanzamientos posteriores heredan la misma ventana ya maximizada.

Definición centralizada de rutas

```
set "SYS_DIR=%~dp0Sistema"
set "BASE_PATH=!SYS_DIR!\herramientas"
set "KAPE=!BASE_PATH!\Triaje\kape\kape.exe"
set "OPENCONSOLE=!BASE_PATH!\Terminal\terminal-1.24.11321.0\OpenConsole.exe"
set "QEMU_IMG=!BASE_PATH!\Virtualizacion\QEMU\qemu-img.exe"
set "SEVENZIP=!BASE_PATH!\Utilidades\7zip\7zr.exe"
set "ADB_EXE=!BASE_PATH!\platform-tools\adb.exe"
set "WRAPPER_PS1=!SYS_DIR!\lib\forensic_wrapper.ps1"
set "HV_RAW_PS1=!SYS_DIR!\lib\hv_savedstate_to_raw.ps1"
for %%F in ("%~dp0..\Evidencias") do set "DEFAULT_DEST=%%~fF"
```

Cada ruta de herramienta se declara una sola vez y siempre relativa a la carpeta del script.

Todas las rutas se derivan de `%~dp0`, la carpeta del propio script, así que el framework es **portátil**: funciona desde una unidad USB con cualquier letra asignada. El destino por defecto de evidencias se resuelve con `%%~fF` a ruta absoluta, eliminando el `..`, porque muchas herramientas rechazan rutas relativas.

El binario de hashing se elige por arquitectura del sistema operativo, comprobando además `PROCESSOR_ARCHITEW6432`, la variable que Windows define cuando un proceso de 32 bits corre en un host de 64 (WOW64) y donde `PROCESSOR_ARCHITECTURE` mentiría diciendo `x86`:

```
if /i "%PROCESSOR_ARCHITECTURE%"=="AMD64" set "_IS_X64=1"
if defined PROCESSOR_ARCHITEW6432 set "_IS_X64=1"
if defined _IS_X64 ( set "HMF_EXE=…\hashmyfiles-x64\HashMyFiles.exe" ) else ( set "HMF_EXE=…
\x32\HashMyFiles.exe" )
if not exist "!HMF_EXE!" set "HMF_EXE=…\x32\HashMyFiles.exe"   :: reserva
```

2.2 Auto-elevación de privilegios

Detección de elevación y relanzamiento con UAC

```
net session >nul 2>&1
if !errorLevel! neq 0 (
    echo [-] ADVERTENCIA: Ejecutando sin privilegios de Administrador.
    for %%P in (powershell.exe) do if not "%%~$PATH:P"=="" set "_PSCHK=1"
    if defined _PSCHK (
        powershell -Command "Start-Process -FilePath '%~f0' -Verb RunAs"
        exit /b
    ) else (
        echo [i] Los modulos de adquisicion de disco/RAM requieren Administrador.
        echo [i] Continuando con funcionalidad limitada (degradacion elegante)...
        timeout /t 3 >nul
    )
)
```

`net session` consulta las sesiones del servicio Server, operación que *solo* tiene éxito con privilegios administrativos: es la prueba de elevación clásica en lotes, sin dependencias. Si falla y PowerShell está disponible, el script se relanza con `-Verb RunAs`, que dispara el diálogo de UAC, y el proceso actual sale.

El chequeo de PowerShell usa `%%~$PATH:P`, un modificador de `for` que devuelve la ruta completa del ejecutable si se encuentra en el `%PATH%` y una cadena vacía si no. Es el mismo idioma que emplea el script para detectar `wmic`, `docker`, `adb`, `vmrun`, `VBoxManage` y `wsl`.

> POR QUÉ IMPORTA
>
> Muchas funciones no necesitan privilegios: la conversión con QEMU, el hashing, la adquisición lógica con robocopy sobre carpetas del usuario, la enumeración de contenedores. Abortar por falta de elevación bloquearía trabajo perfectamente válido. Lo que se hace es avisar de que los módulos de disco y RAM sí la requieren, y dejar que el operador decida.

2.3 La cadena de relanzamiento en dos saltos

El script se relanza hasta dos veces antes de mostrar el menú. El estado viaja en **variables de entorno**, que los procesos hijos heredan, y no en argumentos de línea de comandos, precisamente para sobrevivir a saltos

sucesivos sin ir arrastrando parámetros.

1. **Salto 1: terminal moderno.** Si existe `OpenConsole.exe` y no se ha hecho ya (`FORENSIC_IN_CONSOLE`), el script se relanza dentro de él:

   ```
   set "FORENSIC_IN_CONSOLE=1"
   "!OPENCONSOLE!" cmd /c "%~f0"
   exit /b
   ```

   Se usa `cmd /c` y no `/k` a propósito: con `/k`, al salir del framework quedaría un prompt de cmd abierto en `system32`. Con `/c` la ventana se cierra limpia. Si `OpenConsole` no está, el paso se omite sin más.

2. **Salto 2: aislamiento de CTRL+C.** Si hay PowerShell y existe `launch_noctrlc.ps1`:

   ```
   set "FORENSIC_READY=1"
   powershell -NoProfile -ExecutionPolicy Bypass -File
   "!SYS_DIR!\lib\launch_noctrlc.ps1" -Bat "%~f0"
   if not "!errorlevel!"=="99" exit /b
   echo [i] Aislamiento de CTRL+C no disponible. Continuando en
   modo degradado.
   ```

   El lanzador devuelve el código real del `cmd` hijo, o exactamente **99** si su propia inicialización falló (`Add-Type` o `CreateProcess`). Ese 99 es la señal convenida para que el `.bat` siga adelante sin aislamiento en lugar de quedarse sin interfaz.

3. **Entrada al framework.** Con `FORENSIC_READY` definida, el salto `goto AFTER_RELAUNCH` evita repetir la secuencia, se llama a `:STARTUP_SCREEN` y se entra al menú principal.

El *por qué* de esta arquitectura de dos capas se explica completo en el capítulo 15. Aquí basta con retener el efecto: durante una adquisición, CTRL+C ni mata la herramienta ni saca del menú.

2.4 La pantalla de verificación, comprobación por comprobación

`:STARTUP_SCREEN` imprime una fila por comprobación a través de `:BOOT_ROW`, que rellena la etiqueta a 42 caracteres para alinear la columna de detalle y admite cuatro estados: `[ OK ]`, `[WARN]`, `[ -- ]` (omitido, que no supone un problema) y `[FAIL]`. Dos contadores acumulan el resultado (`_bo` correctos, `_bw` advertencias) y alimentan el resumen final y la cabecera del menú principal.

Bloque [Sistema]

```
reg query "…\CurrentVersion" /v CurrentMajorVersionNumber   :: Win10+ (hexadecimal)
reg query "…\CurrentVersion" /v CurrentMinorVersionNumber
reg query "…\CurrentVersion" /v CurrentBuildNumber
reg query "…\CurrentVersion" /v CurrentVersion               :: reserva para Win7/8
```

Los valores mayor y menor se convierten con `set /a` porque el registro los devuelve en hexadecimal. Con ellos se compone `WIN_NT` y se traduce a nombre comercial:

WIN_NT	NOMBRE	WIN_NT	NOMBRE
5.0	Windows 2000	6.2	Windows 8
5.1	Windows XP	6.3	Windows 8.1
5.2	Windows XP x64 / 2003	10.0	Windows 10
6.0	Windows Vista	10.0 con build 22000 o superior	Windows 11
6.1	Windows 7		

> NOTA
>
> Windows 11 se identifica por **build** y no por versión: Microsoft mantuvo `10.0` como versión NT, así que la única señal fiable es un `CurrentBuildNumber` de 22000 o superior. Esto importa más allá de la estética: la build también gobierna los colores ANSI (10586 o superior) y el flag `/J` de robocopy (9200 o superior).

Bloque [Herramientas forenses]

```
powershell -NoProfile -Command "$sb='NA';
    try { if (Confirm-SecureBootUEFI) { $sb='ON' } else { $sb='OFF' } } catch {};
    Write-Output ([string]$PSVersionTable.PSVersion.Major + ';' + $sb)"
```

Una sola invocación devuelve dos datos separados por punto y coma: la versión mayor de PowerShell y el estado de Secure Boot. Se hace así porque arrancar PowerShell cuesta cientos de milisegundos y dos llamadas duplicarían ese coste en cada arranque. La sintaxis usa `try/catch` como sentencia y no como expresión, para mantener compatibilidad con PowerShell 2.0: `Confirm-SecureBootUEFI` lanza excepción en máquinas BIOS/Legacy, y el `catch` vacío deja el valor en `NA`.

El resto del bloque comprueba presencia: `wmic.exe` y `dism.exe` en el PATH; `HashMyFiles`, `kape.exe`, `qemu-img.exe` y `forensic_wrapper.ps1` por existencia de archivo. La ausencia de HashMyFiles o del wrapper es **advertencia**, porque afecta a la calidad del proceso; la de KAPE o QEMU es **omitido**, porque solo desactiva un módulo opcional.

Bloque [Hipervisores y contenedores]

COMPROBACIÓN	MÉTODO	MOTIVO DE LA ELECCIÓN
Hyper-V	`sc query vmms`	Consulta instantánea del servicio. `Get-Command Get-VM` cargaría el módulo completo de Hyper-V y tardaría varios segundos en cada arranque.
VMware	`vmrun.exe` embebido o en PATH	El framework trae su propia copia; también detecta una instalación del sistema.
VirtualBox	`VBoxManage.exe` embebido o en PATH	Igual criterio; la resolución completa en cascada se hace al entrar al módulo.
Docker	`docker.exe` en PATH	La comprobación del *daemon* se posterga al módulo, con `docker version`.
WSL 2	`wsl.exe` en PATH	Informativo: documenta el entorno del host en el log de capacidades.
ADB	`adb.exe` embebido o en PATH	Si se encuentra en el PATH, `ADB_EXE` se reapunta a esa ruta.

Bloque [Subsistemas] y cierre

```
if "!_sb!"=="ON"  (set "FIRMWARE=UEFI" & set "SECUREBOOT=Activado")
if "!_sb!"=="OFF" (set "FIRMWARE=UEFI" & set "SECUREBOOT=Desactivado")
if "!_sb!"=="NA" ( bcdedit /enum {current} 2^>nul | findstr /i "winload.efi" ^>nul ^&^& set
"FIRMWARE=UEFI" )

set "RBC_J="
if defined WIN_BUILD if !WIN_BUILD! GEQ 9200 set "RBC_J=/J"

call :LOG_INIT
call :LOG_CENTRAL "INFO" "system_caps" "OS:… NT:… Build:… Arch:… PS:… WMIC:… Admin:… HyperV:… …"
```

Si `Confirm-SecureBootUEFI` no dio dato, se recurre a `bcdedit`: la presencia de `winload.efi` en la entrada de arranque actual indica arranque UEFI. Es una detección mínima, pero suficiente para documentar el firmware del sistema examinado.

La variable `RBC_J` merece atención porque se usa en cinco flujos distintos. El flag `/J` de robocopy activa E/S sin buffer, ideal para archivos gigantes como `.vhdx` o `.vmem`, pero **no existe antes de Windows 8** (build 9200). Guardándolo en una variable que se expande vacía en sistemas antiguos, el mismo comando funciona en todas las versiones.

Por último, `:LOG_INIT` crea `Script\Logs`, resuelto como `Sistema\..\..\Logs`, y se registra la línea de capacidades del sistema.

2.5 Menú principal y validación de entrada

La cabecera del menú refleja el resultado del arranque: si no hubo advertencias muestra en verde el número de componentes verificados junto al sistema operativo, la arquitectura y el nombre del equipo; si las hubo, muestra en amarillo el recuento e invita a revisar el log. Las opciones de Docker y ADB se etiquetan en caliente:

```
set "_tgDK=!C_DIM! (no detectado)!C_RESET!" & if defined HAS_DOCKER set "_tgDK=!C_OK!
(disponible)!C_RESET!"
set "_tgADB=!C_DIM! (no detectado)!C_RESET!" & if defined HAS_ADB set "_tgADB=!C_OK!
(disponible)!C_RESET!"
```

Y al elegirlas se vuelve a comprobar la capacidad antes de entrar, con un mensaje que explica el requisito concreto en lugar de dejar que el módulo falle por dentro:

```
if "!opt!"=="5" (
    if not defined HAS_DOCKER (
        echo [-] Docker no esta disponible en este sistema.
        echo     Requiere Docker Desktop/Engine instalado y en el PATH (Windows 10/11 x64).
        call :LOG_CENTRAL "WARN" "module_unavailable" "Docker seleccionado pero no disponible"
        pause
        goto MENU_PRINCIPAL
    )
    goto MENU_DOCKER
)
```

El patrón de validación numérica

```
for /f "delims=0123456789" %%a in ("!opt!") do (
    echo [-] Opcion invalida.
    timeout /t 2 >nul
    goto MENU_PRINCIPAL
)
```

NOTA

`delims=0123456789` declara **todos los dígitos como delimitadores**. Si la entrada es puramente numérica, no queda ningún token y el cuerpo del `for` nunca se ejecuta. Si contiene cualquier otro carácter, ese carácter forma un token, el bucle entra y rechaza la entrada. Es una validación de «solo dígitos» sin expresiones regulares ni PowerShell, y se repite en **todos** los menús y selectores del framework. Los selectores de lista añaden después los límites de rango: `if !sel! lss 1 goto …` y `if !sel! gtr !count! goto …`.

Otra convención uniforme: `0` **siempre significa cancelar o volver**, en todos los menús, prompts de ruta y selectores del framework. El operador nunca queda atrapado en un flujo.

CAPÍTULO 3

Módulo 1. Volcado de memoria RAM

Primer eslabón del orden de volatilidad. Tres herramientas intercambiables, selección automática de binario por arquitectura, y dos controles de calidad que evitan entregar una evidencia inútil.

3.1 Por qué la memoria va primero

El **orden de volatilidad** definido en la RFC 3227 establece que la evidencia debe recogerse de lo más volátil a lo más persistente. La RAM contiene información que *no existe en ningún otro sitio* y desaparece al apagar el equipo: procesos en ejecución, conexiones de red activas, claves de cifrado en claro, contenido descifrado de contenedores montados, comandos ejecutados en consolas abiertas, malware que nunca toca el disco. Cualquier operación previa sobre el disco, incluso montar una unidad externa para volcar, modifica la RAM. De ahí que sea la opción 1 del menú.

3.2 Configuración del destino

```
:CONFIG_VOLCADO_ASK
set "DEST=!DEFAULT_DEST!"
set /p DEST="Ruta de destino (por defecto: !DEFAULT_DEST!): "
if defined DEST set "DEST=!DEST:"=!"          :: elimina comillas pegadas al pegar rutas
if "!DEST!"=="" set "DEST=!DEFAULT_DEST!"
call :VALIDATE_DEST_DIR "!DEST!"
if not "!_VD_OK!"=="1" goto CONFIG_VOLCADO_ASK   :: reintenta, no aborta

set "MEM_NAME=memdump.raw"
set /p MEM_NAME="Nombre del archivo (por defecto memdump.raw): "

set "DEST=!DEST!\Windows\Volcado"
if not exist "!DEST!" mkdir "!DEST!" 2>nul
```

Tres detalles que se repetirán en *todos* los prompts de ruta del framework:

- **La sustitución** `!DEST:"=!` elimina las comillas dobles. Windows Explorer las añade al usar «Copiar como ruta de acceso», y una ruta con comillas internas rompería el `if exist` y la construcción de rutas de destino.

- **El valor por defecto se preasigna antes del `set /p`**. Si el operador pulsa Enter, `set /p` conserva el valor previo de la variable: así Enter siempre produce una entrada válida.
- **La validación devuelve al prompt**, no al menú. Un error de tecleo no cuesta reintroducir toda la configuración.

3.3 Las tres herramientas

Comae DumpIt

```
set "_dumpit_arch=x64"
if /i "%PROCESSOR_ARCHITECTURE%"=="ARM64" set "_dumpit_arch=ARM64"
if not defined _IS_X64 if /i not "%PROCESSOR_ARCHITECTURE%"=="ARM64" set "_dumpit_arch=x86"

"!BASE_PATH!\Volcados\RAM\comae\!_dumpit_arch!\DumpIt.exe" /Q /O "!DEST!\!MEM_NAME!" /TYPE RAW
```

FLAG	SIGNIFICADO	POR QUÉ SE USA
/Q	Modo silencioso	DumpIt pregunta interactivamente por defecto; el prompt bloquearía la ejecución dentro del wrapper.
/O <ruta>	Archivo de salida	Fuerza la ubicación en la carpeta de evidencia en vez del directorio actual.
/TYPE RAW	Formato crudo	Sin él DumpIt genera un `.dmp` de Microsoft con cabecera propietaria. El formato RAW es memoria física plana, lo que aceptan directamente Volatility y Rekall.

Limpieza posterior. DumpIt impone la extensión `.bin` y genera un sidecar `.json` con metadatos:

```
for %%F in ("!MEM_NAME!") do set "BASE_NAME=%%~nF"
if exist "!DEST!\!BASE_NAME!.bin"  move /Y "!DEST!\!BASE_NAME!.bin" "!DEST!\!MEM_NAME!" >nul
2>nul
if exist "!DEST!\!BASE_NAME!.json" del /f /q "!DEST!\!BASE_NAME!.json" >nul 2>nul
```

El renombrado mantiene la coherencia con el nombre que el operador pidió; el `.json` se elimina porque sus datos ya quedan en el diario de la evidencia y su presencia confundiría el inventario de archivos.

POR QUÉ IMPORTA

Los tres binarios están separados porque DumpIt carga un driver de kernel, y un driver debe coincidir con la arquitectura del sistema. La detección considera `ARM64` explícitamente (Surface Pro X, Windows on ARM) y usa `_IS_X64`, que ya tuvo en cuenta WOW64 mediante `PROCESSOR_ARCHITEW6432`.

WinPmem x64 / x86

```
"!BASE_PATH!\Volcados\RAM\winpmem\winpmem_mini_x64_rc2.exe" "!DEST!\!MEM_NAME!"
"!BASE_PATH!\Volcados\RAM\winpmem\winpmem_mini_x86.exe"     "!DEST!\!MEM_NAME!"
```

WinPmem es el adquiridor de memoria del proyecto Rekall/Velocidex: un driver firmado que expone la memoria física y la vuelca secuencialmente. La sintaxis mínima es un único argumento, el archivo de salida. Se ofrece como alternativa a DumpIt porque son implementaciones independientes: si una falla en un sistema concreto, por incompatibilidad de driver, política de firma o antivirus interceptando, la otra suele funcionar. En un contexto forense tener dos vías es más valioso que tener la mejor.

3.4 Los dos controles de calidad

Control 1. Aborto del operador

```
if "!RAM_EL!"=="2" (
    echo [ABORT] Volcado abortado por el usuario. Volviendo al Menu Principal.
    call :LOG_EVENT "Volcado de RAM abortado por el usuario (CTRL+C)."
    pause
    goto MENU_PRINCIPAL
)
```

El código 2 del wrapper significa «el operador confirmó abortar». Se distingue del error real porque no hay nada que arreglar: la decisión fue humana y deliberada, y lo que corresponde es registrarla y devolver el control.

Control 2. Volcado de 0 bytes

```
if not exist "!DEST!\!MEM_NAME!" ( … error … )

set "_ram_filesize=0"
for %%F in ("!DEST!\!MEM_NAME!") do set "_ram_filesize=%%~zF"
if "!_ram_filesize!"=="0" (
    echo [-] El volcado es un archivo vacio: la herramienta creo el archivo pero no escribio
datos.
    echo [i] Causa posible: WinPmem x86 en sistema x64, driver no cargado, o permisos
insuficientes.
    del "!DEST!\!MEM_NAME!" >nul 2>&1
    call :LOG_CENTRAL "ERROR" "ram_dump_empty" "Tool:!RAM_TOOL! … Size:0 - herramienta fallo
silenciosamente"
    pause
    goto MENU_PRINCIPAL
)
```

El control nació de un fallo reproducido, no de una hipótesis.

ADVERTENCIA

WinPmem y DumpIt pueden **crear el archivo de salida, no escribir nada y salir con código 0**. Ocurre cuando el driver no se carga: ejecutar la variante x86 en un sistema x64, falta de privilegios, o bloqueo por política de firma de drivers. Desde el punto de vista del script todo fue bien (código 0, archivo presente) y sin este control se hashearía y documentaría un archivo vacío como si fuera el volcado de la RAM. El modificador `%%~zF` devuelve el tamaño en bytes; si es 0, el archivo se **borra**, para que no quede una evidencia falsa en el caso, y se registra el fallo con la herramienta usada y la causa probable.

Superados ambos controles, se invoca la verificación de integridad opcional:

```
call :HASH_EVIDENCE FILE "!DEST!\!MEM_NAME!" "" "!DEST!\!MEM_NAME!_hashes.txt" "VolcadoRAM"
```

Y el flujo vuelve al menú de volcado, no al principal: es habitual capturar la memoria con dos herramientas distintas para comparar resultados.

CAPÍTULO 4

Módulo 2. Triaje con KAPE

Recolección dirigida de artefactos sin imagen completa del disco. Incluye la generación en caliente de un menú PowerShell para navegar los 267 targets disponibles.

4.1 Qué es un triaje y cuándo sustituye a una imagen

Una imagen forense de un disco de 1 TB tarda horas y ocupa 1 TB. Un *triaje* extrae solo los artefactos con valor investigativo, normalmente entre cientos de megabytes y unos pocos gigabytes, en minutos. KAPE (Kroll Artifact Parser and Extractor) hace exactamente eso: lee una definición declarativa (`.tkape`) que enumera rutas y máscaras de archivo, y las copia preservando estructura, incluso si están bloqueadas por el sistema, porque accede al volumen a bajo nivel.

Está indicado cuando hay urgencia, cuando el disco es demasiado grande para obtener su imagen en la ventana de tiempo disponible, o cuando el alcance de la investigación está bien definido de antemano. No sustituye a una imagen cuando se necesita espacio no asignado, archivos borrados o análisis del sistema de archivos completo.

4.2 Selección de target

El menú ofrece tres caminos:

OPCIÓN	TARGET	CONTENIDO
1	`KapeTriage`	Target compuesto amplio: registro, logs de eventos, prefetch, LNK, jump lists, historial de navegadores, WMI, tareas programadas, USN journal, MFT…
2	`!SANS_Triage`	Conjunto alineado con la metodología SANS (el `!` indica target compuesto en la convención de KAPE).
3	Cualquiera de los 267	Catálogo completo leído del repositorio, para adquisiciones muy específicas.

Escapado del target compuesto

```
set TARGET=^^!SANS_Triage
```

> NOTA
>
> Con `enabledelayedexpansion` activo, el carácter `!` es especial: marca el inicio de una expansión retardada. Un `!` literal dentro de un `set` desaparecería o rompería el valor. El escape `^^!` sobrevive a las dos pasadas de interpretación (la del parser de cmd y la de la expansión retardada) y deja el valor correcto `!SANS_Triage`. Nótese también que este `set` no usa comillas alrededor del par nombre=valor, otra concesión necesaria para que el escape funcione.

Menú de 267 targets generado en caliente

```
:: El .bat ESCRIBE este .ps1 en %TEMP% con un bloque de echo redirigido
$targets = (Get-ChildItem -Path '<kape>\Targets' -Recurse -Filter *.tkape).BaseName
$i = 1
$targets | ForEach-Object {
    Write-Host ("{0,3}. {1,-30}" -f $i, $_) -NoNewline
    if ($i % 3 -eq 0) { Write-Host "" }        :: salto cada 3 -> tres columnas
    $i++
}
$sel = Read-Host "Escribe el NUMERO del Target que deseas utilizar (0 para volver)"
if ($sel -match '^\d+$' -and $sel -gt 0 -and $sel -le $targets.Count) {
    [IO.File]::WriteAllText("$env:TEMP\kape_target.txt", $targets[$sel-1])
}
```

El mecanismo completo. El `.bat` genera el `.ps1`, lo ejecuta con `-ExecutionPolicy Bypass`, y recoge la elección leyendo el archivo de resultado con `set /p TARGET=<archivo`. Después borra ambos temporales. Si el archivo no existe, porque el operador escribió 0 o algo inválido, `TARGET` queda vacío y se vuelve al menú de artefactos.

> POR QUÉ IMPORTA
>
> Formatear 267 entradas en tres columnas alineadas y validar el rango es trivial en PowerShell (`-f` con especificadores de ancho, `-match` con regex) y bastante penoso en lotes. El resultado se transporta por archivo porque un proceso hijo no puede modificar las variables de entorno de su padre.

4.3 Ejecución

kape.exe

```
"!KAPE!" --tsource "!SRC!" --target !TARGET! --tdest "!DEST!" !K_FMT! "!EVID_NAME!" --quiet
```

FLAG	FUNCIÓN
--tsource	Volumen de origen (por defecto `C:`, validado con `:VALIDATE_SRC_DRIVE`).
--target	Nombre del target. **Sin comillas**: KAPE acepta listas separadas por coma y las comillas romperían el parseo.
--tdest	Carpeta de destino de los artefactos.
--vhdx / --zip	Empaqueta el resultado en un disco virtual o un ZIP. El valor que sigue es el nombre base del contenedor.
--quiet	Reduce la verbosidad; el detalle queda en el ConsoleLog propio de KAPE.

El nombre por defecto del contenedor es `<COMPUTERNAME>_<yyyyMMdd_HHmmss>`, generado con PowerShell y con reserva a manipulación de `%DATE%` si no está disponible. La carpeta de destino sigue el mismo patrón, de modo que dos triajes del mismo equipo nunca se solapan.

> **POR QUÉ IMPORTA**
>
> Un contenedor único simplifica el traslado y el hashing: un solo archivo, un solo hash, en lugar de miles de archivos. El VHDX tiene además la ventaja de poder **montarse como unidad** en la máquina de análisis y recorrerse con las herramientas habituales sin extraer nada.

Localización dinámica del contenedor generado

```
set "_kape_hash_file="
for %%F in ("!DEST!\*.zip")  do set "_kape_hash_file=%%~fF"
if not defined _kape_hash_file (
    for %%F in ("!DEST!\*.vhdx") do set "_kape_hash_file=%%~fF"
)
if not defined _kape_hash_file (
    echo [i] No se encontro contenedor ZIP ni VHDX en !DEST! para calcular hash.
) else (
    call :HASH_EVIDENCE FILE "!_kape_hash_file!" "*" "!DEST!\hashes_kape.txt" "KAPE_Triaje"
)
```

NOTA

El nombre no se construye, se busca. KAPE **antepone su propio timestamp** al nombre que se le pasa, así que el nombre final no es predecible desde el `.bat`. Y hay un segundo motivo para buscar el `.zip` primero: si el operador eligió `--vhdx` y el resultado supera cierto tamaño, KAPE lo comprime automáticamente a `.zip` por su cuenta. Buscar ambas extensiones, en ese orden, cubre los dos casos. Si no aparece ninguna, se avisa en lugar de fallar silenciosamente.

CAPÍTULO 5

Módulo 3. Discos físicos y particiones lógicas

El módulo de imaging clásico: cuatro herramientas, dos tipos de origen, traducción de rutas entre tres convenciones distintas, validación de espacio y permisos, y verificación criptográfica de copia byte-fiel.

5.1 Disco físico frente a partición lógica

La primera decisión del módulo no es qué herramienta usar, sino qué se considera el origen. De esa elección depende todo lo demás.

	DISCO FÍSICO	PARTICIÓN LÓGICA
Ruta Windows	`\\.\PHYSICALDRIVE<n>`	`\\.\E:`
Qué incluye	MBR/GPT, tabla de particiones, **todas** las particiones y el espacio no asignado entre ellas	Solo ese volumen, desde su sector de arranque
Cuándo elegirlo	Adquisición completa de un dispositivo incautado	Cuando solo un volumen es relevante, o el disco es demasiado grande
Herramientas ofrecidas	dc3dd, dd, ewfacquire, FTK Imager	dd, ewfacquire

POR QUÉ IMPORTA

El menú se adapta al origen porque no todas las herramientas sirven para ambos casos. dc3dd es un binario Cygwin que solo acepta dispositivos `/dev/sdX`, y FTK Imager CLI no soporta volúmenes lógicos. Ofrecerlos para una partición produciría un fallo confuso. El script decide qué menú mostrar según la variable `DRV_LETTER`: si está definida, el origen es un volumen. Al entrar en cada rama **limpia la variable de la otra** (`set "real_idx="` al entrar en lógica, `set "DRV_LETTER="` al entrar en física), porque un estado residual de una adquisición anterior en la misma sesión mostraría el menú equivocado.

5.2 Detección del origen

Enumeración de discos físicos con reserva

```
call :NEWGUID _dl_guid
set "_dl_raw=!TEMP!orensic_disks_!_dl_guid!.tmp"
set "_dl_ansi=!TEMP!orensic_disks_!_dl_guid!.txt"
wmic diskdrive get Index,Model /value > "!_dl_raw!" 2>nul
if not exist "!_dl_raw!" (
    powershell -NoProfile -Command "Get-PhysicalDisk | ForEach-Object { 'Index=' + $_.DeviceId +
\"`nModel=\" + $_.FriendlyName + \"`n\" }" > "!_dl_raw!" 2>nul
)
type "!_dl_raw!" > "!_dl_ansi!" 2>nul

set "_cur_idx="
for /f "usebackq tokens=1,* delims==" %%K in ("!_dl_ansi!") do (
    if /i "%%K"=="Index" for /f %%b in ("%%L") do set "_cur_idx=%%b"
    if /i "%%K"=="Model" if defined _cur_idx (
        set /a count+=1
        set "DISK_!count!=!_cur_idx!"
        set "MODEL_!count!="
        for /f "tokens=*" %%m in ("%%L") do set "MODEL_!count!=%%m"
        set "_cur_idx="
    )
)
del "!_dl_raw!" "!_dl_ansi!" >nul 2>&1
```

Se usa `/value` para obtener salida en formato `clave=valor`, mucho más fácil de parsear que la tabla por defecto. Los dos temporales no son redundancia: `wmic` escribe en UTF-16 y `for /f` no lo lee bien, así que `type` hace de conversor a ANSI antes del parseo. Los nombres llevan un GUID de `:NEWGUID` porque dos instancias del framework a la vez con un nombre fijo se pisarían la lista de discos, la misma clase de fallo ya corregida en el write blocker.

> ADVERTENCIA
>
> `Index` y `Model` se leen **en pareja**, en un solo recorrido. Recorrerlos por separado y emparejarlos por posición desalinea la lista cuando un disco no informa `Model`, algo que ocurre con algunos USB y con discos virtuales. El número mostrado en pantalla apuntaría entonces a otro disco, y se adquiriría el equivocado. El bucle interior sin delimitadores recorta además los espacios que añade WMIC, para que el índice no quede como `" 2"`.

> POR QUÉ IMPORTA
>
> La reserva hacia PowerShell no es cosmética. `wmic.exe` está marcado como obsoleto y ya no viene incluido en las builds recientes de Windows 11. La ruta alternativa con `Get-PhysicalDisk` emite deliberadamente el **mismo formato** `Index=…` / `Model=…`, de modo que el código de parseo posterior es idéntico para ambas fuentes.

El framework **conserva el índice real de WMI** en `DISK_<n>` y lo recupera tras la selección: la posición en la lista y el índice del disco no tienen por qué coincidir, porque el tercer disco listado puede ser el `PHYSICALDRIVE4`. Confundirlos significaría adquirir el disco equivocado.

Traducción a la convención Cygwin de dc3dd

```
set "DRV_PATH=\\.\PHYSICALDRIVE!real_idx!"
set "letters=abcdefghijklmnopqrstuvwxyz"
for /f %%i in ("!real_idx!") do set "cyg_letter=!letters:~%%i,1!"
if "!cyg_letter!"=="" (
    echo [-] Indice de disco !real_idx! fuera del rango soportado por dc3dd (0-25).
    call :LOG_CENTRAL "ERROR" "dc3dd_index_overflow" …
)
set "CYG_DEV=/dev/sd!cyg_letter!"

:: Ruta de destino en formato Cygwin
set "dlw=!DEST:~0,1!"
set "cyg_path=/cygdrive/!dlw!!DEST:~2!\!IMG_NAME!"
set "cyg_path=!cyg_path:\=/!"
```

La extracción de la letra usa la sintaxis de subcadena `!letters:~<offset>,1!`: el índice del disco es directamente el desplazamiento dentro del alfabeto, porque Cygwin mapea `PHYSICALDRIVE0` a `/dev/sda`, el índice `1` a `/dev/sdb`, y así sucesivamente. La ruta de destino se transforma en dos pasos: se sustituye `D:` por `/cygdrive/d` y luego todos los backslashes por barras normales.

Para volúmenes lógicos la traducción es distinta:

```
set "DRV_PATH=\\.\!drive!"
set "CYG_DEV=//./!drive!"      :: barras normales, sin escape
```

> ADVERTENCIA
>
> Problema real documentado en el código: pasar `\\.\E:` a través de un `echo` hacia el archivo de comando **pierde un backslash** y llega como `\.\E:`, una ruta inválida. Cygwin acepta la forma equivalente con barras normales `//./E:`, que atraviesa el `echo` sin alteración. El control de rango de 0-25 protege el otro extremo: en un servidor con más de 26 discos, la sintaxis de subcadena devolvería vacío y dc3dd recibiría `/dev/sd`, un dispositivo inexistente. El script lo detecta antes y lo explica.

5.3 Controles previos a escribir

Nombre por defecto informativo y único

```
call :UTCNOW _ADQ_TSN COMPACT
if defined DRV_LETTER (
    set "IMG_DEFAULT=Particion_!DRV_LETTER!_!_ADQ_TSN!"
) else (
    set "IMG_DEFAULT=Disco!real_idx!_!_ADQ_TSN!"
)
```

Produce nombres como `Disco2_20260724_213500Z` o `Particion_E_20260724_213500Z`: identifican el origen y el momento, y garantizan que pulsar Enter **nunca** sobrescriba una adquisición anterior. La `Z` final indica UTC; si no hubo PowerShell y la marca vino de WMIC en hora local, el sufijo es `L`, de modo que el nombre del archivo declara qué zona horaria representa.

Prueba de escritura efectiva

```
set "TEST_FILE=!DEST!\!IMG_NAME!\_test_write.tmp"
echo test >"!TEST_FILE!" 2>nul
if not exist "!TEST_FILE!" (
    echo [-] No hay permisos de escritura en !DEST!\!IMG_NAME!.
    goto ADQ_DESTINO_ASK
)
del "!TEST_FILE!" 2>nul
```

> POR QUÉ IMPORTA
>
> Se comprueba la existencia del archivo y no el `errorlevel`, porque el `errorlevel` de una redirección fallida **no es fiable entre versiones de cmd.exe**. Que el archivo exista realmente es la única prueba concluyente. Es preferible descubrir un problema de permisos ahora, con un archivo de 5 bytes, que dos horas después de empezar a escribir una imagen de 500 GB.

Justo después, la variable `DEST` se reasigna a `<DEST>\<IMG_NAME>`: a partir de ese punto la imagen, su `.log`, los hashes, el archivo de verificación y el `Forensic_Journal.log` quedan **todos juntos** en la carpeta de esa evidencia, y no sueltos mezclándose con otras adquisiciones.

Validación de espacio disponible

```
:: Tamaño del origen (rama WMIC o rama PowerShell según HAS_WMIC)
wmic diskdrive where "Index=!real_idx!" get Size /value | findstr /I "Size"
wmic logicaldisk where "DeviceID='!DRV_LETTER!:'" get Size /value | findstr /I "Size"

:: Espacio libre en el destino
wmic logicaldisk where "DeviceID='!DEST_DRIVE!:'" get FreeSpace /value | findstr /I "FreeSpace"

:: Comparación delegada a PowerShell
powershell -Command "if ([long]!SRC_SIZE! -gt [long]!FREE_SPACE!) { exit 1 } else { exit 0 }"
```

> POR QUÉ IMPORTA
>
> Comparar dos números se delega a PowerShell por una limitación del lenguaje de lotes: la aritmética de `set /a` en cmd.exe usa enteros **de 32 bits con signo**, con un máximo de 2.147.483.647, algo más de 2 GB. Cualquier tamaño de disco moderno desborda y produce comparaciones erróneas. `[long]` en PowerShell es de 64 bits y maneja petabytes sin problema.

Si el destino no alcanza, se advierte específicamente que «si usas formato crudo [.dd] la adquisición fallará por falta de espacio», matiz relevante porque una imagen E01 comprimida sí podría caber, y se pide confirmación. Tanto la cancelación como la decisión de continuar quedan registradas en el diario.

5.4 Las cuatro herramientas de imaging en detalle

dc3dd, la opción recomendada

```
"!DC3DD_EXE!" "if=!CYG_DEV!" "of=!cyg_path!.dd" "hash=sha256" "log=!cyg_path!.log"
```

PARÁMETRO	FUNCIÓN
if=	**Input file**: el dispositivo de origen en notación Cygwin.
of=	**Output file**: la imagen cruda de destino.
hash=sha256	Calcula el hash **del stream leído del dispositivo**, sobre la marcha, y lo escribe en el log.
log=	Archivo de log con progreso, errores de lectura por sector y el hash resultante.

> **POR QUÉ IMPORTA**
>
> dc3dd es la variante de `dd` desarrollada por el DoD Cyber Crime Center específicamente para uso forense. Su ventaja decisiva aquí es que hashea el **origen** durante la lectura: eso permite comparar después contra el hash de la imagen y demostrar que la copia es byte a byte idéntica al dispositivo, y no solo que la imagen es internamente consistente. Registra también los errores de lectura por sector en lugar de abortar, algo esencial en discos con sectores defectuosos.

dd

```
"!DD_EXE!" "if=!DRV_PATH!" "of=!DEST!\!IMG_NAME!.dd" bs=512k --progress
```

PARÁMETRO	FUNCIÓN
bs=512k	Tamaño de bloque de 512 KB. Bloques pequeños (512 B) serían lentísimos por la sobrecarga de llamadas al sistema; muy grandes desperdiciarían trabajo al reintentar un bloque con un sector ilegible.
--progress	Muestra bytes copiados en tiempo real. Sin él la consola quedaría muda durante horas y el operador no sabría si avanza.

Es la **única** herramienta de imaging crudo disponible tanto para discos físicos como para volúmenes lógicos, porque acepta la ruta nativa de Windows `\\.\` en ambas formas. No tiene hashing integrado: la integridad se

verifica después con `HASH_EVIDENCE`.

ewfacquire, formatos EnCase y SMART

```
"!EWF_EXE!" -u -c none !SEG_OPT! !FMT_OPT! -t "!DEST!\!IMG_NAME!" "!DRV_PATH!"
```

FLAG	FUNCIÓN	ORIGEN DEL VALOR
-u	Modo no interactivo (**unattended**)	Fijo: sin él la herramienta preguntaría por cada parámetro y bloquearía el wrapper.
-c none	Sin compresión	Fijo: prioriza velocidad y fidelidad; comprimir alarga la adquisición.
-S 80000G	Tamaño máximo de segmento	Opción 2 del menú de segmentación: un valor absurdamente alto fuerza una **imagen única**. Sin el flag, ewfacquire segmenta en 1,4 GB.
-f encase7 / smart	Formato del contenedor	Menú de formato. Sin flag: EnCase v6 (`.E01`). Con `encase7`: `.Ex01`. Con `smart`: `.S01`.
-t	Objetivo (**target**) sin extensión	La herramienta añade la extensión según el formato elegido.

NOTA

La segmentación por defecto en 1,4 GB es una convención heredada de la era en que las imágenes se archivaban en CD y en sistemas de archivos con límite de 2 GB por archivo. Sigue siendo útil para traslado en medios con restricciones o para copiar por red en trozos manejables. La opción de imagen única es preferible cuando el destino es NTFS y la imagen se va a montar directamente.

FTK Imager CLI

```
:: Modos comprimidos (E01 / S01)
"!FTK_EXE!" "!DRV_PATH!" "!DEST!\!IMG_NAME!!ext_out!" !fmt_flag! ^
    --case-number "!fcase!" --evidence-number "!fevid!" --examiner "!fexam!" !frag_flag! --
compress 6

:: Modo Raw: sin metadatos ni compresión
"!FTK_EXE!" "!DRV_PATH!" "!DEST!\!IMG_NAME!.dd" !frag_flag!
```

FLAG	FUNCIÓN
--e01 / --s01	Formato de salida. Ausente = RAW.
--case-number	Número de caso, embebido en la cabecera del contenedor E01.
--evidence-number	Identificador de la pieza de evidencia.
--examiner	Nombre del examinador; por defecto `%USERNAME%`.
--frag 2000M	Segmentos de 2000 MB, por debajo del límite de 4 GB de FAT32.
--compress 6	Nivel de compresión intermedio (rango 0-9): buen equilibrio entre tamaño y tiempo de CPU.

> POR QUÉ IMPORTA
>
> El modo Raw omite los metadatos porque el formato RAW es un volcado plano de sectores: **no tiene estructura donde almacenarlos**. Pasarle `--case-number` haría que la herramienta los ignorara o fallara. Por eso el script bifurca a `:FTK_CMD_RAW` y construye una línea de comando distinta, sin esos flags ni `--compress`.

Los metadatos de caso son la aportación diferencial de este formato: quedan dentro del contenedor E01, de modo que la evidencia lleva su propia identificación aunque el archivo se renombre o se separe de la documentación del caso. Es el elemento de cadena de custodia más difícil de perder.

5.5 Verificación de copia byte-fiel

VERIFY_DC3DD

```
for /f "usebackq tokens=1" %%H in (`findstr /c:"(sha256)" "!_vd_log!"`) do ^
    if not defined _vd_src set "_vd_src=%%H"

for /f "usebackq skip=1 tokens=*" %%I in (`certutil -hashfile "!_vd_img!" SHA256`) do ^
    if not defined _vd_dst set "_vd_dst=%%I"
set "_vd_dst=!_vd_dst: =!"          :: certutil inserta espacios en el hash

if /i "!_vd_src!"=="!_vd_dst!" goto VD_MATCH
```

Del log de dc3dd se extrae el primer token de la línea que contiene `(sha256)`, que es donde está el hash del dispositivo. De `certutil` se toma la segunda línea (`skip=1` omite la cabecera «Hash SHA256 de archivo…») y se le eliminan los espacios que `certutil` intercala. La comparación es insensible a mayúsculas con `/i`, porque las dos herramientas usan convenciones distintas.

El veredicto se documenta en un archivo, en los dos casos:

```
:: Coincidencia
>"!DEST!\!IMG_NAME!_VERIFICACION.txt" echo RESULTADO: EXITOSO - copia byte-fiel verificada
>>… echo SHA256 (origen y imagen identicos): !_vd_src!

:: Discrepancia
>"!DEST!\!IMG_NAME!_VERIFICACION.txt" echo RESULTADO: FALLIDO - los hashes NO coinciden
>>… echo Origen (dc3dd) SHA256: !_vd_src!
>>… echo Imagen (.dd)   SHA256: !_vd_dst!
```

El archivo de verificación se escribe tanto si los hashes coinciden como si difieren, con los dos valores a la vista en el segundo caso.

POR QUÉ IMPORTA

Hashear y verificar no son lo mismo. Calcular el hash de una imagen solo demuestra que esa imagen no ha cambiado desde el cálculo. Comparar el hash del **origen** contra el de la **imagen** demuestra algo mucho más fuerte: que la imagen es una réplica exacta del dispositivo. Esta verificación es la que sostiene la afirmación «copia byte-fiel» ante un tribunal, y solo es posible porque dc3dd hasheó el stream durante la lectura. Con las otras tres herramientas habría que releer el dispositivo completo por separado.

Cinco casos de fallo se manejan con etiquetas propias y mensaje explícito: falta el log, falta la imagen, no se pudo extraer el hash de origen, no se pudo calcular el de la imagen, o los hashes difieren. Ninguno interrumpe el framework.

CAPÍTULO 6

Módulo 3.3. Adquisición lógica de archivos y carpetas

Copia forense selectiva con preservación total de metadatos NTFS, mediante un navegador interactivo del sistema de archivos y captura previa de metadatos de custodia.

6.1 El navegador de origen

El punto de entrada acepta dos formas: una letra de unidad o una ruta completa. La discriminación es sintáctica:

```
if "!log_drv:~2,1!"=="\" goto ADQ_LOG_FULLPATH      :: C:\algo -> 3er carácter es \
set "log_drv=!log_drv::=!"                           :: normaliza "C:" -> "C"
set "log_drv=!log_drv:\=!"                           :: normaliza "C\" -> "C"
if not exist "!log_drv!:\" ( … error … )
```

Si es una ruta completa se distingue además entre carpeta y archivo, porque el flujo posterior es distinto: `if exist "!log_drv!\"`, con backslash final, es verdadero solo para directorios.

Listado del contenido del directorio actual

```
for /D %%D in ("!LOG_SRC!*") do ( echo     [DIR]  %%~nxD & set "_nav_has_dirs=1" )
for %%F in ("!LOG_SRC!*")    do ( echo     [FILE] %%~nxF & set "_nav_has_files=1" )
```

`for /D` itera solo directorios; el `for` normal, solo archivos. Las banderas `_nav_has_dirs` y `_nav_has_files` permiten mostrar «(sin subcarpetas)» o «(sin archivos)» en lugar de dejar una sección vacía y ambigua. El modificador `%%~nx` imprime solo nombre y extensión, no la ruta completa, para que el listado sea legible.

Comandos del navegador:

COMANDO	EFECTO
`S`	Selecciona la carpeta actual completa, incluyendo subcarpetas.
`..`	Sube un nivel. Desde la raíz de la unidad, detectado con `if "!LOG_SRC:~3,1!"==""`, vuelve a la lista de unidades. El ascenso usa `for %%P in ("!LOG_SRC!..") do set "LOG_SRC=%%~fP\"`, que resuelve el `..` a ruta absoluta.
`<nombre>`	Si es carpeta, entra en ella; si es archivo, lo selecciona como origen único.
`0`	Cancela y vuelve al menú de adquisición.

La confirmación posterior muestra el origen y el modo detectado («Carpeta completa - incluye subcarpetas» o «Archivo individual») y admite `S` para confirmar, `N` para volver al navegador y `0` para cancelar. Es el punto donde el operador verifica que va a copiar lo que cree.

6.2 Metadatos de custodia

```
set "fcase=SIN_CASO"  & set /p fcase="Caso # [SIN_CASO]: "
set "fevid=SIN_ID"    & set /p fevid="Evidencia # [SIN_ID]: "
set "fexam=%USERNAME%" & set /p fexam="Examinador [%USERNAME%]: "
set "fcase=!fcase:"=!"   :: limpieza de comillas en los tres
```

El idioma `set "var=defecto" & set /p var="prompt"` aprovecha que `set /p` **conserva el valor previo** si el usuario pulsa Enter. Los tres valores se escriben en el diario de la evidencia junto con origen, destino y resultado, formando el registro mínimo de cadena de custodia para esa pieza.

6.3 La copia con robocopy

Robocopy en modo carpeta y en modo archivo

```
:: Carpeta completa
robocopy "!LOG_SRC_Q!" "!LOG_COPY_DEST!" /E /COPYALL /NP /TEE /LOG:"!RBC_LOG!" ^
    & if errorlevel 8 (exit 1) else (exit 0)

:: Archivo individual: se separa carpeta y nombre
robocopy "!_rbc_dir!" "!LOG_COPY_DEST!" "!_rbc_file!" /COPYALL /NP /TEE /LOG:"!RBC_LOG!" ^
    & if errorlevel 8 (exit 1) else (exit 0)
```

FLAG	FUNCIÓN	RELEVANCIA FORENSE
/E	Copia subdirectorios, incluidos los vacíos	Un directorio vacío es información: puede indicar borrado deliberado.
/COPYALL	Equivale a `/COPY:DATSOU`	Datos, Atributos, Timestamps, ACLs (**Security**), información de propietario (**Owner**) y auditoría (**aUditing**). Es el conjunto completo de metadatos NTFS: quién era el dueño, quién tenía acceso, cuándo se creó, modificó y accedió cada archivo.
/NP	Sin porcentaje de progreso	El porcentaje se reescribe constantemente e inundaría el log de miles de líneas inútiles.
/TEE	Salida simultánea a consola y log	El operador ve el avance y el log queda completo, sin tener que elegir entre las dos cosas.
/LOG:	Archivo de log de la copia	Queda como `<nombre>_robocopy.log`: registro archivo por archivo de lo copiado y lo omitido.

> **POR QUÉ IMPORTA**
>
> Robocopy usa un código de salida con **bits de estado**, no un simple 0/1: 0 = nada que copiar, 1 = archivos copiados, 2 = extras en destino, 4 = desajustes, 8 = fallos de copia, 16 = error fatal. Los valores de 0 a 7 son todos variantes de éxito. Sin traducción, el wrapper interpretaría un exitoso «1 = archivos copiados» como error. La cláusula `& if errorlevel 8 (exit 1) else (exit 0)` se ejecuta dentro del proceso lanzado y normaliza el resultado al contrato del framework. Este mismo patrón aparece en **todos** los puntos del script donde robocopy corre bajo el wrapper, sin excepción.

En modo archivo individual, robocopy exige separar carpeta y nombre, porque no acepta una ruta de archivo completa como origen, así que el script los descompone con `%%~dpF` y `%%~nxF` y le quita el backslash final al directorio.

La verificación final hashea toda la carpeta copiada con patrón `*`, produciendo un reporte con una línea por archivo:

```
call :HASH_EVIDENCE FOLDER "!LOG_COPY_DEST!" "*" "!HMF_REPORT!" "Adquisicion_Logica"
```

CAPÍTULO 7

Módulo 4. VMware

Cuatro flujos: volcado de memoria, consolidación de disco con política de snapshots, reconstrucción de un snapshot concreto como disco completo y clon íntegro de la máquina. Es el módulo con más lógica defensiva del framework, porque VMware no expone ninguna señal directa de «este archivo es el volcado que acabas de generar».

7.1 Conceptos previos: los archivos de una VM de VMware

Nada de lo que viene después se sostiene sin saber qué guarda cada archivo de la carpeta de una VM, y sobre todo cuáles sirven como origen válido de adquisición y cuáles no.

ARCHIVO	CONTENIDO	RELEVANCIA FORENSE
`.vmx`	Configuración: hardware virtual, discos adjuntos, red	Sin él VMware no puede abrir la VM. Texto plano, legible.
`.vmdk` (descriptor)	Texto plano con `CID`, `parentCID`, `parentFileNameHint` y la lista de extents	Define la topología de la cadena de discos.
`-flat.vmdk`, `-sNN.vmdk`	Extents: los datos binarios reales del disco	Nunca son origen válido para vdiskmanager.
`-delta.vmdk`, `-NNNNNN.vmdk`	Discos diferenciales de snapshots	Contienen solo los bloques modificados desde el padre.
`.vmsd`	Diccionario de snapshots: nombres visibles, discos y archivos de estado	Única fuente que vincula un `.vmdk` de snapshot con su memoria.
`.vmsn`	Estado del snapshot (dispositivos y, si aplica, referencia a memoria)	Siempre pertenece a un snapshot con nombre.
`.vmem`	**Volcado de la RAM**, del mismo tamaño que la memoria asignada	El objetivo del volcado de memoria. Analizable con Volatility.
`.vmss`	Estado de suspensión: dispositivos, sin la RAM	Complementa al `.vmem`; solo, no contiene memoria.
`.nvram`	Estado de la BIOS/UEFI virtual	Necesario para reabrir la VM en el mismo estado de firmware.
`.vmdk.lck`	Directorio de bloqueo creado mientras el disco está en uso	Señal fiable de VM encendida **o suspendida**.

7.2 Selección de la VM: SELECT_VMWARE_VM

```
if not exist "%APPDATA%\VMware\inventory.vmls" ( … error … )

for /f "tokens=2 delims==" %%A in ('findstr /I ".vmx" "%APPDATA%\VMware\inventory.vmls"') do (
    for %%F in (%%A) do (
        set "vp=%%~dpF"
        if "!vp:~-1!"=="\" set "vp=!vp:~0,-1!"
        set "is_dup=0"
        for /L %%k in (1,1,!vm_count!) do if /I "!VM_PATH_%%k!"=="!vp!" set "is_dup=1"
        if "!is_dup!"=="0" (
            set /a vm_count+=1
            set "VM_PATH_!vm_count!=!vp!"   & set "VM_NAME_!vm_count!=%%~nF"
            set "VM_VMX_!vm_count!=%%~fF"
        )
    )
)
```

SELECT_VMWARE_VM: inventario, deduplicación por carpeta y estado real de cada máquina.

El inventario de VMware Workstation es un archivo de texto con líneas `clave = "valor"`. Se filtran las que contienen `.vmx` y se extrae la ruta. La deduplicación por carpeta hace falta porque el inventario puede tener varias entradas apuntando a la misma máquina: favoritos, entradas huérfanas o referencias duplicadas que quedan atrás después de mover la VM.

Detección del estado, en dos pasos:

```
if exist "!VMRUN_EXE!" for /f "skip=1 delims=" %%L in ('"!VMRUN_EXE!" list 2^>nul') do echo
%%L>>"!_svm_running!"

findstr /I /X /C:"!VM_VMX_%%i!" "!_svm_running!" >nul 2>&1
if !errorlevel! equ 0 ( set "_svm_state=Encendida" ) else (
    for %%L in ("!VM_PATH_%%i!\*.vmdk.lck") do set "_svm_haslock=1"
    if defined _svm_haslock set "_svm_state=Suspendida"
)
```

ADVERTENCIA

Descarte documentado en el código. La línea `checkpoint.vmState` del `.vmx` **no sirve** para detectar suspensión: se verificó en pruebas reales que VMware la deja como referencia obsoleta incluso después de apagar la VM normalmente, sin limpiarla. La única señal que refleja el estado *actual* es la existencia del bloqueo `.vmdk.lck`. A esto se suma que `vmrun list` solo reporta VMs **encendidas**: una suspendida no aparece ahí pero sí mantiene el bloqueo, de modo que ambas comprobaciones son complementarias y las dos son necesarias.

El listado de `vmrun list` se acumula en un temporal con nombre GUID y se compara con `findstr /X`, es decir coincidencia de línea completa, para que una ruta no case parcialmente con otra más larga. El temporal se borra en cuanto se ha usado.

7.3 Volcado de memoria (opción 1)

El objetivo es obtener un `.vmem` que corresponda al estado *actual* de la VM, sin confundirlo con volcados de snapshots antiguos que vivan en la misma carpeta. El flujo tiene cinco etapas.

Etapa 1. Marca de referencia y suspensión

```
for /f "usebackq delims=" %%T in (`powershell -NoProfile -Command ^
    "(Get-Date).ToUniversalTime().ToString('o')"`) do set "_vmwm_cutoff=%%T"

call :LOG_CENTRAL "PRE_ACTION" "vmrun_suspend_inicio" "VM: !vm_vmx!" ^
    "Alteracion minima controlada - DFIR standard NIST SP800-86"
call :UTCNOW _VMR_TS1
"!VMRUN_EXE!" suspend "!vm_vmx!" >nul 2>&1
set "_vmr_el=!errorLevel!"
call :UTCNOW _VMR_TS2
call :LOG_CMD "VMware_Suspend" "vmrun suspend ""!vm_vmx!""" "!_vmr_el!" "!_VMR_TS1!"
"!_VMR_TS2!" "" ""
```

`vmrun suspend` enmarcado entre registro previo y posterior.

El instante de referencia se captura en formato ISO-8601 con precisión de milisegundos (el especificador `'o'`) **antes** de suspender. La acción queda enmarcada entre marcas de tiempo y se registra como `PRE_ACTION` y `POST_ACTION`, con la justificación metodológica escrita de forma explícita en el propio log.

POR QUÉ IMPORTA

Suspender evita el memory smearing. Copiar un `.vmem` de una VM en ejecución produce un volcado *inconsistente*: mientras se copian los primeros gigabytes, el sistema operativo invitado sigue modificando estructuras en los últimos. El resultado son listas de procesos con punteros a memoria ya reasignada, tablas de páginas incoherentes y plugins de Volatility que fallan o devuelven datos falsos. Es lo que se conoce como *memory smearing*. Suspender congela el estado: la RAM se escribe completa y coherente. NIST SP 800-86 admite explícitamente este tipo de **alteración mínima, controlada y documentada** cuando resulta necesaria para obtener evidencia válida, y por eso el script la registra en el log con esa etiqueta en lugar de dejarla pasar como un efecto colateral silencioso.

Si `vmrun` no está disponible, el script no aborta. Imprime una advertencia crítica en rojo pidiendo al operador que suspenda la VM a mano antes de continuar, y espera con un `pause`.

Etapa 2. Identificar el volcado nuevo por fecha

```
$dir = '<carpeta de la VM>'
$cutoff = [datetime]::Parse('<instante de referencia>').ToUniversalTime()
Get-ChildItem -Path (Join-Path $dir '*') -Include *.vmem,*.vmss -File |
    Where-Object { $_.LastWriteTimeUtc -ge $cutoff } |
    Select-Object -ExpandProperty Name
```

Consulta por `LastWriteTimeUtc` contra el instante de referencia capturado en la etapa 1.

ADVERTENCIA

Por qué por fecha y no por nombre. La aproximación intuitiva sería listar los archivos antes y después de suspender y quedarse con la diferencia. No funciona: se verificó en pruebas reales que VMware *reutiliza el mismo nombre* de `.vmss` y `.vmem` entre suspensiones sucesivas de la misma VM, porque el sufijo hash del nombre es estable. Un archivo que *ya existía* y que VMware sobrescribe con contenido fresco nunca aparecería como «nuevo» en un diff por nombre, aunque sea exactamente el volcado que se pidió. Comparar por fecha de modificación funciona igual en los dos casos: archivo nuevo o archivo reutilizado.

Etapa 3. La condición de carrera .vmss / .vmem

```
for %%V in (!VMWM_LIVE_LIST!) do (
    set "_vmwm_nf=%%~V"
    if /I "!_vmwm_nf:~-5!"==".vmss" (
        for %%X in ("!_vmwm_nf!") do set "_vmwm_nfbase=%%~nX"
        echo !VMWM_LIVE_LIST!| findstr /I /C:"!_vmwm_nfbase!.vmem" >nul
        if !errorlevel! neq 0 set "_vmwm_pending_vmem=!_vmwm_nfbase!"
    )
)

set "_vmwm_vmem_found="          :: limpieza CRÍTICA - ver nota
set "_vmwm_vmem_wait=0"
:VMWM_WAIT_VMEM_LOOP
if not defined _vmwm_pending_vmem goto VMWM_WAIT_VMEM_DONE
if exist "!vm_dir!\!_vmwm_pending_vmem!.vmem" ( set "_vmwm_vmem_found=1" & goto
VMWM_WAIT_VMEM_DONE )
timeout /t 2 >nul
set /a _vmwm_vmem_wait+=2
if !_vmwm_vmem_wait! lss 120 goto VMWM_WAIT_VMEM_LOOP
```

Detección del `.vmss` huérfano y espera activa con límite de 120 segundos.

ADVERTENCIA

La condición de carrera. `vmrun suspend` **retorna en cuanto la suspensión empieza**, no cuando VMware ha terminado de volcar toda la RAM a disco. En una VM con 16 GB asignados eso significa que el `.vmss` (estado de dispositivos, pequeño) puede quedar escrito y con fecha actualizada mientras el `.vmem` de 16 GB todavía se está escribiendo. La consulta de la etapa 2 vería el `.vmss` y no a su compañero, y el script copiaría el estado de dispositivos **sin la memoria**. La detección es simple: si hay un `.vmss` modificado cuyo `.vmem` homónimo no está en la lista, esa es la condición de carrera, y no una señal de que el `.vmem` no vaya a existir.

ADVERTENCIA

Segundo fallo real: variable no reinicializada. La línea `set "_vmwm_vmem_found="` antes del bucle es imprescindible. Sin ella, un valor `1` heredado de una invocación *anterior* de este mismo flujo, con el operador volviendo a usar la opción del menú para otra VM en la misma sesión, se leería como «encontrado» sin haberlo verificado en esta corrida, y una espera que en realidad falló se marcaría como exitosa. Es un fallo detectado en auditoría y documentado en el propio código.

Durante la espera se comprueba **existencia simple** del `.vmem`, no su fecha. Ya se sabe con certeza que se está a mitad de suspender la VM en esta sesión, porque su `.vmss` compañero se acaba de detectar recién modificado, y en ese contexto la aparición del archivo es señal suficiente. El código anota además que `LastWriteTimeUtc` resultó no reflejar con fiabilidad el instante real de escritura de VMware: el `.vmem` existía de forma verificable pero la comparación por fecha no lo detectaba.

Si a los 120 segundos no aparece, el script no miente: avisa de que solo se copiará el `.vmss` **sin la RAM**, apunta las causas posibles (memoria no reservada, VMware Tools, host bajo carga) y pide verificación manual.

Etapa 4. Plan B: aislamiento por descarte

```
for /f "usebackq tokens=1,* delims==" %%K in ("!_vmwm_vmsd!") do (
    set "_vmwm_keyt=!_vmwm_key: =!"
    echo !_vmwm_keyt!| findstr /R /I /C:"^snapshot[0-9]*\.filename" >nul
    if !errorlevel! equ 0 (
        set "_vmwm_val=%%L"
        …
        for %%X in ("!_vmwm_val!") do >>"!_vmwm_exclude!" echo %%~nX
    )
)

for %%E in (vmss vmem) do (
    for %%F in ("!vm_dir!\*.%%E") do (
        findstr /X /I /C:"%%~nF" "!_vmwm_exclude!" >nul 2>&1
        if !errorlevel! neq 0 set "VMWM_LIVE_LIST=!VMWM_LIVE_LIST! "%%~nxF""
    )
)
```

Exclusión de los archivos de estado que pertenecen a snapshots con nombre.

Cuándo entra en juego. Si la VM ya estaba suspendida antes de entrar al módulo, `vmrun suspend` no modificó nada y la etapa 2 no encontró candidatos. Entonces se razona por eliminación: se leen del `.vmsd` todos los nombres base de archivos de estado que *pertenecen a un snapshot con nombre* y se excluyen. Lo que queda es, necesariamente, el estado independiente «en vivo».

Los `.vmsn` se excluyen siempre de este descarte, porque por definición *siempre* son artefactos de un snapshot con nombre.

NOTA

Peculiaridad de findstr documentada. El patrón regex omite deliberadamente el ancla de fin de línea `$`. Se verificó con reproducción aislada que el ancla `$` de `findstr /R` **falla cuando la entrada llega por pipe desde `echo`**. El prefijo `^snapshot[0-9]*\.filename` ya identifica la clave sin ambigüedad: no existe otra clave en el formato `.vmsd` que comparta ese prefijo.

Si tampoco así se consigue distinguir, el script activa `VMWM_COPY_ALL=1`, avisa de que copiará **todos** los archivos de memoria incluidos los de snapshots anteriores, y pide revisión manual. Nunca finge una certeza que no tiene.

Etapa 5. Copia y reanudación

```
:: Copia de la lista aislada (VMW_COPY_MEM_LIST)
robocopy "!vm_dir!" "!vm_dest!" !VMWM_LIVE_LIST! !RBC_J! /NJH /NJS /NDL /NC /NS ^
    & if errorlevel 8 (exit 1) else (exit 0)

:: Copia por extensión, una llamada por tipo (VMW_COPY_MEM_EXT)
robocopy "!vm_dir!" "!vm_dest!" *.vmem !RBC_J! /NJH /NJS /NDL /NC /NS
```

FLAG	FUNCIÓN
/J	E/S sin buffer. Para un `.vmem` de varios GB evita saturar la caché del sistema con datos que no se van a reutilizar. Solo se añade si la build es ≥ 9200 (variable `RBC_J`).
/NJH /NJS	Sin cabecera ni resumen de trabajo: reduce el ruido cuando se hacen varias llamadas seguidas.
/NDL /NC /NS	Sin listado de directorios, sin clases de archivo, sin tamaños: deja solo lo esencial en pantalla.

Al terminar se hashea la carpeta con el patrón `*.vmem,*.vmss,*.vmsn` y se ofrece reanudar la VM con `vmrun start`, registrando también esa acción con sus marcas de tiempo.

7.4 Consolidación de disco (opción 2)

Clasificación de descriptores por contenido

```
for /f "usebackq tokens=1,* delims==" %%K in (`findstr /B /I ^
        /C:"CID=" /C:"parentCID=" /C:"parentFileNameHint=" %1 2^>nul`) do (
    if /I "!_c_key!"=="CID"                set "_c_cid=!_c_val!"
    if /I "!_c_key!"=="parentCID"          set "_c_parentcid=!_c_val!"
    if /I "!_c_key!"=="parentFileNameHint" set "_c_parenthint=!_c_val!"
)
if defined _c_parenthint set "_c_parenthint=!_c_parenthint:"=!"
if defined _c_parentcid ( if /I not "!_c_parentcid!"=="ffffffff" set "_c_isbase=0" )
endlocal & set "_VMDK_ISBASE=%_c_isbase%" & set "_VMDK_PARENTHINT=%_c_parenthint%" & set
"_VMDK_CID=%_c_cid%"
```

VMWARE_CLASSIFY_VMDK: lee tres campos del descriptor y devuelve la clasificación al llamador.

Los tres campos provienen de la especificación pública *VMware Virtual Disk Format*. `CID` es el identificador de contenido del disco; `parentCID`, el del padre; `parentFileNameHint`, el nombre del archivo padre. Un disco es **base** si no declara `parentCID` o si ese campo vale `ffffffff`, la convención de «sin padre».

POR QUÉ IMPORTA

Por qué por contenido y no por nombre de archivo. La convención habitual es que los snapshots se llamen `<disco>-000001.vmdk`, pero **es solo una convención**: un disco puede renombrarse, y un disco base podría llamarse `algo-000001.vmdk` sin ser snapshot. Clasificar por el `parentCID` real es la única forma correcta. El script sí filtra por nombre los *extents binarios* (`-sNN`, `-flat`, `-delta`) antes de clasificar, porque esos nunca son descriptores válidos y su exclusión por patrón es segura.

El uso de `setlocal` y `endlocal` con la técnica `endlocal & set var=%local%` permite que la subrutina trabaje con variables aisladas y devuelva solo tres resultados al ámbito del llamador. `VMDK_PRINT_CHILDREN` aprovecha lo mismo para dibujar de forma recursiva el árbol de la cadena, con indentación por nivel.

Los cuatro modos de snapshot

MODO	DESCRIPCIÓN	IMPLEMENTACIÓN	CUÁNDO USARLO
1	Disco completo: original + snapshots aplicados	Consolida el descriptor elegido; además copia sin alterar `*-delta.vmdk`, `*-s??.vmdk`, `*-??????.vmdk` y `*.vmsn` a `Snapshots\`	Estado actual de la máquina, que es lo que el usuario veía
2	Solo el disco original	Filtra la lista de descriptores a los clasificados como base	Se quiere el estado previo a cualquier snapshot
3	Solo un snapshot concreto	Redirige a `VM_VMWARE_RECONSTRUCT` con la VM ya seleccionada	Ya se adquirió el original, o solo interesa ese punto
4	Original + un snapshot, dos entregables	Adquiere el base intacto y luego encadena a la reconstrucción	Comparación entre dos momentos de la misma máquina

El encadenamiento del modo 4 usa dos banderas: `VMW_CHAIN_RECONSTRUCT` marca la intención y `_vmw_base_ok` confirma que la primera adquisición terminó bien. Solo si ambas son ciertas se salta a la reconstrucción, y se hace con `VMW_RECON_SKIP_SELECT=1` para no volver a preguntar por la VM.

Los dos niveles de bloqueo

```
for /f "skip=1 delims=" %%L in ('"!VMRUN_EXE!" list 2^>nul') do (
    if /I "%%L"=="!SELECTED_VM_VMX!" set "_vmwr_is_running=1"
)
…
"!VMRUN_EXE!" stop "!SELECTED_VM_VMX!" soft
:: espera hasta 60 s reconsultando cada 3 s
```

VMWARE_CHECK_AND_STOP_IF_RUNNING: primer nivel, la VM encendida.

`vmrun list` se usa **sin pipes** a propósito: se evita así por completo la clase de fallo de comillas con rutas que contienen espacios combinadas con un pipe dentro de un `for /f`, ya corregida en otras partes del script.

POR QUÉ IMPORTA

soft frente a hard. `soft` envía una señal ACPI al SO invitado a través de VMware Tools: equivale a pulsar el botón de apagado, el sistema cierra sesiones y desmonta el sistema de archivos limpiamente. `hard` es el equivalente a tirar del cable, y dejaría el sistema de archivos **sucio**, con journal pendiente, lo que además de alterar la evidencia complicaría su análisis posterior. El script solo usa `soft`, y siempre con confirmación del operador.

Tras 60 segundos sin éxito explica las dos causas reales, que son diálogos de «guardar cambios» dentro del guest bloqueando el cierre, o VMware Tools sin responder, y cancela en lugar de forzar.

```
:VWL_CHECK
if not exist "!_vwl_path!.lck" exit /b 0
:: [1] apagar automáticamente  [2] lo hago yo y reintento  [0] cancelar
```

VMWARE_HANDLE_DISK_LOCKED: segundo nivel, el bloqueo del disco concreto.

POR QUÉ IMPORTA

Por qué una segunda comprobación. `vmrun list` solo detecta VMs **encendidas**. Una VM *suspendida* también mantiene el `.vmdk` bloqueado pero no aparece en esa lista: sin este segundo chequeo, vdiskmanager fallaría con un error opaco. La comprobación se hace sobre el archivo `.lck` del **disco concreto seleccionado**, y no sobre el `.vmx.lck`: ese último es un bloqueo de *configuración* que Workstation mantiene mientras la pestaña siga abierta en su biblioteca, incluso con la VM apagada y el disco perfectamente libre. Confundirlos bloquearía adquisiciones válidas.

El bucle es persistente: tras un apagado fallido vuelve a `:VWL_CHECK` y ofrece las opciones de nuevo, sin dejar al operador sin salida ni continuar sobre un disco bloqueado.

La consolidación

```
"!VMDK_EXE!" -r "!vmdk_src!" -t 0 "!vmdk_dest!"
```

FLAG	FUNCIÓN
-r	*Rebuild*: reconstruye el disco resolviendo la cadena completa de padres.
-t 0	Tipo de disco de salida. **0 = monolítico plano** (*single growable virtual disk* sin dispersión): un único archivo con todos los sectores.

> POR QUÉ IMPORTA
>
> **Por qué -t 0 y no otro tipo.** VMware admite varios tipos (0 monolítico disperso, 1 segmentado disperso, 2 monolítico preasignado, 3 segmentado preasignado, 4 ESX preasignado, 5 stream-optimized). El valor 0 produce **un solo archivo**, sin descriptor separado ni extents múltiples. Eso importa porque las herramientas de análisis forense, Autopsy, The Sleuth Kit o FTK, trabajan mucho mejor con un archivo único: no hay que mantener juntos descriptor y extents, ni preocuparse por rutas relativas rotas al mover la evidencia.

Tras consolidar se preserva siempre la configuración (`*.vmx *.vmxf *.vmsd *.nvram` a `Config\`, con su propio hash) y, en modo 1, la carpeta `Snapshots\` con la cadena diferencial intacta. Al final se ofrece la conversión a RAW con QEMU.

7.5 Reconstrucción de un snapshot (VM_VMWARE_RECONSTRUCT)

Genera un VMDK nuevo que representa el estado de la VM **en el momento exacto de un snapshot**, acompañado de la RAM de ese mismo instante si existe. Es el flujo que mejor ilustra el principio de «no tocar el original».

1. **Hash del origen.** `HASH_EVIDENCE FOLDER "<VM>" "*.vmdk" … hashes_originales_PREcopia.sha256`, para dejar constancia del estado de la evidencia primaria antes de nada.
2. **Copia de la cadena completa** a `_trabajo_<snapshot>\` con `robocopy … *.vmdk /NFL /NDL /NJH /NJS`. Se copia *toda* la cadena porque vdiskmanager necesita resolver desde el snapshot elegido hasta la base.
3. **Consolidación sobre la copia:** `vmware-vdiskmanager -r "<copia>\<snap>.vmdk" -t 0 "<salida>\<snap>_reconstruido.vmdk"`.

4. **Hash del resultado** como `hashes_evidencia_derivada.sha256`, en un archivo con nombre distinto del anterior para que la distinción entre evidencia primaria y evidencia derivada quede explícita.
5. **Preservación de la memoria del snapshot:** el `.vmsn` localizado en el `.vmsd` y su `.vmem` homónimo, copiados a `Consolidado_<snap>\Memoria\` con hash propio.
6. **Preservación de la configuración** a `Consolidado_<snap>\Config\`.

La copia de trabajo con sufijo `_trabajo_<snapshot>\` es el corazón del flujo: vdiskmanager escribe y reordena metadatos al resolver una cadena, así que nunca se le deja actuar sobre los archivos originales. Trabaja sobre un duplicado, y la carpeta original queda tal como se hasheó en el primer paso.

```
:: Pasada 1: encontrar el bloque snapshotN que referencia este .vmdk
for /f "usebackq tokens=1,* delims==" %%K in ("!_fm_vmsd!") do (
    echo !_fm_key! | findstr /I "disklist" >nul
    if !errorlevel! equ 0 (
        if /I "!_fm_val!"=="!_fm_vmdk!" (
            for /f "tokens=1 delims=." %%P in ("!_fm_key!") do set "_fm_prefix=%%P"
        )
    )
)

:: Pasada 2: con el prefijo, leer su archivo de estado y su nombre visible
if /I "!_fm_key2t!"=="!_fm_prefix!.filename"    set "_fm_vmsn=!_fm_val2:"=!"
if /I "!_fm_key2t!"=="!_fm_prefix!.displayName" set "_fm_dispname=!_fm_val3:"=!"
```

VMWARE_FIND_SNAPSHOT_MEMORY: dos pasadas sobre el `.vmsd`.

El `.vmsd` tiene una estructura de bloques `snapshot0`, `snapshot1` y sucesivos, cada uno con sus claves. La primera pasada busca la clave `snapshotN.disklistM.fileName` cuyo valor coincide con el `.vmdk` buscado y extrae el prefijo `snapshotN`. La segunda usa ese prefijo para leer `snapshotN.filename`, que es el `.vmsn`, y `snapshotN.displayName`, el nombre que el operador puso al snapshot en la interfaz de VMware.

> NOTA
>
> **Valor del displayName.** Mostrar `Ubuntu.final-000002.vmdk` junto a su rótulo «Antes de instalar el malware» en lugar de solo el nombre de archivo cambia por completo la usabilidad: el operador reconoce el punto temporal que busca. Ese nombre solo existe en el `.vmsd`; el descriptor `.vmdk` no lo contiene.

El `.vmem` comparte nombre base con el `.vmsn`, convención de VMware, y **solo existe si la VM estaba encendida o suspendida** al tomar el snapshot. Si no está, el script lo explica con el mensaje «Normal si la VM estaba APAGADA al tomar el snapshot (solo hay estado de disco)» en lugar de reportarlo como error.

7.6 Clon completo (opción 3)

Copia la VM entera sin consolidar nada. La diferencia conceptual con la opción 2 es la razón de que ambas existan:

	CONSOLIDAR DISCO (OPCIÓN 2)	CLON COMPLETO (OPCIÓN 3)
Resultado	Un `.vmdk` plano nuevo	La carpeta de la VM tal cual, íntegra
Incluye memoria	No	Sí, los `.vmsn` y `.vmem` de los snapshots
Vínculo con la cadena	Ninguno: es un archivo nuevo	Intacto: CID/UUID siguen coincidiendo
Uso previsto	Análisis en Autopsy / FTK / Sleuth Kit	Reabrir en VMware y «Go to snapshot» para **reanudar** con la RAM real

```
set "_vmwc_list=%vmwc_sel:,= %"        :: comas -> espacios, expansión INMEDIATA

for /L %%k in (1,1,!vmdk_count!) do set "VMDK_NEEDED_%%k=!VMDK_CHOSEN_%%k!"
:VMWC_ANCESTOR_LOOP
set "_vmwc_changed=0"
for /L %%k in (1,1,!vmdk_count!) do (
    if "!VMDK_NEEDED_%%k!"=="1" if "!VMDK_ISBASE_%%k!"=="0" (
        for /L %%j in (1,1,!vmdk_count!) do (
            if /I "!VMDK_NAME_%%j!"=="!VMDK_PARENT_%%k!" if "!VMDK_NEEDED_%%j!"=="0" (
                set "VMDK_NEEDED_%%j=1" & set "_vmwc_changed=1"
            )
        )
    )
)
if "!_vmwc_changed!"=="1" goto VMWC_ANCESTOR_LOOP
for /L %%k in (1,1,!vmdk_count!) do if "!VMDK_ISBASE_%%k!"=="1" set "VMDK_NEEDED_%%k=1"
```

VMWARE_SELECT_CLONE_SNAPSHOTS: selección del operador y cierre de ancestros.

Es un **bucle de punto fijo**: repite el barrido hasta que una pasada completa no marque ningún disco nuevo como necesario. Así se cierra la clausura transitiva de ancestros sin recursión, que en lotes sería frágil. Los discos base

se marcan siempre al final, porque son la raíz de cualquier cadena.

> NOTA
>
> **Peculiaridad de cmd documentada.** La sustitución de comas por espacios se hace en una variable aparte con **expansión inmediata** (`%vmwc_sel:,= %`) y no con `!var:,= !` dentro del `for`. Esa combinación es poco fiable: el `for` tokeniza el texto antes de que la sustitución retardada se resuelva correctamente en esa posición. Verificado con reproducción aislada.

Asimetría deliberada: los discos ancestros se incluyen de forma automática porque VMware necesita la cadena completa para resolver cualquier snapshot; en cambio la **memoria solo se incluye para los snapshots elegidos explícitamente**, no para los ancestros arrastrados por necesidad de disco. Un `.vmem` puede pesar 16 GB, y copiar el de un ancestro que nadie va a examinar sería un coste sin retorno.

Para cada disco necesario se añaden también sus extents: `-flat.vmdk`, `-delta.vmdk` y todos los `-s*.vmdk`, comprobando la existencia uno por uno.

La copia se hace en dos pasadas cuando hay filtro activo, primero la configuración completa, que es pequeña, y luego los discos y la memoria seleccionados, y en una sola pasada con `/E` cuando no hay filtro. Se hashea antes y después con el mismo patrón de siete extensiones, y el script imprime al final la instrucción de uso, incluyendo el detalle de elegir **«I Copied It»** al abrir el clon para que VMware regenere los UUID y no entre en conflicto con la VM original.

CAPÍTULO 8

Módulo 4. VirtualBox

Tres flujos de trabajo: adquisición de disco con cuatro modos de snapshot y cinco formatos de salida, volcado de memoria en formato ELF, y clon completo. Todo pasa por `VBoxManage`, que se resuelve en cascada antes de tocar nada.

8.1 Resolución del binario en cascada

RESOLVE_VBOX (infraestructura)

```
set "VBOX_EXE=!BASE_PATH!\Virtualizacion\VirtualBox\VBoxManage.exe"
if exist "!VBOX_EXE!" exit /b 0
if exist "%ProgramFiles%\Oracle\VirtualBox\VBoxManage.exe" (
    set "VBOX_EXE=%ProgramFiles%\Oracle\VirtualBox\VBoxManage.exe"
    exit /b 0
)
for %%P in (VBoxManage.exe) do if not "%%~$PATH:P"=="" ( set "VBOX_EXE=%%~$PATH:P" & exit /b 0 )
echo [-] No se encontro VBoxManage.exe (bundled, instalacion estandar ni PATH).
exit /b 1
```

Tres niveles en orden de preferencia: la copia embebida en el framework, la instalación estándar de Oracle y el PATH del sistema. Se prefiere la embebida por trazabilidad, porque así se sabe exactamente qué versión se usó, pero se acepta la del sistema porque `VBoxManage` debe coincidir con la versión de VirtualBox que creó los discos si se quieren evitar incompatibilidades de formato. Si no aparece en ninguno de los tres sitios, la subrutina devuelve código 1 y el llamador vuelve al menú con un mensaje que explica las dos soluciones posibles.

8.2 Enumeración y estado

Listado de VMs y determinación de estado (enumeración)

```
:: "Nombre" {uuid}  ->  se parte por la llave de apertura
for /f "tokens=1,* delims={" %%A in ('"!VBOX_EXE!" list vms 2^>nul') do (
    set "vb_raw_uuid=%%B"  &  set "vb_raw_name=%%A"
    if defined vb_raw_uuid (
        set "vb_raw_uuid=!vb_raw_uuid:}=!"      :: quita la llave de cierre
        set "VB_UUID_!vb_count!=!vb_raw_uuid!"
        set "vb_nm=!vb_raw_name:~1,-2!"         :: quita comillas y espacio final
        set "VB_NAME_!vb_count!=!vb_nm!"
    )
)
```

La salida de `list vms` tiene el formato `"Nombre de la VM" {uuid-con-guiones}`. Se parte usando `{` como delimitador: el token 1 es el nombre entrecomillado y el resto es el UUID con su llave de cierre, que se elimina por sustitución. El nombre se limpia con `~1,-2`, que descarta la comilla inicial y los dos últimos caracteres, comilla y espacio.

VBOX_GET_VM_STATE_LABEL (enumeración)

```
findstr /I /X /C:"%~1" "%~2" >nul 2>&1
if !errorlevel! equ 0 ( set "_vb_state=Encendida" & exit /b 0 )

for /f "tokens=1,* delims==" %%A in ('""!VBOX_EXE!" showvminfo "%~1" --machinereadable 2^>nul ^
                                     ^| findstr /I "^VMState=""') do set "_vgs_vmstate=%%B"
if /I "!_vgs_vmstate!"=="saved"  set "_vb_state=Suspendida"
if /I "!_vgs_vmstate!"=="paused" set "_vb_state=Pausada"
```

`list runningvms` solo reporta las máquinas encendidas y no distingue una suspendida de una apagada. Para las que no aparecen en esa lista se consulta puntualmente la propiedad oficial `VMState`, que devuelve `running`, `paused`, `saved`, `poweroff`, `aborted` y otros valores. La consulta se hace solo sobre esas máquinas, no sobre todas, para no multiplicar invocaciones de `VBoxManage` en inventarios grandes.

NOTA

El envoltorio de comillas dobles merece atención. En el patrón `'""!VBOX_EXE!" … ^| findstr …""'` hay un par **extra** de comillas envolviendo el comando completo. Es necesario: al combinar una ruta con espacios entrecomillada con un pipe dentro de un `for /f`, cmd.exe pierde el pareo de comillas del ejecutable e intenta ejecutar la primera parte de la ruta como si fuera un comando. El fallo apareció con datos reales de un usuario cuya carpeta se llamaba «Proyecto final», y se reprodujo después de forma aislada. El mismo patrón se repite en varios puntos del script.

8.3 El parseo de --machinereadable

Extracción de propiedades de la VM (enumeración)

```
for /f "tokens=1,* delims==" %%A in ('"!VBOX_EXE!" showvminfo "!VB_SEL_UUID!" --machinereadable
2^>nul') do (
    set "vb_key=%%A"  &  set "vb_val=%%B"
    echo !vb_key! | findstr /i "ImageUUID" >nul 2>&1
    if !errorlevel! equ 0 (
        set "vb_val=!vb_val:~1,-1!"        :: quita las comillas del valor
        if defined vb_val ( set /a vb_disk_count+=1 & set
"VB_DISK_UUID_!vb_disk_count!=!vb_val!" )
    )
)
```

POR QUÉ IMPORTA

Fallo real confirmado contra VirtualBox. El síntoma era que una VM con disco adjunto reportaba «no se encontraron discos». La causa: `--machinereadable` emite `clave="valor"` **usando `=` como separador, nunca `:`**. Con `delims=:` no hay ningún delimitador en la línea, así que el token del valor (`%%B`) queda siempre vacío y el UUID no se detecta jamás, exista el disco o no. El mismo error estaba presente en la localización del disco del clon temporal de snapshots, y se corrigió en los dos sitios.

Cuatro propiedades se extraen con este patrón:

- `ImageUUID`: UUID del disco adjunto, y puede haber varios.
- `VMState`: estado actual de la máquina.

- `CfgFile`: ruta absoluta del `.vbox`; su carpeta contenedora es la carpeta de la VM.
- `SnapshotFolder`: carpeta donde VirtualBox guarda los discos diferenciales de los snapshots.

Si no se detecta ningún `ImageUUID`, hay una vía de reserva por extensión de archivo:

```
for /f "tokens=1,* delims==" %%A in ('""!VBOX_EXE!" showvminfo … ^| findstr /i ".vdi .vmdk
.vhd""') do …
```

> NOTA
>
> Esa misma vía de reserva tuvo un segundo fallo. El patrón original era `"\.vdi\> …"`. Sin el flag `/R`, `findstr` hace coincidencia **literal**: buscaba un backslash, un punto, «vdi» y un carácter `>` que no existe en el valor real, porque la línea termina en comilla, por ejemplo `… Proyecto.Final.vdi"`. Nunca coincidía. Se corrigió a coincidencia literal de la extensión sin la comilla de cierre, para no complicar más el escapado que ya exige el envoltorio del pipe.

8.4 Los cuatro modos de snapshot y la cadena de discos

El conteo de snapshots usa `findstr` y no `find /C`. El motivo es el flag `/C:` de `find.exe`, con los dos puntos pegados a la comilla, que rompe el envoltorio de comillas dobles necesario para pipes con rutas que contienen espacios:

```
for /f "tokens=1" %%C in ('""!VBOX_EXE!" snapshot "!VB_SEL_UUID!" list 2^>nul ^| findstr /I
"Name:""') do set /a vb_snap_count+=1
```

MODO	QUÉ ADQUIERE	MECANISMO
1	Disco completo (estado actual, snapshots aplicados)	`clonemedium` sobre el `ImageUUID` adjunto, más la copia de la carpeta de snapshots
2	Solo el disco base	`VBOX_FIND_BASE_UUID` sube la cadena y clona la raíz
3	Solo un snapshot concreto	`VBOX_CLONE_SNAPSHOT_DISK`: clon temporal de la VM en ese punto
4	Disco base más un snapshot, dos archivos	Primero la raíz, después el snapshot como segundo entregable independiente

VBOX_FIND_BASE_UUID: recorrido de la cadena de padres (enumeración)

```
:: Fase 1: cargar todos los medios registrados en arrays paralelos
for /f "usebackq tokens=1,* delims=:" %%K in (`""!VBOX_EXE!" list hdds 2^>nul"`) do (
    :: bloques separados por línea en blanco, con claves "UUID:" y "Parent UUID:"
)

:: Fase 2: caminar de hijo a padre hasta la raíz
:VBB_WALK_LOOP
for /L %%i in (1,1,!_vbb_hcount!) do if /I "!_VBB_HUUID_%%i!"=="!_vbb_walk!" set
"_vbb_found_parent=!_VBB_HPARENT_%%i!"
if not defined _vbb_found_parent exit /b 0
if /I "!_vbb_found_parent!"=="base" exit /b 0      :: raíz alcanzada
set "VBOX_BASE_UUID=!_vbb_found_parent!"
set /a _vbb_hops+=1
if !_vbb_hops! lss 50 goto VBB_WALK_LOOP
```

`list hdds` lista todos los medios registrados en bloques con las claves `UUID:` y `Parent UUID:`. Aquí sí se usa `delims=:`, porque este comando **no** es `--machinereadable` y su formato es `Clave: valor`. La detección de la clave usa `findstr /X /I "UUID"`, es decir coincidencia de línea completa, para no confundir `UUID` con `Parent UUID`.

> POR QUÉ IMPORTA
>
> Dos salvaguardas gobiernan el recorrido. VirtualBox usa el literal **`base`** como valor de `Parent UUID` para indicar «sin padre»: esa es la condición de parada correcta. El límite de **50 saltos** protege contra una cadena corrupta o cíclica, que de otro modo colgaría el script en un bucle infinito. Y si la cadena no se puede determinar, la subrutina devuelve el mismo UUID que recibió, sin cambios, para no romper el flujo del llamador.

8.5 Clonado del disco

clonemedium con formato a elección (adquisición)

```
"!VBOX_EXE!" clonemedium disk "!_vb_clone_src!" "!vb_dest!" !VB_FMT_FLAG!
```

OPCIÓN	FLAG	EXTENSIÓN	CUÁNDO ELEGIRLO
1. Original	(ninguno)	la del origen	Máxima fidelidad al formato de la evidencia
2. VDI	`--format VDI`	`.vdi`	Nativo de VirtualBox, para reabrir en el mismo hipervisor
3. VMDK	`--format VMDK`	`.vmdk`	Compatibilidad con Autopsy, FTK y VMware
4. VHD	`--format VHD`	`.vhd`	Montaje nativo en Windows y Hyper-V
5. RAW	`--format RAW`	`.raw`	Análisis forense universal, sin capa de formato

La extensión de salida se determina en cascada: por defecto `.vdi`, se sustituye por la del disco de origen si se conoce su ruta, y finalmente manda la del formato elegido si el operador pidió conversión. El comando se ejecuta a través de `:ACQUIRE`, así que es cancelable con confirmación.

VBOX_CLONE_SNAPSHOT_DISK: extraer un snapshot sin tocarlo (adquisición)

```
call :NEWGUID _vbcs_guid
set "_vbcs_tmpname=ForensicTemp_!_vbcs_guid!"
set "_vbcs_tmpfolder=!TEMP!\vbox_snap_work_!_vbcs_guid!"

:: 1) Clonar la VM completa en el punto del snapshot
"!VBOX_EXE!" clonevm "!_vbcs_vm!" --snapshot "!_vbcs_snap!" --mode machine ^
    --name "!_vbcs_tmpname!" --basefolder "!_vbcs_tmpfolder!" --register

:: 2) Localizar el disco del clon y extraerlo al destino final
"!VBOX_EXE!" clonemedium disk "!_vbcs_srcuuid!" "!_vbcs_dest!" !_vbcs_fmt!

:: 3) Desregistrar y borrar el clon temporal
"!VBOX_EXE!" unregistervm "!_vbcs_tmpname!" --delete
rmdir /s /q "!_vbcs_tmpfolder!"
```

POR QUÉ IMPORTA

La ruta indirecta tiene explicación. No existe un comando de VirtualBox que exporte directamente «el disco tal como estaba en el snapshot N». Lo que sí existe es `clonevm --snapshot`, el **método oficial** para materializar el estado de una VM en un punto concreto. Con `--mode machine` el clon obtiene un disco **independiente y aplanado**, ya resuelto contra toda su cadena de padres. De ahí se extrae el disco al destino final y el clon temporal se destruye: era solo un área de trabajo de solo lectura sobre el snapshot, y no evidencia en sí mismo. **El snapshot y la VM original nunca se modifican.**

Los nombres del clon temporal y de su carpeta llevan GUID, de modo que dos instancias del framework trabajando a la vez no colisionan. El resultado se comunica al llamador en `VBCS_OK` (1 éxito, 0 error), y la limpieza se ejecuta en cualquier caso.

8.6 VM encendida: la decisión de no apagar automáticamente

VBOX_HANDLE_VM_RUNNING (control)

```
:VBXR_CHECK
"!VBOX_EXE!" list runningvms 2>nul | findstr /I "!_vbxr_uuid!" >nul 2>&1
if !errorlevel! neq 0 exit /b 0        :: no está en la lista -> apagada, continuar

echo [-] La VM "!_vbxr_name!" esta en ejecucion.
echo [i] Debe estar apagada para garantizar la integridad forense.
echo [i] Apaguela usted mismo desde dentro de la VM (este framework no puede
echo [i] garantizar un apagado automatico en VirtualBox…)
echo [1] Ya la apague - reintentar verificacion
echo [0] Cancelar
```

POR QUÉ IMPORTA

Aquí el framework se aparta a propósito de lo que hace con VMware y con Hyper-V, y la razón está escrita en el propio script. `vmrun stop soft` usa VMware Tools para invocar directamente el script de apagado del sistema operativo invitado, sin pasar por el entorno de escritorio. En cambio `VBoxManage acpipowerbutton` es una señal ACPI real cuya respuesta depende de cómo la maneje el guest, y se verificó en pruebas reales que GNOME puede interceptarla y mostrar un diálogo de confirmación («Goodbye, ¿qué deseas hacer?») que exige interacción manual **dentro** de la VM, algo que `VBoxManage` no puede forzar desde el host. La alternativa, `poweroff`, equivale a cortar la corriente y dejaría el sistema de archivos sucio. Ante la imposibilidad de garantizar un apagado limpio, el framework pide al operador que lo haga él y reintenta la verificación. Es honestidad técnica: prefiere no ofrecer una automatización que no puede cumplir.

8.7 Volcado de memoria: el formato ELF

debugvm dumpvmcore (adquisición)

```
set "vb_mem_dest=!VBM_EVIDENCE!\Memory\!VBR_SEL_NAME!.elf"
"!VBOX_EXE!" debugvm "!VBR_SEL_UUID!" dumpvmcore --filename "!vb_mem_dest!"
```

Solo se listan las VMs en ejecución (`list runningvms`), porque volcar la RAM requiere que la máquina esté encendida. A diferencia de VMware, aquí no hay que suspender: el depurador del hipervisor congela la VM el tiempo necesario para escribir el volcado.

> ADVERTENCIA
>
> La extensión importa, y el comentario del código lo marca como corregido: `dumpvmcore` produce un **ELF Core Dump**, no memoria RAW. La extensión `.elf` se eligió deliberadamente para evitar la confusión. Si el archivo se llamara `.raw`, un analista lo cargaría en Volatility como imagen física plana, obtendría resultados incoherentes y probablemente concluiría que la evidencia está corrupta, cuando el problema sería solo la interpretación del formato.

Y para que ese conocimiento no dependa de la memoria de nadie, el script escribe una nota junto a la evidencia:

```
FORMATO: ELF Core Dump (NO RAW memory dump)
GENERADO POR: VBoxManage debugvm dumpvmcore

ANALISIS CON VOLATILITY:
  vol.py -f "<archivo>.elf" --profile=WIN_PROFILE vboxelf [plugin]
  Requiere: perfil del SO invitado instalado en Volatility

ALTERNATIVA - Conversion a RAW:
  objcopy -I elf64-x86-64 -O binary "<archivo>.elf" output.raw
```

Contenido de `FORMAT_NOTE.txt`, escrito en la misma carpeta `Memory` que el volcado.

El archivo se llama `FORMAT_NOTE.txt` y vive junto al volcado. Incluye la capa de dirección correcta de Volatility (`vboxelf`) y la vía de conversión a RAW con `objcopy`, por si se prefiere trabajar con memoria plana.

8.8 Preservación de configuración y snapshots

```
:: Configuración: el .vbox localizado vía CfgFile
copy /y "!vb_cfgfile!" "!VB_EVIDENCE!\Config\" >nul
call :HASH_EVIDENCE FOLDER "!VB_EVIDENCE!\Config" "*.vbox" … "VirtualBox_Configuracion"

:: Snapshots (solo en modo 1): discos diferenciales desde SnapshotFolder
robocopy "!vb_snap_folder!" "!VB_EVIDENCE!\Snapshots" *.vdi *.vmdk *.vhd /NFL /NDL /NJH /NJS
call :HASH_EVIDENCE FOLDER "!VB_EVIDENCE!\Snapshots" "*.vdi,*.vmdk,*.vhd" …
"VirtualBox_Snapshots"
```

El `.vbox` es el equivalente funcional del `.vmx` de VMware: un XML con la definición completa del hardware virtual y las referencias a los discos. Sin él, VirtualBox no puede registrar ni abrir la máquina después. Los discos diferenciales de los snapshots se copian **sin alterarlos**, y cada conjunto recibe su propio archivo de hashes, de modo que disco principal, configuración y snapshots quedan verificables por separado.

8.9 Clon completo (opción 3)

Mismo patrón que en VMware, con la carpeta localizada a partir de `CfgFile`:

```
for %%F in ("!vbc_cfgfile!") do set "VBC_VMFOLDER=%%~dpF"
if "!VBC_VMFOLDER:~-1!"=="\" set "VBC_VMFOLDER=!VBC_VMFOLDER:~0,-1!"

call :HASH_EVIDENCE FOLDER "!VBC_VMFOLDER!" "*.vdi,*.vmdk,*.vhd,*.vbox,*.vbox-prev,*.sav" ^
     "…\hashes_originales_PREcopia.sha256" "VirtualBox_Original_PreCopia_ClonCompleto"

robocopy "!VBC_VMFOLDER!" "!VBC_EVIDENCE!\VM" /E /COPY:DAT /R:1 /W:1 /NFL /NDL ^
    & if errorlevel 8 (exit 1) else (exit 0)

call :HASH_EVIDENCE FOLDER "!VBC_EVIDENCE!\VM" "…" "…\hashes_copia_POSTcopia.sha256"
"VirtualBox_Copia_ClonCompleto"
```

El patrón de hashes incluye `*.vbox-prev`, la copia de seguridad automática de la configuración que hace VirtualBox, y `*.sav`, los estados guardados. El script imprime al final la instrucción de uso, `VBoxManage registervm "<ruta>\<maquina>.vbox"`, y la advertencia de que el clon es una copia **operativa** y no una imagen de solo lectura, mientras los originales permanecen intactos con hash verificado antes y después.

CAPÍTULO 9

Módulo 4. Hyper-V

Es el módulo técnicamente más complejo, porque Hyper-V no ofrece ningún equivalente directo al volcado de RAM de VMware o VirtualBox. Cuatro flujos lo componen: discos con cadena de checkpoints, memoria con dos métodos de captura y conversión a RAW por API oficial, consolidación con Merge-VHD, y clon completo.

9.1 Requisitos y validación del entorno

Los tres cmdlets se validan en una sola llamada

```
powershell -NoProfile -Command "if ((Get-Command Get-VM,Get-VMHardDiskDrive,Get-VMSnapshot ^
    -ErrorAction SilentlyContinue).Count -eq 3) { exit 0 } else { exit 1 }"
```

> POR QUÉ IMPORTA
>
> El comentario del código explica por qué la consulta va agrupada: cada llamada separada a `Get-Command` recargaría el módulo completo de Hyper-V, lo que cuesta **aproximadamente el triple de tiempo**. Pedir los tres cmdlets a la vez y comparar el recuento con 3 obtiene el mismo resultado en un solo arranque de PowerShell.

Si falta alguno, el mensaje de error nombra los tres cmdlets requeridos y recuerda las dos condiciones necesarias: el rol de Hyper-V habilitado y la ejecución como Administrador. La detección previa del arranque, `sc query vmms`, ya había marcado la disponibilidad general del servicio, pero esta validación es más específica: comprueba que los cmdlets estén realmente accesibles en esta sesión.

El escapado de nombres de VM para PowerShell

```
set "_hvr_name_ps=!_hvr_name:'=''!"
```

POR QUÉ IMPORTA

El apóstrofo es un carácter **válido** en un nombre de VM de Hyper-V («Servidor de Juan's Lab»). Al interpolarlo dentro de un literal PowerShell de comilla simple, es decir `Get-VM -Name '<nombre>'`, cerraría la cadena antes de tiempo y produciría un error de sintaxis. La convención de PowerShell para escapar una comilla simple dentro de un literal es **duplicarla**. El script mantiene dos variables paralelas para cada VM: el nombre real, que se muestra al operador, y el nombre escapado (`_PS`), que es el que se interpola. El patrón se aplica en `HV_HANDLE_VM_RUNNING`, `HV_DISK`, `HV_MEM` y `HV_CLONE_FULL`.

9.2 Adquisición de discos

Enumeración

```
for /f "tokens=1,* delims=|" %%A in ('powershell -Command ^
        "Get-VM | ForEach-Object { $_.Name + '|' + $_.State }" 2^>nul') do (
    set /a hv_count+=1
    set "HV_NAME_!hv_count!=%%A"   &  set "HV_STATE_!hv_count!=%%B"
)
```

La salida se construye con un separador explícito `|` en lugar de usar el formato de tabla de PowerShell: así el parseo con `delims=|` es exacto y no depende del ancho de columna ni de los espacios de alineación. El mismo idioma se usa en el módulo de memoria, que añade un tercer campo (`$_.Version`, la versión de configuración de la VM), y en el clon completo.

Los tres modos de checkpoint

```
function Get-VhdChain($p) {
    $chain = @($p)
    $cur = $p
    while ($true) {
        try { $info = Get-VHD -Path $cur -ErrorAction Stop } catch { break }
        if ($info.ParentPath) { $chain += $info.ParentPath; $cur = $info.ParentPath } else {
break }
    }
    return $chain
}

if ($mode -eq 3) {                              :: solo un checkpoint
    $snap = Get-VMSnapshot -VMName $vmName -Name $chosenSnap
    $snapDisks = $snap | Get-VMHardDiskDrive
    foreach ($d in $snapDisks) { $paths += Get-VhdChain $d.Path }
} elseif ($mode -eq 2) {                        :: solo el disco base
    foreach ($d in (Get-VMHardDiskDrive -VMName $vmName)) {
        $chain = Get-VhdChain $d.Path
        $paths += $chain[-1]                    :: último elemento = raíz
    }
} else {                                        :: modo 1: cadena adjunta actual
    $paths += (Get-VMHardDiskDrive -VMName $vmName).Path
}
if ($paths.Count -gt 0) { $paths | Sort-Object -Unique } else { Write-Output 'NONE' }
```

Resolución de la cadena de discos según el modo elegido.

Toda la lógica se resuelve en PowerShell y no en lotes, escrita a un `.ps1` temporal con nombre GUID. El comentario del código señala la razón: evita los problemas de comillas con nombres de VM que contienen espacios, que en una línea de comando de una sola invocación serían difíciles de escapar correctamente.

MODO	DISCOS QUE DEVUELVE	SIGNIFICADO FORENSE
1	Los discos adjuntos ahora mismo (`.avhdx` activo si hay checkpoints)	Estado actual de la máquina
2	Solo `$chain[-1]`: la raíz de cada cadena	Estado previo a cualquier checkpoint
3	El disco del checkpoint elegido **más toda su cadena de ancestros**	Ese punto temporal; sin consolidar, para eso existe la opción Merge-VHD

`Sort-Object -Unique` elimina duplicados: si dos discos de la VM comparten un padre común, ese padre aparecería dos veces. `'NONE'` es el centinela que el `.bat` reconoce para saber que no hubo resultados. La marca `-ErrorAction`

`SilentlyContinue` y el `try/catch` alrededor de `Get-VHD` evitan que un disco inaccesible aborte la enumeración completa.

Copia de los discos

```
for %%F in ("!hv_copy_src!") do ( set "_hvc_dir=%%~dpF" & set "_hvc_file=%%~nxF" )
if "!_hvc_dir:~-1!"=="\" set "_hvc_dir=!_hvc_dir:~0,-1!"

robocopy "!_hvc_dir!" "!HV_EVIDENCE!\Disks" "!_hvc_file!" !RBC_J! /NJH /NJS /NDL /NC /NS ^
    & if errorlevel 8 (exit 1) else (exit 0)
```

`HV_COPY_DISK_ITEM`: descomposición de la ruta y copia de un disco a la carpeta de evidencia.

> **POR QUÉ IMPORTA**
>
> El comentario del código lo indica de forma explícita: los `.vhdx` pueden ser **enormes** y `copy /B` no era interrumpible de forma segura por el wrapper. Robocopy aporta `/J` para E/S sin buffer y se integra limpiamente en el mecanismo de suspensión y terminación del árbol de procesos. El precio es tener que descomponer la ruta en carpeta y nombre, porque robocopy no acepta una ruta de archivo completa como origen.

El bucle de copia usa dos banderas de control. `hv_copy_ok` se activa si al menos un disco se copió, de modo que una cadena parcialmente inaccesible no se reporta como fallo total. `hv_copy_abort` corta el resto de iteraciones si el operador abortó, sin intentar los discos siguientes.

Preservación de la configuración

```
for /f "usebackq delims=" %%C in (`powershell -NoProfile -Command ^
    "(Get-VM -Name '!HV_VMNAME_PS!').ConfigurationLocation"`) do set "hv_cfgfolder=%%C"

if defined hv_cfgfolder if exist "!hv_cfgfolder!\Virtual Machines" (
    robocopy "!hv_cfgfolder!\Virtual Machines" "!HV_EVIDENCE!\Config" /E /NFL /NDL /NJH /NJS
>nul
    call :HASH_EVIDENCE FOLDER "!HV_EVIDENCE!\Config" "*.vmcx,*.vmrs,*.xml" …
"HyperV_Configuracion"
)
```

La carpeta `Virtual Machines` contiene los archivos nombrados por el GUID de la máquina: `.vmcx` (configuración binaria en VMs modernas), `.vmrs` (estado runtime) y `.xml` en VMs legacy o de Generación 1. Son pequeños pero imprescindibles: sin ellos Hyper-V no puede importar la VM después.

9.3 Volcado de memoria: el problema y su solución

> ADVERTENCIA
>
> Hyper-V **no tiene equivalente a** `.vmem` (VMware) ni a `debugvm dumpvmcore` (VirtualBox). La RAM del guest solo se materializa en disco a través del mecanismo de estado guardado.

Ese estado guardado adopta dos formatos distintos según la versión de configuración de la VM:

- **Configuración 6.2 o superior** (Windows Server 2016, Windows 10 y posteriores): un único `<Id>.vmrs` con la RAM completa sin comprimir más el estado runtime; en VMs de Generación 2 recientes se añade `<Id>.vmgs`.
- **Configuración anterior a 6.2** (2008 a 2012R2, Windows 8.1): el par `<Id>.bin` (memoria) más `<Id>.vsv` (estado de dispositivos).

> ADVERTENCIA
>
> Ninguno de los dos formatos es una imagen RAW: **Volatility 3 no los abre sin conversión**. El estado quedó verificado y anotado en el código con referencia al issue volatility3#1886. La vía DFIR aceptada es leerlos con la API oficial `VmSavedStateDumpProvider` del Windows SDK, la misma que emplean MemProcFS y LeechCore. `vm2dmp` solo sirve para Hyper-V 2008R2 o anterior y falla con 4 GB o más de RAM.

Selección del método según el estado de la VM

ESTADO	ACCIÓN DEL SCRIPT	INTRUSIÓN
Off	Rechaza: «No hay memoria RAM que volcar de una VM apagada»	No aplica
Saved	`HVM_METHOD=SAVED`: copia directa de los archivos existentes	**Nula**: la RAM ya está en disco, no se toca la VM
Running / Paused	Pregunta: método A (checkpoint) o B (Save-VM)	Mínima o media, según elección

Método A. Checkpoint estándar forzado

```
$vm = Get-VM -Name '<vm>'
$oldType = $null
try { $oldType = $vm.CheckpointType.ToString() } catch {}
if ($oldType -and $oldType -ne 'Standard') {
    Set-VM -VM $vm -CheckpointType Standard
    Write-Output "OLDTYPE|$oldType"                 :: se devuelve al .bat para restaurarlo
}
$snap = Checkpoint-VM -VM $vm -SnapshotName 'DFIR_MEM_<timestamp>' -Passthru
$snapId = $snap.Id.ToString()

$dirs = @($vm.SnapshotFileLocation, $vm.ConfigurationLocation, $vm.Path) | Where-Object { $_ } |
Select-Object -Unique
foreach ($d in $dirs) {
    if (Test-Path $d) {
        $found += Get-ChildItem -Path $d -Recurse ^
            -Include "$snapId.bin","$snapId.vsv","$snapId.vmrs","$snapId.VMRS","$snapId.vmgs"
    }
}
$found | Select-Object -ExpandProperty FullName -Unique | ForEach-Object { Write-Output
"FILE|$_" }
```

ADVERTENCIA

Punto crítico verificado en la documentación de Microsoft: desde Windows Server 2016 y Windows 10 el tipo de checkpoint por defecto es **Production**, que usa VSS dentro del guest y **no incluye la RAM**. Tomar un checkpoint sin cambiar el tipo produciría un checkpoint perfectamente válido y completamente inútil para análisis de memoria. El script fuerza `Standard` temporalmente, informa del tipo anterior al `.bat` mediante la línea `OLDTYPE|…`, y lo **restaura** al terminar. En 2012R2 y anteriores la propiedad `CheckpointType` no existe: el `try/catch` lo tolera sin ruido, porque en esas versiones todos los checkpoints son estándar.

El protocolo de comunicación entre el `.ps1` y el `.bat` usa tres prefijos con separador `|`: `FILE|` por cada archivo de estado localizado, `OLDTYPE|` con el tipo a restaurar, y `PSERROR|` con el mensaje de excepción si algo falló. El `.bat` parsea con `tokens=1,* delims=|` y actúa según el prefijo.

La búsqueda de archivos recorre **tres** ubicaciones posibles, deduplicadas: `SnapshotFileLocation`, `ConfigurationLocation` y `Path`. Hyper-V permite configurarlas por separado y el estado guardado puede estar en cualquiera de ellas. Se buscan las cinco extensiones posibles, incluida `.VMRS` en mayúsculas.

HVM_RESTORE_CKTYPE: restauración garantizada

```
:HVM_RESTORE_CKTYPE
if not defined HVM_OLDCKTYPE exit /b
powershell -NoProfile -Command "Set-VM -Name '!HVM_VMNAME_PS!' -CheckpointType !HVM_OLDCKTYPE! -
ErrorAction Stop"
if !errorlevel! equ 0 ( echo [+] CheckpointType restaurado a: !HVM_OLDCKTYPE! ) ^
else ( echo [-] No se pudo restaurar CheckpointType. Restaurelo con Set-VM. )
set "HVM_OLDCKTYPE="
```

> **POR QUÉ IMPORTA**
>
> Lo relevante es dónde se llama. Esta subrutina se invoca en **todas** las rutas de salida del método A: error al crear el checkpoint, checkpoint creado pero sin archivos localizables, aborto del operador durante la copia, fallo de copia y finalización correcta. Cambiar la configuración de una VM del sistema examinado y dejarla cambiada sería una alteración no documentada del entorno; garantizar la restauración en todos los caminos es lo que hace reversible la operación. Si aun así falla, el script dice exactamente qué hacer manualmente.

Al terminar, el script también ofrece `Remove-VMSnapshot` para eliminar el checkpoint `DFIR_MEM_<timestamp>`, ya que cumplió su función una vez copiada la evidencia. Si el operador prefiere conservarlo, esa decisión se registra.

Método B. Save-VM

```
echo [ATENCION] Save-VM DETENDRA la ejecucion de la VM (quedara en Saved).
echo [i] ESTANDAR DFIR: alteracion controlada y documentada.
set /p hvm_confirm="Desea continuar? [S/N]: "
if /I not "!hvm_confirm!"=="S" ( … cancelar y registrar … )

powershell -NoProfile -Command "Save-VM -Name '!HVM_VMNAME_PS!' -ErrorAction Stop"
```

Es más intrusivo que el checkpoint, porque la VM deja de ejecutarse, pero también más simple y sin dependencia del tipo de checkpoint. Exige confirmación explícita, registra la acción con marcas de tiempo y código de salida, y al final ofrece `Start-VM` para reanudar. Tras guardar, la localización de archivos se hace por el **Id de la VM** en lugar del Id del snapshot, con la misma búsqueda en tres carpetas.

9.4 Conversión a memoria RAW por API oficial

HVM_CONVERT_RAW: selección de fuente e invocación

```
:: Preferir .vmrs (configuración >= 6.2)
for %%F in ("!HVM_EVIDENCE!\Memory\*.vmrs") do if not defined _hvsrc ( set "_hvsrc=%%F" & set
"_hvtype=vmrs" )

:: Si no hay, buscar el par .bin + .vsv (configuración < 6.2)
if not defined _hvsrc (
    for %%F in ("!HVM_EVIDENCE!\Memory\*.bin") do if not defined _hvsrc ( set "_hvsrc=%%F" & set
"_hvtype=bin" )
    if defined _hvsrc (
        for %%F in ("!HVM_EVIDENCE!\Memory\*.vsv") do if not defined _hvvsv set "_hvvsv=%%F"
        if not defined _hvvsv ( echo [-] Se encontro .bin pero no su .vsv; no se puede
convertir. & exit /b )
    )
)

powershell -File "!HV_RAW_PS1!" -SavedStatePath "!_hvsrc!" [-VsvPath "!_hvvsv!"] -OutputRaw
"!_hvraw!"
```

La conversión opera sobre los archivos **ya copiados** a la carpeta de evidencia, no sobre los originales del sistema: si algo va mal, la evidencia primaria no se ve afectada. La preferencia por `.vmrs` es deliberada. Un solo archivo con la RAM completa resulta más simple y más moderno que el par `.bin` más `.vsv`, que exige ambos y aborta limpiamente si falta uno.

El resultado se interpreta según tres códigos:

CÓDIGO	MENSAJE AL OPERADOR	CONSECUENCIA
0	«Imagen RAW generada» más la sintaxis de análisis `vol.py -f … windows.info`	Evidencia lista para Volatility 3
3	«Falta vmsavedstatedumpprovider.dll (Windows SDK)»	**Degradación limpia:** los archivos de estado quedan copiados para convertirlos en otra máquina
otro	«La conversión a RAW falló (código N)»	«Los archivos de estado originales quedan intactos como evidencia»

hv_savedstate_to_raw.ps1: la secuencia de la API

```
// P/Invoke sobre vmsavedstatedumpprovider.dll
LoadSavedStateFile(vmrs, out handle)                      // o LoadSavedStateFiles(bin, vsv, out handle)
GetGuestRawSavedMemorySize(handle, out size)              // tamaño total de la RAM del guest
ReadGuestRawSavedMemory(handle, offset, buffer, 2MB, out read)   // en bucle
ReleaseSavedStateFiles(handle)
```

Es la secuencia documentada por Microsoft en el ejemplo `rawmemtofile.cpp`. El script la implementa con `Add-Type` y `DllImport`, escribiendo cada bloque de 2 MB directamente a un `FileStream` y mostrando el progreso en porcentaje sobre la misma línea.

La resolución del DLL tiene cuatro niveles, en este orden:

1. Parámetro `-DllPath` explícito.
2. Carpeta del propio script, o `herramientas\Volcados\Virtual\vmsavedstate\` y `…\vm2dump\`.
3. Windows SDK instalado: recorre `Windows Kits\10\bin` filtrando por `\x64\` y toma la **versión más reciente** (`Sort-Object FullName -Descending`).
4. Cada directorio del `%PATH%`.

El DLL se precarga por **ruta absoluta** con `LoadLibrary` más `SetDllDirectory`, de modo que los `DllImport` posteriores, declarados solo por nombre, resuelvan contra el módulo ya cargado sin depender del directorio actual. Si `LoadLibrary` falla, se informa el error Win32 y se apunta que el DLL puede requerir componentes de Hyper-V o una versión distinta del SDK.

> POR QUÉ IMPORTA
>
> MemProcFS también puede leer estos archivos, pero requiere el **driver Dokany** instalado en el sistema para montar el sistema de archivos virtual. Instalar un driver en la máquina de análisis, o peor, en el equipo examinado, es una alteración considerable. Usar la API directamente no instala nada, no monta unidades y no necesita símbolos: solo lee la memoria física del huésped y la escribe secuencialmente. Es la opción menos intrusiva que produce un artefacto estándar.

9.5 Consolidación con Merge-VHD

La secuencia completa de consolidación tiene siete pasos:

1. **Validación de cmdlets:** `Get-Command Get-VHD` y `Get-Command Merge-VHD` por separado.
2. **Detección de cadena:** si no hay `.avhdx` en `Disks\`, devuelve `NO_CHAIN` y el script informa que no hay nada que consolidar.
3. **Copia a área de trabajo:** todos los `.vhdx`, `.vhd` y `.avhdx` de `Disks\` a `Consolidation\Working\`.
4. **Re-enlace de la cadena:** por cada disco se obtiene su `ParentPath` y se reapunta con `Set-VHD … -IgnoreIdMismatch` hacia la copia local del padre.
5. **Fusión:** `Merge-VHD -Path $activePath -Force -Confirm:$false`.
6. **Extracción del resultado:** el `.vhdx` base resultante se copia a `Consolidation\<VM>_consolidated.vhdx`.
7. **Limpieza:** se elimina `Working\` para no duplicar gigabytes.

> POR QUÉ IMPORTA
>
> Cada VHDX lleva un identificador único, y el disco hijo guarda el del padre que espera encontrar. Al **copiar** el padre a otra ubicación, ese identificador ya no coincide con lo que el hijo tiene registrado, y `Set-VHD` se negaría a re-enlazarlos por seguridad. En este contexto la discrepancia es esperada y benigna: la copia es idéntica al original porque acaba de hacerse, y se trabaja precisamente sobre copias para no tocar la evidencia. `-IgnoreIdMismatch` indica a Hyper-V que proceda.

Reserva: leer el ParentPath del binario

```
$stream = [System.IO.File]::OpenRead($currentPath)
$headerSize = [Math]::Min($stream.Length, 1048576)     // 1 MB de cabecera
$bytes = New-Object byte[] $headerSize
[void]$stream.Read($bytes, 0, $headerSize)
$text = [System.Text.Encoding]::Unicode.GetString($bytes)
$match = [regex]::Match($text, '([A-Z]:\\([^\x00]+)\.vhdx)', 'IgnoreCase')
```

Cuando `Get-VHD` falla, lo habitual si el padre original ya no está accesible desde la nueva ubicación (que es justo el caso que importa), la ruta se extrae directamente de los metadatos del archivo. El VHDX almacena las rutas de padre en UTF-16, así que se decodifica como Unicode y se busca con

expresión regular un patrón de ruta terminado en `.vhdx`. Sin esta reserva la consolidación se bloquearía precisamente en el escenario para el que fue escrita.

El `.bat` monitoriza la salida del `.ps1` buscando tres centinelas, `MERGE_OK`, `MERGE_FAIL` y `NO_CHAIN`, y muestra además cada línea de progreso al operador. Al final valida que el archivo consolidado exista realmente y reporta su tamaño en bytes.

> NOTA
>
> El script cierra el flujo marcando el resultado como evidencia derivada, con dos afirmaciones explícitas: «El disco consolidado es un artefacto derivado para análisis» y «La evidencia original en Disks no fue eliminada». Después ofrece la conversión a RAW con QEMU.

9.6 Clon completo

```
for /f "usebackq delims=" %%P in (`powershell -NoProfile -Command ^
    "(Get-VM -Name '!HVC_VMNAME_PS!').Path"`) do set "HVC_VMFOLDER=%%P"

call :HASH_EVIDENCE FOLDER "!HVC_VMFOLDER!"
"*.vhdx,*.vhd,*.avhdx,*.vmcx,*.vmrs,*.xml,*.bin,*.vsv" ^
     "…\hashes_originales_PREcopia.sha256" "HyperV_Original_PreCopia_ClonCompleto"

robocopy "!HVC_VMFOLDER!" "!HVC_EVIDENCE!\VM" /E /COPY:DAT /R:1 /W:1 /NFL /NDL ^
    & if errorlevel 8 (exit 1) else (exit 0)
```

La propiedad `.Path` de la VM apunta a la carpeta raíz que contiene `Virtual Machines\`, `Virtual Hard Disks\` y `Snapshots\`. El patrón de hashes es el más amplio del framework: ocho extensiones que cubren discos, diferenciales, configuración y estados guardados. Al terminar imprime la instrucción de importación:

```
Import-VM -Path "<destino>\VM\Virtual Machines\<archivo>.vmcx"
```

con la alternativa gráfica («Import Virtual Machine…») y la indicación de restaurar el checkpoint deseado desde el panel de Checkpoints.

CAPÍTULO 10

Módulo 4. Conversión con QEMU

Convierte cualquier disco virtual a RAW plano. Es el único flujo del framework con un control de integridad obligatorio, y produce un reporte autocontenido de la conversión.

10.1 Por qué convertir a RAW

Los formatos de disco virtual (VMDK, VHD, VHDX, VDI) añaden una capa de metadatos, asignación dinámica y a veces compresión sobre los sectores reales. Las herramientas de análisis forense pueden manejar algunos de ellos, pero con limitaciones: versiones concretas, extensiones específicas, o pérdida de funcionalidad. El formato **RAW** (un volcado plano donde el byte N del archivo es el byte N del disco) es el denominador común que todas aceptan sin fricción: Autopsy, Volatility, The Sleuth Kit, FTK y X-Ways.

El módulo es accesible de dos formas: directamente desde el menú de máquinas virtuales (opción 4), o como oferta al final de casi todos los flujos de adquisición de disco virtual, sobre el archivo recién obtenido.

10.2 Selección del origen

Dos vías: ruta completa de un archivo, o búsqueda automática en una carpeta:

```
for %%E in (vmdk vhd vhdx vdi) do (
    for %%F in ("!qemu_scan_dir!\*.%%E") do (
        if exist "%%F" (
            set /a qemu_found+=1
            set "QEMU_FILE_!qemu_found!=%%F"
            echo   !qemu_found!. %%~nxF
        )
    )
)
```

El doble bucle recorre las cuatro extensiones soportadas y numera los resultados. El `if exist` interior evita que un patrón sin coincidencias produzca una entrada literal con comodines.

10.3 Mapeo de formatos

Detección del formato de origen por extensión

```
for %%F in ("!qc_src!") do set "qc_ext=%%~xF"
set "qc_ext=!qc_ext:.=!"

if /I "!qc_ext!"=="vmdk"  set "qc_fmt=vmdk"
if /I "!qc_ext!"=="vhd"   set "qc_fmt=vpc"
if /I "!qc_ext!"=="vhdx"  set "qc_fmt=vhdx"
if /I "!qc_ext!"=="avhdx" set "qc_fmt=vhdx"
if /I "!qc_ext!"=="vdi"   set "qc_fmt=vdi"

if "!qc_fmt!"=="" (
    echo [-] Formato no soportado: .!qc_ext!
    echo [i] Formatos soportados: .vmdk, .vhd, .vhdx, .avhdx, .vdi
    exit /b 1
)
```

POR QUÉ IMPORTA

Hay dos detalles en el mapeo que no son evidentes. El primero: la extensión `.vhd` corresponde al nombre de formato **`vpc`** en QEMU (por «Virtual PC», su origen histórico), no `vhd`. Usar el nombre equivocado hace que qemu-img falle con un error de formato desconocido. El segundo: `.avhdx`, los diferenciales de checkpoint de Hyper-V, se mapea también a `vhdx`, porque estructuralmente lo son. Eso permite convertir un diferencial directamente, algo útil para examinar solo los cambios de un checkpoint.

Por qué se especifica `-f` explícitamente en lugar de dejar que qemu-img autodetecte: la detección automática se basa en la firma del archivo, y en un contexto forense es preferible ser explícito. Si el archivo tiene la extensión equivocada o la firma está dañada, un fallo claro es mejor que una interpretación silenciosa y posiblemente errónea.

10.4 La secuencia de conversión, paso a paso

Paso 1. Información del disco de origen

```
"!QEMU_IMG!" info "!qc_src!"
```

Muestra formato detectado, tamaño virtual, tamaño real en disco, tamaño de cluster y, si existe, la cadena de backing files. Se ejecuta antes de convertir y su salida acaba embebida en el reporte final: documenta el punto de partida.

Paso 2. Hash SHA-256 del origen (obligatorio)

```
for /f "skip=1 tokens=*" %%H in ('certutil -hashfile "!qc_src!" SHA256 2^>nul') do (
    if "!qc_hash_src!"=="" set "qc_hash_src=%%H"
)
call :LOG_CMD "QEMU_Hash_Origen" "certutil -hashfile ""!qc_src!"" SHA256" "0" …
```

> POR QUÉ IMPORTA
>
> Este es el único hash no opcional del framework. En todo el resto del script el hashing es una decisión del operador. Aquí **no se pregunta**: se calcula siempre. La razón es que esta operación transforma el archivo de evidencia a otro formato, y el hash del origen es el único punto de control que permite después demostrar de qué archivo exacto salió la imagen convertida. Sin él, la evidencia derivada quedaría huérfana. El hash del **destino** sí es opcional, como en el resto del framework, porque el destino es un artefacto derivado y su trazabilidad ya está garantizada por el reporte.

Paso 3. La conversión

```
"!QEMU_IMG!" convert -p -f !qc_fmt! -O raw "!qc_src!" "!qc_output!"
```

FLAG	FUNCIÓN
convert	Subcomando de conversión de formato.
-p	Barra de progreso en tiempo real. Sin ella, la conversión de un disco de 200 GB sería una consola muda durante media hora.
-f <fmt>	Formato del **origen**, según el mapeo de la sección anterior.
-O raw	Formato de **salida** (mayúscula O). Siempre `raw`, aunque la extensión pueda ser `.raw` o `.img` a elección del operador.

Se ejecuta a través de `:ACQUIRE`, así que es cancelable con confirmación. Si el archivo de destino ya existe, se pide confirmación antes de sobrescribir y solo entonces se borra el anterior.

NOTA

RAW e IMG son la misma cosa. El menú ofrece `.raw` (recomendado) e `.img`, y la diferencia es **únicamente la extensión**: el contenido es idéntico, porque ambas opciones pasan `-O raw`. La opción existe porque algunas herramientas y flujos de trabajo esperan una u otra por convención.

Pasos 4 a 6. Verificación y reporte

```
"!QEMU_IMG!" info "!qc_output!"                       :: paso 4: verificar el resultado
call :HASH_EVIDENCE FILE "!qc_output!" "" "…\destino_hashes.txt" "QEMU_Destino"   :: paso 5:
opcional

:: paso 6: reporte autocontenido
echo ============================================== > "!qc_report!"
echo   REPORTE DE CONVERSION FORENSE (QEMU-IMG)   >> "!qc_report!"
echo Fecha:       %DATE% %TIME%                    >> "!qc_report!"
echo --- ORIGEN ---                                >> "!qc_report!"
echo Archivo:     !qc_src!                         >> "!qc_report!"
echo Formato:     !qc_fmt! (.!qc_ext!)             >> "!qc_report!"
echo SHA256:      !qc_hash_src!                    >> "!qc_report!"
echo --- DESTINO ---                               >> "!qc_report!"
echo Archivo:     !qc_output!                      >> "!qc_report!"
echo Formato:     raw (.!qc_out_ext!)              >> "!qc_report!"
echo --- INFO QEMU-IMG (DESTINO) ---               >> "!qc_report!"
"!QEMU_IMG!" info "!qc_output!"                     >> "!qc_report!" 2>&1
```

El paso 4 vuelve a interrogar el archivo convertido; el paso 5 ofrece el hash del destino; el paso 6 escribe el reporte y le incrusta la salida de `info` del destino.

El `conversion_report.txt` es un documento autocontenido: quien lo lea sabe qué archivo se convirtió, con qué hash, a qué formato, con qué herramienta, cuándo, y cuáles son las características del resultado. Es la pieza que vincula la evidencia derivada con la primaria en la cadena de custodia.

CAPÍTULO 11

Módulo 5. Docker

Cuatro vías de adquisición sobre un contenedor, con un navegador interactivo de solo lectura y un control de completitud que evita documentar como inexistente algo que sí existe.

11.1 Verificación del motor

```
docker version >nul 2>&1
if !errorlevel! neq 0 goto DK_NO_DOCKER
```

POR QUÉ IMPORTA

`docker --version` solo imprime la versión del cliente: funciona incluso con el daemon detenido. `docker version` (sin guiones) consulta **cliente y servidor**, lo que implica contactar el daemon, y falla si no responde. Con una sola llamada se comprueba lo que realmente importa: que el motor esté operativo. La detección del arranque solo había verificado que el binario existiera en el PATH.

11.2 Enumeración y selección

```
:: Vista para el operador
docker ps -a --format "table {{.ID}}\t{{.Names}}\t{{.Image}}\t{{.Status}}"

:: Alimentación del selector interno
for /f "tokens=1,2 delims=|" %%A in ('docker ps -a --format "{{.ID}}|{{.Names}}" 2^>nul') do (
    set /a dk_count+=1
    set "DK_ID_!dk_count!=%%A"  &  set "DK_NAME_!dk_count!=%%B"
)
```

El flag `-a` incluye contenedores **detenidos**, que son evidencia igual que los activos: un contenedor que se detuvo tras ejecutar algo malicioso sigue teniendo su sistema de archivos intacto. Se usan las plantillas Go de Docker con un separador explícito para el parseo interno, de modo que no haya que depender del ancho de las columnas de la vista de tabla.

El nombre del contenedor se sanea para usarlo como nombre de carpeta, sustituyendo por _ los seis caracteres prohibidos en nombres de archivo de Windows: `/ : \ < > |`.

11.3 Opción 2. Exportar el filesystem

```
docker export "!DK_SEL_ID!" -o "!DK_EVIDENCE!\ContainerExport\!DK_FOLDER!_filesystem.tar"
```

Produce un TAR con el sistema de archivos completo del contenedor: todas sus capas aplanadas en un solo árbol. Se puede extraer y analizar con cualquier herramienta, o recorrer sin extraer.

> ADVERTENCIA
>
> El script imprime la advertencia antes de ejecutar: «`docker export` NO incluye volúmenes montados, solo el filesystem interno». Es una limitación de la propia herramienta, no del script, y tiene consecuencias forenses directas. Los datos que realmente importan (bases de datos, logs de aplicación, archivos subidos) suelen vivir en volúmenes montados desde el host, precisamente para que sobrevivan al contenedor. Adquirir solo el `export` y creer que se tiene todo es un error de completitud. De ahí que exista la opción 3 como flujo separado y explícito.

Al terminar se informa el tamaño real del TAR con `%%~zF`, dato que queda también en el diario: permite detectar después una exportación truncada.

11.4 Opción 3. Volúmenes montados

```
$json = docker inspect "<id>" 2>$null | ConvertFrom-Json
$mounts = $json[0].Mounts
if (-not $mounts -or $mounts.Count -eq 0) { Write-Output 'NO_VOLUMES'; exit }
foreach ($m in $mounts) {
    $name = if ($m.Name) { $m.Name } else { $m.Destination -replace '[/\\:]','_' }
    Write-Output "$($name)|$($m.Source)|$($m.Destination)|$($m.Type)"
}
```

Detección de volúmenes con parseo JSON estructurado.

> POR QUÉ IMPORTA
>
> `docker inspect` devuelve JSON anidado. Extraer `Mounts[].Source` con `findstr` y manipulación de subcadenas sería frágil: cualquier cambio en el formato de salida, un valor con caracteres especiales o una ruta con comillas rompería el parseo. `ConvertFrom-Json` lo interpreta correctamente por construcción, y la salida se reduce a una línea por volumen con cuatro campos separados por `|`, trivial de parsear en lotes.

Los *bind mounts* anónimos no tienen `Name`: para ellos se genera uno a partir del punto de montaje, sustituyendo separadores por _. Así cada volumen tiene un nombre de carpeta válido y distinguible.

El centinela `NO_VOLUMES` distingue dos situaciones que no son lo mismo: «el contenedor no tiene volúmenes», que es informativo, y «no se detectó nada», que sí es un error.

```
robocopy "!dk_v_source!" "!dk_v_out!" /E /COPY:DAT /R:1 /W:1 /NFL /NDL ^
    & if errorlevel 8 (exit 1) else (exit 0)
```

Un robocopy por volumen, cada uno en su propia subcarpeta. Se usa `/COPY:DAT` y no `/COPYALL` porque los volúmenes de Docker en Windows viven bajo rutas gestionadas por el motor, cuyas ACLs no son significativas y cuya copia podría provocar problemas de acceso en el destino. Si un volumen no es accesible desde el host, se avisa y se registra sin abortar los demás.

11.5 Opción 4. Adquisición lógica con navegador

Control de completitud antes de listar

```
for /f "usebackq delims=" %%R in (`docker inspect --format "{{.State.Running}}" "!DK_SEL_ID!"
2^>nul`) do set "_dk_running=%%R"
if not "!_dk_running!"=="true" (
    echo [-] El contenedor "!DK_SEL_NAME!" no esta accesible (detenido o eliminado).
    echo [i] Esto NO significa que la carpeta actual este vacia: se perdio la conexion.
    echo   R    Reintentar
    echo   0    Cancelar y volver
)
```

ADVERTENCIA

Fallo de completitud detectado en auditoría. El problema: `docker exec … 2^>nul` silencia stderr, así que un contenedor detenido produce **exactamente la misma salida vacía** que una carpeta realmente vacía. Para el operador son indistinguibles. El riesgo no está en la corrupción de datos sino en la **completitud del informe**: podría documentar un artefacto como «no existe» cuando en realidad el contenedor se detuvo a mitad de la navegación. Comprobar `State.Running` *antes* de listar convierte una ambigüedad silenciosa en un mensaje explícito con opción de reintentar. El mismo control se implementó en el navegador de ADB.

Listado del directorio: carpetas y archivos separados

```
docker exec "<id>" sh -c "ls -1pL '<ruta>' 2>/dev/null | grep '/$' | sed 's|/$||'"     # carpetas
docker exec "<id>" sh -c "ls -1pL '<ruta>' 2>/dev/null | grep -v '/$'"                # archivos
```

ELEMENTO	FUNCIÓN
-1	Una entrada por línea: imprescindible para parsear con `for /f`.
-p	Añade `/` al final de los directorios: es el marcador que permite clasificarlos.
-L	**Sigue enlaces simbólicos.** Sin este flag, `/bin` apuntando a `/usr/bin` se clasificaría como archivo y el operador no podría entrar en él.
grep '/$'	Filtra las líneas terminadas en `/`: los directorios.
sed 's|/$||'	Quita el `/` final para mostrar el nombre limpio.
grep -v '/$'	Lo contrario: solo lo que no termina en `/`, los archivos.

Todo el navegador es de **solo lectura**: `ls` y `test`, nada más. No se ejecuta código arbitrario dentro del contenedor ni se modifica su estado. La discriminación entre carpeta y archivo al escribir un nombre se hace con `test -d` y luego `test -e`, evaluando el código de salida.

Adquisición, protección contra sobrescritura y empaquetado

```
:: Saneado del nombre de destino
set "dk_safe_name=!dk_cp_src!"
set "dk_safe_name=!dk_safe_name:/=_!"  &  set "dk_safe_name=!dk_safe_name:\=_!"  &  set
"dk_safe_name=!dk_safe_name::=_!"
if "!dk_safe_name:~0,1!"=="_" set "dk_safe_name=!dk_safe_name:~1!"

:: Protección contra sobrescribir una adquisición previa
:DK_DUP_CHECK
if not exist "!dk_acq_dest!" goto DK_DUP_OK
set /a _dk_dup+=1
set "dk_acq_dest=!DK_EVIDENCE!\Artifacts\!dk_safe_name!__!_dk_dup!"
goto DK_DUP_CHECK

:: Adquisición y empaquetado
docker cp "!DK_SEL_ID!:!dk_cp_src!" "!dk_acq_dest!"
"!SEVENZIP!" a -t7z "!dk_zip_dest!" "!dk_acq_dest!"
```

La ruta del contenedor `/var/log/app` se convierte en el nombre de carpeta `var_log_app`, quitando el guion bajo inicial que dejaría la barra raíz. El bucle de duplicados añade `__2`, `__3` y así sucesivamente hasta encontrar un nombre libre: **nunca se sobrescribe una adquisición anterior**, algo crítico cuando se extraen varios artefactos en la misma sesión y dos rutas distintas producen el mismo nombre saneado.

> POR QUÉ IMPORTA
>
> Empaquetar en `.7z` tiene tres razones. Primera: un artefacto puede ser un árbol de miles de archivos pequeños, y un contenedor único simplifica el traslado y el hashing, un archivo y un hash. Segunda: el `.7z` preserva la estructura y los timestamps internos. Tercera: `7zr.exe` es la build reducida de 7-Zip, que **solo soporta el formato `.7z`**, de ahí el flag explícito `-t7z`. Tras un empaquetado exitoso el original sin comprimir se elimina (`rmdir /s /q` para carpetas, `del /f /q` para archivos), de modo que la evidencia queda solo en el contenedor y no duplicada.

Si el empaquetado falla, la evidencia **se conserva sin comprimir** y se informa de la ruta: nunca se borra el original antes de confirmar que el contenedor existe. El hash opcional se calcula sobre el `.7z`, y el operador puede seguir adquiriendo más artefactos sin salir del navegador.

CAPÍTULO 12

Módulo 6. Android por ADB

Gestión de la conexión y adquisición lógica sobre un dispositivo Android, reutilizando el patrón de navegador de Docker adaptado al shell del dispositivo.

12.1 Estado y autorización

```
"!ADB_EXE!" devices -l
```

El flag `-l` añade información descriptiva: modelo, dispositivo y ruta de transporte. El script explica en pantalla la distinción que más confunde en la práctica: `device` significa autorizado y operativo, mientras que `unauthorized` significa que el dispositivo está conectado pero **hay un diálogo pendiente de aceptar en su pantalla**. Sin esa aclaración, un operador podría concluir que falla el cable o el driver.

```
for /f "usebackq skip=1 tokens=1,2" %%A in (`"!ADB_EXE!" devices 2^>nul`) do (
    if "%%B"=="device" ( set /a adb_dev_count+=1 & set "ADB_DEV_!adb_dev_count!=%%A" )
)
```

`skip=1` omite la cabecera «List of devices attached». El filtro `if "%%B"=="device"` deja fuera los estados `unauthorized`, `offline` y `no permissions`: al selector solo entran los dispositivos con los que realmente se puede trabajar. Si hay exactamente uno, se selecciona automáticamente sin preguntar.

12.2 Conexión por red y desconexión

```
"!ADB_EXE!" connect "!adb_ip!"        :: ej. 192.168.1.50:5555
"!ADB_EXE!" disconnect               :: cierra solo sesiones de red
"!ADB_EXE!" kill-server              :: detiene el demonio completo
```

La conexión inalámbrica exige que la depuración por red esté activada previamente en el dispositivo (Ajustes > Opciones de desarrollador); por USB no hace falta este paso, y el script lo aclara. La diferencia entre

`disconnect` y `kill-server` también se explica: el primero cierra solo las sesiones TCP, el segundo detiene el demonio ADB por completo y libera también las conexiones USB. Ambas acciones se registran.

12.3 El navegador del dispositivo

Control de conectividad y listado (enumeración)

```
for /f "usebackq delims=" %%R in (`"!ADB_EXE!" -s !ADB_SERIAL! get-state 2^>nul`) do set
"_adb_state=%%R"
if not "!_adb_state!"=="device" ( … reintentar o cancelar … )

call :NEWGUID _adb_ls_guid
set "ADB_LS_TMP=!TEMP!\forensic_adbls_!_adb_ls_guid!.tmp"
"!ADB_EXE!" -s !ADB_SERIAL! shell ls -1p "!adb_nav_path!" > "!ADB_LS_TMP!" 2>nul

for /f "usebackq delims=" %%N in ("!ADB_LS_TMP!") do (
    set "_adb_n=%%N"
    if "!_adb_n:~-1!"=="/" ( echo     [DIR]  !_adb_n:~0,-1! )
)
```

Mismo control de completitud que en Docker: `get-state` antes de listar, porque un dispositivo desconectado produciría una salida vacía indistinguible de una carpeta vacía. Aquí el listado se vuelca a un **archivo temporal** en lugar de leerse por pipe, y se recorre dos veces (una pasada para las carpetas y otra para los archivos), porque una sola pasada obligaría a acumular dos listas en variables, con los límites de longitud de cmd.

La clasificación es puramente sintáctica: el sufijo `/` que añade `ls -1p` marca los directorios, y `!_adb_n:~0,-1!` lo recorta para mostrar el nombre limpio. Aquí no se usa `-L`, a diferencia de Docker, porque el shell de Android sin root tiene un acceso muy limitado y seguir symlinks a rutas protegidas produciría errores sin aportar navegabilidad.

> ADVERTENCIA
>
> Por qué `:NEWGUID` y no `%RANDOM%`, anotado en el código: `%RANDOM%` tiene un rango de 0 a 32767 y puede **colisionar entre instancias concurrentes** del framework. Dos operadores, o el mismo con dos ventanas abiertas, podrían generar el mismo nombre de archivo temporal y leer el listado del dispositivo del otro. Es el mismo razonamiento que motivó el cambio en el Write Blocker, donde la consecuencia era más grave: mostrar el estado de un disco distinto justo antes de confirmar su bloqueo.

El navegador arranca en `/sdcard/` y no en `/`, porque sin root la visibilidad está limitada a las rutas de almacenamiento externo legibles por el usuario shell, que son además donde vive la mayor parte del contenido de usuario relevante. Los comandos son los mismos que en Docker: `S` adquiere la carpeta actual, `..` sube, `0` cancela, y escribir un nombre entra o adquiere según `test -d` / `test -e`.

El ascenso de directorio se delega a PowerShell, porque manipular rutas con barras normales en lotes es incómodo:

```
powershell -NoProfile -Command "$p='!adb_nav_path!'.TrimEnd('/');
    if($p -eq ''){'/'}else{$p.Substring(0,$p.LastIndexOf('/')+1)}"
```

12.4 Extracción

adb pull con preservación de timestamps (adquisición)

```
"!ADB_EXE!" -s !ADB_SERIAL! pull -a "!adb_pull_src!" "!adb_acq_dest!"
```

> POR QUÉ IMPORTA
>
> El flag `-a` es imprescindible: preserva el **timestamp** y el modo del archivo original. Sin él, todos los archivos extraídos tendrían como fecha de modificación el **momento de la extracción**, lo que destruye por completo la información temporal. En análisis forense de móviles la cronología suele ser la evidencia principal: cuándo se tomó una foto, cuándo se recibió un archivo, cuándo se modificó una base de datos de mensajería. Perder los timestamps equivale a perder el caso.

El resto del flujo es idéntico al de Docker: saneado del nombre, aquí sustituyendo `/` y `:`, bucle de protección contra duplicados con sufijo `__N`, empaquetado en `.7z`, eliminación del original sin comprimir solo si el contenedor se creó, hash opcional, y opción de seguir adquiriendo más artefactos sin salir del navegador.

CAPÍTULO 13

Módulo 7. Write blocker por software

Marca un disco como solo lectura a nivel de sistema operativo. Es el módulo más corto del framework y el que más advertencias imprime, porque su limitación es conceptual antes que técnica.

13.1 Qué es y qué no es

El script imprime siempre las mismas dos advertencias antes de tocar nada, y conviene leerlas literalmente: «Mitigación por software: marca el disco como solo lectura» y «**NO equivale a un write blocker físico. Documentar en CoC**».

> ADVERTENCIA
>
> `Set-Disk -IsReadOnly` establece un atributo que respeta el **sistema operativo**: cualquier software que use las APIs normales de Windows no podrá escribir en el disco. Pero no impide un acceso a bajo nivel que evite esas APIs, no protege durante el arranque antes de que Windows aplique el atributo, y no sirve de nada si el disco se conecta a otro equipo. Un write blocker **físico** interrumpe eléctricamente las líneas de escritura del bus: la protección está en el hardware y no depende de ningún software. Por eso el uso de este módulo **debe consignarse en la cadena de custodia** como lo que es, una mitigación por software, y nunca presentarse como un bloqueador de escritura por hardware.
>
> El uso legítimo, que el script también enuncia, es proteger un disco o un USB de evidencia contra **escrituras accidentales** mientras se trabaja con él en la estación de análisis: montajes automáticos de Windows, indexación, puntos de restauración, archivos `System Volume Information`.

13.2 Disponibilidad y alternativa

```
if not defined PS_AVAILABLE goto WB_NO_PS
powershell -NoProfile -Command "if (Get-Command Set-Disk -ErrorAction SilentlyContinue) { exit 0
} else { exit 1 }"
if errorlevel 1 goto WB_NO_PS

:WB_NO_PS
echo [-] Requiere PowerShell con cmdlets Get-Disk/Set-Disk (Windows 8 o superior).
echo     Alternativa manual: diskpart > select disk N > attributes disk set readonly
```

Doble verificación: que exista PowerShell y que el cmdlet esté disponible, porque `Set-Disk` pertenece al módulo `Storage`, introducido en Windows 8. En Windows 7 no existe, y el script no se limita a rechazar la operación: ofrece **la alternativa exacta con diskpart**, incluida la secuencia de comandos. Es una decisión de diseño consistente con el resto del framework: cuando no puede hacer algo, explica cómo hacerlo a mano.

13.3 Listado y operación

```
powershell -NoProfile -Command "Get-Disk | Sort-Object Number | Format-Table -AutoSize ^
    Number, FriendlyName, @{N='Size(GB)';E={[math]::Round($_.Size/1GB,1)}}, BusType,
@{N='ReadOnly';E={$_.IsReadOnly}}"
```

Cinco columnas elegidas con criterio. `Number` es el identificador que pide el prompt. `FriendlyName` permite reconocer el modelo. El tamaño en GB con un decimal ayuda a distinguir discos parecidos. **`BusType`** distingue un USB de un disco interno, dato clave para no bloquear por error el disco del sistema. Y `ReadOnly` muestra el estado vigente antes de cualquier cambio.

```
powershell -NoProfile -Command "try{Set-Disk -Number !wb_num! -IsReadOnly $true -ErrorAction
Stop; exit 0}
                                catch{Write-Host $_.Exception.Message; exit 1}"
```

El `try/catch` con `-ErrorAction Stop` convierte cualquier error en un código de salida limpio y muestra el mensaje real de la excepción. Si falla, el script apunta la causa más frecuente: «Ejecute el framework como Administrador». Tanto el éxito como el fallo se registran con el valor exacto del atributo (`IsReadOnly=true` / `false`).

13.4 El aislamiento entre instancias concurrentes

Archivo de estado con nombre único (infraestructura)

```
call :NEWGUID _wb_guid
set "_wb_state_file=!TEMP!\forensic_wbstate_!_wb_guid!.tmp"
powershell -NoProfile -Command "$d=Get-Disk -Number !wb_num! -ErrorAction SilentlyContinue;
    if($d){Write-Host ('  Disco !wb_num!: '+$d.FriendlyName+'   ReadOnly actual =
'+$d.IsReadOnly)}
    else{Write-Host '  __NOEXISTE__'}" > "!_wb_state_file!" 2>nul
for /f "usebackq delims=" %%S in ("!_wb_state_file!") do set "wb_state=%%S"
del "!_wb_state_file!" >nul 2>&1
echo !wb_state! | findstr "__NOEXISTE__" >nul 2>&1
if not errorlevel 1 goto WB_BADNUM
```

POR QUÉ IMPORTA

Fallo real detectado en auditoría dinámica. Con un nombre de archivo temporal **fijo**, dos instancias del framework corriendo a la vez (o el mismo operador abriéndolo dos veces) se pisaban el resultado: una instancia mostraba el estado del disco consultado por la **otra**, **justo antes de confirmar el bloqueo o el desbloqueo**. El operador leería «Disco 1: SanDisk USB» y confirmaría, pero el comando se ejecutaría sobre el disco 3. El nombre con GUID elimina la posibilidad por construcción. El centinela `__NOEXISTE__` resuelve además un segundo problema: distinguir «el disco N no existe» de «la consulta no devolvió nada», que con salida vacía serían indistinguibles.

El flujo completo es deliberadamente redundante en confirmaciones: se lista, se pide el número, se **vuelve a consultar y a mostrar** ese disco concreto con su estado actual, se pide la acción, y se pide una confirmación `S/N` final. Para una operación que modifica un dispositivo de evidencia, esa redundancia es la protección.

CAPÍTULO 14

Verificación de integridad

Dos mecanismos complementarios: `HASH_EVIDENCE`, la biblioteca única de hashing que se invoca al final de cada adquisición, y `VERIFY_DC3DD`, que demuestra que una imagen es copia byte-fiel del dispositivo.

14.1 Por qué el hash es opcional

Cabría esperar que un framework forense hasheara siempre y sin preguntar. El script hace lo contrario, y lo hace a propósito. Tres razones lo sostienen.

- **Coste real.** Calcular SHA-256 de una imagen de 2 TB lleva horas y satura el disco. Si el operador necesita empezar la siguiente adquisición, forzarlo bloquea el trabajo de campo.
- **Un solo algoritmo, el elegido.** Calcular MD5, SHA1 y SHA256 a la vez triplica el tiempo. El script calcula *únicamente* el que se pide.
- **La decisión es un dato forense.** Cuando el operador declina, la omisión no queda en silencio: se escribe en el reporte y en los dos logs.

> POR QUÉ IMPORTA
>
> En un procedimiento forense, la ausencia de un hash es un dato tan relevante como su presencia. Registrar la negativa separa dos situaciones que de otro modo se confunden: «no se hasheó porque el examinador lo postergó» y «no se sabe si se hasheó». La primera es una decisión documentada, con etiqueta, equipo y marca de tiempo; la segunda es un hueco en la cadena de custodia.

La única excepción es el hash del origen antes de una conversión QEMU, que sí es obligatorio: esa operación transforma el archivo, y sin ese punto de control la evidencia derivada quedaría huérfana.

14.2 Contrato e interfaz

PARÁMETRO	NOMBRE	VALORES
`%1`	MODE	`FILE` (un archivo) o `FOLDER` (conjunto por patrón)
`%2`	TARGET	Ruta del archivo o de la carpeta
`%3`	WILDCARD	Patrones para FOLDER, separados por coma (por ejemplo `*.vhdx,*.vhd,*.avhdx`); cadena vacía en FILE
`%4`	REPORT	Ruta del archivo de reporte a generar
`%5`	LABEL	Etiqueta de contexto que aparece en el reporte y en los logs (por ejemplo `VMware_Memoria`)

Todos los módulos llaman a esta misma subrutina, cada uno con su propia etiqueta. Esa etiqueta es lo que permite rastrear en el log central qué hash corresponde a qué operación de qué módulo, sin duplicar la lógica de hashing en cada módulo que produce evidencia.

14.3 Motor primario: HashMyFiles

```
start "" /wait "!HMF_EXE!" /enable_hash !_he_hmf! /file   "!_he_target!" /stext "!_he_report!"
start "" /wait "!HMF_EXE!" /enable_hash !_he_hmf! /folder "!_he_target!" ^
      /wildcard "!_he_wild!" /subfolders 1 /stext "!_he_report!"
```

Las dos invocaciones de HashMyFiles dentro de `HASH_EVIDENCE`: modo archivo y modo carpeta.

FLAG	FUNCIÓN
`/enable_hash 1`	MD5
`/enable_hash 2`	SHA1
`/enable_hash 8`	SHA256
`/file`	Un único archivo objetivo
`/folder` + `/wildcard` + `/subfolders 1`	Carpeta con patrones y recursión
`/stext`	Exporta el resultado a texto plano

Los valores 1, 2 y 8 son una **máscara de bits** de HashMyFiles (1=MD5, 2=SHA1, 4=CRC32, 8=SHA256). Pasar un solo valor activa un solo algoritmo, que es exactamente el comportamiento buscado.

> NOTA
>
> HashMyFiles es una aplicación **gráfica**, no de consola. Sin `/wait`, cmd la lanzaría y continuaría de inmediato, y el script pasaría a la siguiente instrucción antes de que el reporte existiera; con `/wait` se espera su terminación. El primer par de comillas vacío es obligatorio: `start` interpreta el primer argumento entrecomillado como el *título* de la ventana, así que sin él la ruta del ejecutable se tomaría como título y nada se ejecutaría.

14.4 Motor de reserva: certutil

```
>>"!_he_report!" echo # Hash !_he_algo! (certutil) - !_he_label!
>>"!_he_report!" echo # Generado UTC: !_he_ts1!

:: Modo archivo
certutil -hashfile "!_he_target!" !_he_algo! >> "!_he_report!" 2>&1

:: Modo carpeta
pushd "!_he_target!" 2>nul
for /r %%F in (!_he_wild!) do certutil -hashfile "%%F" !_he_algo! >> "!_he_report!" 2>&1
popd 2>nul
```

Hashing nativo sin dependencias: la ruta de reserva de `HASH_EVIDENCE` cuando HashMyFiles no está disponible.

> ADVERTENCIA
>
> Peculiaridad de cmd que obliga al `pushd`, anotada en el código: `for /r "!var!"` con el directorio en **expansión retardada no itera**. Es un comportamiento conocido de cmd.exe. La solución es `pushd` al directorio objetivo y usar `for /r` sin ruta explícita, de modo que tome el directorio de trabajo actual, que sí funciona, incluso con listas de patrones separadas por coma en expansión retardada. El `popd` restaura el directorio anterior.

La reserva no es equivalente en formato. `certutil` produce una salida por archivo, con cabecera y hash en líneas separadas, mientras HashMyFiles genera una tabla estructurada. Cumple la función esencial (dejar constancia

verificable del hash) sin ninguna dependencia externa, lo que la hace válida en un sistema mínimo o en un equipo intervenido donde no se quiera desplegar utilidades adicionales.

La cabecera que el script añade al reporte incluye el algoritmo, la etiqueta de contexto y la marca UTC de generación. Sin ella, un archivo con solo la salida cruda de `certutil` no diría a qué operación pertenece.

14.5 Registro de la decisión negativa

```
:HE_SKIP
echo [i] Hash no calculado por decision del operador.
>>"!_he_report!" echo # Hash no calculado por decision del operador. (!_he_label!)
call :LOG_EVENT "Hash no calculado por decision del operador - !_he_label!"
call :LOG_CMD "Hash_!_he_label!" "HASH_NO_CALCULADO_decision_operador" "0" "" "" "!_he_target!"
""
```

La negativa se escribe en **tres** sitios: el propio archivo de reporte, que así existe y explica por qué está vacío; el diario de la evidencia, que viaja con ella; y el registro central de comandos. El `ExitCode 0` es correcto. No hubo error, hubo una decisión.

14.6 Estructura defensiva: subrutinas planas

> ADVERTENCIA
>
> `HASH_EVIDENCE`, `VERIFY_DC3DD`, `VALIDATE_DEST_DIR`, `VALIDATE_SRC_DRIVE` y `LOG_EVENT` están escritas como subrutinas **planas**: usan `goto` a etiquetas internas en lugar de bloques `if (…)`. La razón está documentada en el código. Un argumento que contenga un paréntesis de cierre `)`, frecuente en nombres de máquinas virtuales, IDs de contenedor o rutas, **cierra el bloque prematuramente** al expandirse, y cmd.exe aborta el script completo con «X was unexpected at this time». No es un error recuperable: el framework entero muere. Escribir estas subrutinas sin paréntesis elimina la clase de fallo por construcción.

El mismo razonamiento explica el uso de `setlocal enabledelayedexpansion` al entrar en `HASH_EVIDENCE` y de `endlocal` antes de cada `exit /b`: las variables `_he_*` quedan aisladas y no contaminan el ámbito del llamador, que puede tener en uso variables con nombres parecidos.

14.7 VERIFY_DC3DD: la diferencia entre hashear y verificar

El detalle operativo está en el capítulo 5, sección 5.5. El punto conceptual merece repetirse aquí porque es el argumento central del módulo de integridad: hashear y verificar responden a preguntas distintas.

OPERACIÓN	QUÉ DEMUESTRA	QUÉ NO DEMUESTRA
Hash de la imagen	Que la imagen no ha cambiado desde ese momento	Que la imagen corresponda al dispositivo original
Hash del origen frente al hash de la imagen	Que la imagen es una **réplica exacta** del dispositivo, sector a sector	No aplica

Un hash de la imagen sella el archivo a partir del instante en que se calcula, y nada más: si la copia salió mal, el hash certifica fielmente una copia mala. La comparación entre origen e imagen es la que cierra ese hueco.

Solo dc3dd permite la segunda comparación sin releer el disco completo, porque hashea el stream durante la lectura. Es la razón por la que el script la marca como herramienta recomendada, y por la que `VERIFY_DC3DD` solo se invoca cuando `ADQ_TOOL` es `dc3dd`.

CAPÍTULO 15

Núcleo anti-interrupción: las dos capas de CTRL+C

El componente más técnico de Ámbar. Resuelve un problema con consecuencias forenses directas: un CTRL+C accidental durante una adquisición de tres horas no debe destruir la evidencia ni cerrar el framework.

15.1 El problema completo

Cuando se pulsa CTRL+C en una consola de Windows, el evento se entrega a **todos** los procesos asociados a esa consola. En el escenario del framework eso significa tres víctimas simultáneas:

1. **La herramienta forense** (`dd`, `robocopy`, `ftkimager`) termina en seco y deja una imagen parcial sin ninguna indicación, en el propio archivo, de que está incompleta.
2. **El `cmd.exe` que ejecuta el `.bat`** responde con el prompt «Terminate batch job (Y/N)?». Si el operador contesta Y, lo natural cuando ya ha pulsado CTRL+C, sale del framework por completo y pierde el contexto de la sesión.
3. **El operador** se queda sin saber en qué punto se interrumpió, si el archivo sirve, ni cómo continuar.

Hay además un requisito contradictorio: CTRL+C **debe seguir funcionando**. Una adquisición mal configurada sobre el disco equivocado tiene que poder cancelarse. La solución no es bloquear la señal, sino *mediarla*.

15.2 Capa 1: launch_noctrlc.ps1

Ignorar CTRL+C y heredarlo al cmd hijo

```
public static int Run(string fullCommandLine) {
    // NULL + add=true  =>  este proceso IGNORA CTRL+C; se hereda por el cmd hijo
    SetConsoleCtrlHandler(IntPtr.Zero, true);

    var si = new STARTUPINFO();
    si.cb = Marshal.SizeOf(typeof(STARTUPINFO));
    bool ok = CreateProcessW(
        null, buf,
        IntPtr.Zero, IntPtr.Zero,
        true,   // bInheritHandles: comparte handles de consola (mismo terminal)
        0,      // sin flags: misma consola, mismo grupo, hereda ignore-CTRL+C
        IntPtr.Zero, null,
        ref si, out pi);
    if (!ok) return -1;
    WaitForSingleObject(pi.hProcess, 0xFFFFFFFF);   // INFINITE
    GetExitCodeProcess(pi.hProcess, out code);
    return unchecked((int)code);
}
```

El mecanismo. `SetConsoleCtrlHandler(NULL, TRUE)` es una llamada con semántica especial: en lugar de registrar un handler, activa el atributo «este proceso ignora CTRL+C». Ese atributo **se hereda** por los procesos hijos. El `cmd.exe` que ejecuta el `.bat` se lanza desde aquí y por tanto también lo ignora, con lo que desaparece el prompt «Terminate batch job».

`dwCreationFlags = 0` es intencional: el hijo va en la **misma consola y el mismo grupo de procesos**, de modo que comparte la ventana y la experiencia es transparente para el operador. Con `bInheritHandles = true` hereda los handles de entrada, salida y error.

> POR QUÉ IMPORTA
>
> **El contrato del código 99.** La función devuelve el código de salida real del `cmd` hijo, salvo un valor reservado: **99** significa que el propio lanzador falló (`Add-Type` no compiló, `CreateProcess` devolvió error). El `.bat` comprueba exactamente ese valor y, si lo recibe, continúa en modo degradado sin aislamiento en lugar de quedarse sin interfaz. Es el principio de degradación elegante aplicado a un componente crítico: la ausencia de la mejora no impide el funcionamiento.

`STARTUPINFO` se declara con **alineamiento natural**, sin `Pack` ni padding manual: el marshaller de .NET inserta el relleno correcto según el tamaño de `IntPtr` (4 bytes en x86, 8 en x64). En x64 la estructura mide 104 bytes y en x86, 68; `Marshal.SizeOf` calcula el valor correcto en tiempo de ejecución en ambos casos. Escribir el padding a mano habría producido un binario que solo funciona en una arquitectura.

15.3 Capa 2: forensic_wrapper.ps1

Registro del handler y limpieza del flag heredado

```
public static void RegisterHandler() {
    // CRITICO: este proceso HEREDA el flag "ignorar CTRL+C" del lanzador.
    // Con ese flag activo Windows NUNCA invoca los handlers registrados
    // -> CTRL+C parecería muerto durante la adquisición.
    // Registrar un handler NO limpia el flag: hay que limpiarlo explícito.
    SetConsoleCtrlFlag(IntPtr.Zero, false);
    _ctrlHandler = new CtrlHandlerDelegate(OnCtrl);
    SetConsoleCtrlHandler(_ctrlHandler, true);
}

public static void UnregisterHandler() {
    SetConsoleCtrlHandler(_ctrlHandler, false);
    // Restaurar "ignorar CTRL+C" para el resto de vida del proceso: evita
    // que un CTRL+C tardío mate al wrapper durante la limpieza final.
    SetConsoleCtrlFlag(IntPtr.Zero, true);
}
```

ADVERTENCIA

La interacción entre las dos capas. Aquí está el detalle más sutil de todo el framework. El wrapper es *nieto* del lanzador, así que hereda el flag «ignorar CTRL+C». Y ese flag es un atributo **independiente** de la lista de handlers: mientras está activo, Windows *no invoca ningún handler registrado*. Registrar el handler propio no basta, porque el flag heredado lo neutraliza y CTRL+C parecería no hacer nada durante la adquisición. Hay que limpiarlo explícitamente con `SetConsoleCtrlHandler(NULL, FALSE)`. Al terminar se vuelve a activar, para que una pulsación tardía no mate al wrapper mientras cierra handles y escribe el resultado. Se declaran dos sobrecargas del mismo import de kernel32, una con delegado y otra con `IntPtr`, porque la API usa el mismo punto de entrada para las dos semánticas.

El handler propio consume la señal y no la propaga:

```
private static bool OnCtrl(uint ctrlType) {
    if (ctrlType == 0 || ctrlType == 1) {   // CTRL_C_EVENT o CTRL_BREAK_EVENT
        Interrupted = true;
        return true;   // consumido - no pasa al handler por defecto
    }
    return false;
}
```

Solo levanta una bandera `volatile`. La lógica pesada no puede ejecutarse dentro de un handler de consola, que corre en un hilo aparte y con restricciones, así que el bucle principal es quien la consulta y actúa.

Lanzamiento en grupo de proceso aislado

```
const uint CREATE_NEW_PROCESS_GROUP = 0x00000200;

bool ok = CreateProcessW(
    null, buf,
    IntPtr.Zero, IntPtr.Zero,
    true,                          // bInheritHandles - el hijo escribe en nuestra consola
    CREATE_NEW_PROCESS_GROUP,      // aislado del CTRL+C del teclado
    IntPtr.Zero, null,
    ref si, out pi);

// La línea que se ejecuta realmente:
$hProcess = [ForensicCtrl]::StartIsolated("cmd.exe /s /c `"$cmdLine`"", [ref]$procPid)
```

`CREATE_NEW_PROCESS_GROUP` hace que el proceso hijo pertenezca a un grupo distinto: el CTRL+C del teclado, que se dirige al grupo de la consola, **no le llega**. Solo lo recibe el wrapper, que decide qué hacer. Al mismo tiempo `bInheritHandles = true` permite que la herramienta siga escribiendo su progreso en la misma consola, que es lo que el operador necesita ver.

El `/s` de `cmd.exe /s /c` modifica el tratamiento de las comillas: con `/s`, cmd toma todo lo que hay entre la primera y la última comilla como el comando, sin intentar reinterpretar las comillas internas. Es imprescindible porque los comandos del framework contienen múltiples rutas entrecomilladas.

15.4 El árbol de procesos: por qué no basta con el hijo

Toolhelp32 y recorrido BFS

```
// BFS: raíz primero, luego hijos, nietos, etc.
static List<uint> GetProcessTree(uint rootPid) {
    List<uint> result = new List<uint>();
    result.Add(rootPid);
    IntPtr snap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    // … recopilar todos los pares (pid, parentPid) del sistema …
    int i = 0;
    while (i < result.Count) {
        uint parent = result[i]; i++;
        foreach (uint[] p in pairs) {
            if (p[1] == parent && p[0] != parent && !result.Contains(p[0]))
                result.Add(p[0]);
        }
    }
    return result;
}

public static void SuspendTree(int rootPid) { /* NtSuspendProcess sobre cada pid */ }
public static void ResumeTree (int rootPid) { /* NtResumeProcess  sobre cada pid */ }
public static void KillTree   (int rootPid) { /* TerminateProcess sobre cada pid */ }
```

ADVERTENCIA

El problema del nieto. El wrapper lanza `cmd.exe /s /c <herramienta>`, así que el handle que guarda es el de **cmd**, mientras la herramienta real (`dd`, `robocopy`, `adb`) es un **nieto**. Suspender o matar solo a cmd deja la herramienta viva: seguiría escribiendo su progreso *encima del diálogo de confirmación*, haciéndolo ilegible, y la adquisición continuaría aunque el operador eligiera abortar. Por eso las tres operaciones se aplican a todos los descendientes, obtenidos con `CreateToolhelp32Snapshot` y un recorrido en anchura sobre los pares (pid, parentPid) del sistema.

POR QUÉ IMPORTA

El orden al matar. `KillTree` mata **la raíz primero** y después los descendientes. El motivo es concreto: si el comando era una cadena `a & b`, como los `robocopy … & if errorlevel 8 (exit 1)` del framework, matar primero la herramienta dejaría a cmd libre para *lanzar el siguiente comando de la cadena*. Matando cmd antes se corta esa posibilidad. Y `TerminateProcess` funciona sobre procesos suspendidos, así que el orden suspender, preguntar, matar es seguro.

15.5 El bucle de monitorización y el diálogo

Polling, suspensión y decisión

```
:monitorLoop while ($true) {
    $waitResult = [ForensicCtrl]::WaitForSingleObject($hProcess, 300)

    if ($waitResult -eq 0) {                    # WAIT_OBJECT_0: el proceso terminó
        [ForensicCtrl]::GetExitCodeProcess($hProcess, [ref]$rawCode) | Out-Null
        if ($rawCode -eq [uint32]0) { $exitCode = 0 } else { $exitCode = 1 }
        break monitorLoop
    }

    if ([ForensicCtrl]::Interrupted) {
        [ForensicCtrl]::ResetInterrupt()
        [ForensicCtrl]::SuspendTree($procPid)     # congelar TODO el árbol

        # … advertencia y diálogo …
        $decision = $Host.UI.PromptForChoice("INTERRUPCION FORENSE",
            "Desea continuar la adquisicion o abortarla?", $choices, 0)   # 0 = Continuar por
defecto

        if ($decision -eq 1) {
            [ForensicCtrl]::KillTree($procPid)
            [ForensicCtrl]::WaitForSingleObject($hProcess, 5000) | Out-Null
            $aborted = $true
            break monitorLoop
        } else {
            [ForensicCtrl]::ResumeTree($procPid)  # continuar donde estaba
        }
    }
}
```

El *polling* de 300 ms es un compromiso: lo bastante frecuente para que la respuesta a CTRL+C se perciba inmediata, lo bastante espaciado para no consumir CPU durante horas. La misma llamada detecta también la terminación normal del proceso, así que un solo bucle cubre los dos caminos.

> ADVERTENCIA
>
> **Códigos de salida que desbordan Int32.** `$rawCode` se declara `uint32` a propósito: herramientas como WinPmem devuelven `0xFFFFFFFF` (4.294.967.295), que **desborda un `Int32`** y provocaría una excepción de conversión. Como el framework solo necesita distinguir éxito de error, se compara contra `[uint32]0` y todo lo demás se normaliza a 1.

El texto del diálogo es parte del diseño forense, no decoración:

```
[!] SOLICITUD DE INTERRUPCION DETECTADA (CTRL+C / BREAK)
    Herramienta: <descripción>
    ADVERTENCIA: Abortar ahora dejara la evidencia INCOMPLETA.
    Una imagen parcial puede ser inadmisible en procedimiento judicial.

  [C] Continuar  - Ignorar la interrupcion y continuar (RECOMENDADO para DFIR)
  [A] Abortar    - Terminar el proceso ahora (la evidencia quedara INCOMPLETA)
```

Texto literal que ve el operador cuando el árbol de procesos ya está congelado.

Se nombra la herramienta concreta, porque el operador puede haber lanzado varias cosas, se enuncia la consecuencia real y se marca **Continuar como opción por defecto** (el último parámetro `0` de `PromptForChoice`). Así, pulsar Enter por inercia *no* destruye la adquisición. Abortar exige una elección deliberada.

Las opciones se construyen con `New-Object` y no con `::new()` por compatibilidad con PowerShell 2.0, 3.0 y 4.0 en Windows 7 y 8, el mismo criterio que atraviesa los tres módulos PowerShell del proyecto.

15.6 Lectura del comando: la codificación OEM

Por qué no se lee como UTF-8

```
$oemEnc  = [System.Text.Encoding]::GetEncoding(
               [System.Globalization.CultureInfo]::CurrentCulture.TextInfo.OEMCodePage)
$cmdLine = ([System.IO.File]::ReadAllLines($CmdFile, $oemEnc))[0].Trim()
Remove-Item -LiteralPath $CmdFile -Force -ErrorAction SilentlyContinue
```

ADVERTENCIA

Corrupción de rutas con acentos. El archivo de comando lo escribe cmd.exe con `echo >`, en la codificación **OEM** del sistema (CP850, CP437 u otra según la región), **no** en UTF-8. Leerlo como UTF-8 corrompe acentos y eñes: una ruta como `C:\Máquinas virtuales\Servidor Ñ` llegaría transformada y la herramienta recibiría una ruta inexistente, fallando con un error incomprensible. La solución es leer con la página de códigos OEM real de la cultura actual, obtenida en tiempo de ejecución. `ReadAllLines` se usa en lugar de `Get-Content -Encoding` por compatibilidad con PowerShell 2.0.

El temporal se elimina **inmediatamente** después de leerlo, porque contiene rutas de evidencia que no deben quedar en `%TEMP%`. Si el archivo no existe o está vacío, el wrapper devuelve 1 con un mensaje explícito en lugar de intentar ejecutar una cadena vacía.

15.7 La subrutina :ACQUIRE, el puente desde el .bat

Contrato completo de :ACQUIRE

```
:ACQUIRE
if not exist "!WRAPPER_PS1!" ( … error, borrar temporal, exit /b 1 … )

:: Leer el comando ANTES de que el wrapper lo elimine (necesario para LOG_CMD)
for /f "usebackq delims=" %%L in ("%~2") do if "!_acq_cmd_str!"=="" set "_acq_cmd_str=%%L"

call :LOG_CENTRAL "INFO" "acquisition_start" "%~1"
call :UTCNOW _ACQ_TS_START
powershell -NoProfile -ExecutionPolicy Bypass -File "!WRAPPER_PS1!" -CmdFile "%~2" -Description
"%~1"
set "ACQ_EL=!errorlevel!"
call :UTCNOW _ACQ_TS_END
del "%~2" >nul 2>&1

choice /c YN /n /t 0 /d Y >nul 2>&1     :: purga del buffer de teclado

if "!ACQ_EL!"=="2" ( … WARN aborted, avisar de evidencia incompleta … )
else if "!ACQ_EL!"=="1" ( … ERROR … )
else if "!ACQ_EL!"=="0" ( … INFO success … )

call :LOG_CMD "%~1" @ACQ_CMD_STR "!ACQ_EL!" "!_ACQ_TS_START!" "!_ACQ_TS_END!" "!DEST!" ""
exit /b !ACQ_EL!
```

El patrón de uso desde cada módulo es siempre el mismo, y se repite en todos los puntos de adquisición del framework:

```
call :NEWGUID ACQ_GUID
set "ACQ_CMD=!TEMP!\forensic_cmd_!ACQ_GUID!.tmp"
>"!ACQ_CMD!" echo <comando completo con sus rutas entre comillas>
call :ACQUIRE "<Descripción para el log y el diálogo>" "!ACQ_CMD!"
set "_el=!errorlevel!"
```

POR QUÉ IMPORTA

Por qué el comando va por archivo y no por argumento. Los comandos contienen múltiples rutas entrecomilladas, y pasarlos como argumento de línea de comandos a PowerShell obligaría a un escapado en varias capas (cmd, PowerShell, y de nuevo cmd dentro del wrapper) extremadamente frágil. Escribirlo en un archivo con `echo` y pasar solo la *ruta* del archivo elimina el problema por completo. El nombre GUID evita colisiones entre instancias concurrentes.

ADVERTENCIA

El centinela @ACQ_CMD_STR. Anotado en el código: el comando **no** se pasa como argumento a `:LOG_CMD` porque sus comillas internas desbalancean la tokenización de argumentos. `%~2` recibiría fragmentos con comillas sueltas y los `if` de `:LOG_CMD` abortarían *todo* el script con «was unexpected at this time». Se pasa el literal `@ACQ_CMD_STR` como centinela y el valor real por variable; `:LOG_CMD` detecta el centinela y sustituye. Nótese también que el comando se lee del archivo **antes** de invocar al wrapper, porque el wrapper lo borra tras leerlo.

NOTA

La purga del buffer con choice. Tras un proceso con mucha salida en consola, robocopy sin `/NP` por ejemplo, puede quedar una entrada «fantasma» en el buffer de entrada, y el siguiente `set /p` del llamador leería una línea vacía **sin que el operador haya escrito nada**. Es un comportamiento conocido de cmd.exe. `choice` limpia el buffer de entrada al iniciar, y `/t 0 /d Y` hace que retorne de inmediato con el valor por defecto sin bloquear. El resultado se descarta: solo interesa el efecto secundario.

15.8 El contrato de salida universal

CÓDIGO	SIGNIFICADO	REACCIÓN DEL LLAMADOR
0	La herramienta terminó correctamente	Continúa: verificación de integridad, preservación de configuración, oferta de conversión
1	La herramienta devolvió error	Informa, registra y vuelve al menú del módulo (en discos, al menú de herramientas para reintentar con otra)
2	El operador confirmó abortar	Imprime `[ABORT]`, registra que la evidencia **puede estar incompleta**, y vuelve al menú correspondiente

El **2** es la señal que recorre todo el framework: aparece en cada punto de comprobación, siempre con su mensaje de contexto. El script nunca deja caer al operador fuera del programa tras un aborto, y **siempre** deja constancia de que la imagen resultante puede no estar completa, porque documentarlo es la diferencia entre una evidencia parcial identificada como tal y una evidencia parcial presentada como completa.

CAPÍTULO 16

Sistema de registro y trazabilidad

Dos registros complementarios en texto plano, marcas de tiempo con degradación etiquetada, y varias defensas contra las trampas de escribir logs con contenido arbitrario en cmd.exe.

16.1 Los dos destinos

	FORENSIC_COMMANDS.LOG	FORENSIC_JOURNAL.LOG
Ubicación	`Script\Logs\`, uno por instalación	`<DEST>\`, uno por carpeta de evidencia
Escrito por	`:LOG_CMD`	`:LOG_EVENT`
Contenido	Cada comando externo ejecutado, íntegro, con su contexto técnico completo	Diario narrativo de lo que le ocurrió a esa evidencia
Alcance	Toda la sesión, todos los módulos, todas las evidencias	Solo esa evidencia; **viaja con ella**
Visible desde	Opción 8 del menú principal	Se lee directamente en la carpeta

POR QUÉ IMPORTA

Los dos logs responden a preguntas distintas. El central responde «¿qué se ejecutó exactamente en este equipo, cuándo y con qué resultado?», que es la reproducibilidad técnica del proceso. El diario responde «¿qué le pasó a *esta* pieza de evidencia?», y por vivir en su misma carpeta acompaña a la evidencia si se copia a otro medio o se entrega a un tercero. Si solo existiera el central, una evidencia trasladada perdería su historia.

16.2 Estructura de una entrada de comando

```
[2026-07-24T21:35:00Z]
Modulo    : dc3dd SHA256 - /dev/sdb
Host      : LAB-FORENSE-01
Usuario   : examinador
OS        : Windows 11 (x64)
Directorio: C:\Users\…\Script\Windows
Comando   :
  "C:\…\dc3dd.exe" "if=/dev/sdb" "of=/cygdrive/d/Casos/img.dd" "hash=sha256"
"log=/cygdrive/d/Casos/img.log"
ExitCode  : 0
Inicio    : 2026-07-24T21:35:00Z
Fin       : 2026-07-24T23:12:41Z
Duracion  : 01:37:41
Estado    : Exito
Salida    : D:\Casos\Disco2_20260724
------------------------------------------------------------------
```

Una entrada completa de `:LOG_CMD` en el log central de comandos.

El **comando completo** se registra íntegro, no resumido: es lo que permite reproducir la adquisición o auditar exactamente qué se hizo, con las rutas y los parámetros que se usaron de verdad. El campo `Estado` traduce el código de salida a lenguaje natural.

```
set "_lc_status=Exito"
if "!_lc_exit!"=="2" set "_lc_status=Cancelado"
if not "!_lc_exit!"=="0" if not "!_lc_exit!"=="2" set "_lc_status=Error"
```

Los campos `Salida` y `HashFile` solo se escriben si tienen valor, para no ensuciar el log con líneas vacías. El bloque completo se genera con un `(…) >> archivo`, una sola redirección para toda la entrada.

16.3 El diario de la evidencia

```
:LOG_EVENT
set "_le_msg=%~1"
if "!DEST!"=="" exit /b
if not exist "!DEST!" exit /b
call :UTCNOW LGT
>>"!DEST!\Forensic_Journal.log" echo [!LGT!] [!COMPUTERNAME! - !USERNAME!] !_le_msg! 2>nul
```

`:LOG_EVENT` completa: dos guardas, marca de tiempo y una única línea de escritura.

> ADVERTENCIA
>
> Por qué flujo lineal y no un bloque `if`, documentado en el código: `%~1` puede contener paréntesis, por ejemplo un ID de contenedor Docker o un mensaje como «abortado por el usuario (CTRL+C)». Expandir ese texto **dentro** de un bloque `if (…)` hace que el `)` del mensaje cierre el bloque prematuramente y cmd aborte todo el script con «X was unexpected at this time». El flujo lineal con `exit /b` temprano elimina el riesgo.

Las dos guardas iniciales son deliberadas: si `DEST` no está definida o la carpeta no existe, la subrutina **no hace nada y sale limpiamente**. Nunca crea carpetas por su cuenta ni falla por escribir en una ruta inexistente. Eso permite llamarla desde cualquier punto del flujo, incluso antes de que el destino esté configurado, sin condicionar cada llamada.

Cada línea lleva la marca de tiempo, el equipo y el usuario. En un laboratorio con varios examinadores, el diario identifica quién hizo qué.

16.4 Marcas de tiempo con degradación etiquetada

```
if defined PS_AVAILABLE (
    if /i "%~2"=="COMPACT" (
        powershell -Command "[DateTime]::UtcNow.ToString('yyyyMMdd_HHmmssZ')"
    ) else (
        powershell -Command "[DateTime]::UtcNow.ToString('yyyy-MM-ddTHH:mm:ssZ')"
    )
)
if not defined _ts if defined HAS_WMIC (
    wmic os get LocalDateTime          :: hora LOCAL, se marca con sufijo L
    set "_ts=!_Y!-!_Mo!-!_D!T!_H!:!_Mi!:!_S!L"
)
if not defined _ts (
    set "_ts=!DATE!_!TIME!"            :: último recurso
    set "_ts=!_ts: =!"
)
```

Los tres niveles de `:UTCNOW`, en orden de preferencia.

POR QUÉ IMPORTA

Hay tres niveles de degradación, y lo decisivo es que **el resultado declara qué es**: la `Z` final indica UTC (norma ISO-8601) y la `L` indica hora local. En un informe forense, una marca de tiempo sin zona horaria conocida es casi inservible: no se puede correlacionar con logs de otros sistemas ni establecer una cronología fiable. Al etiquetar la degradación, un timestamp con `L` avisa al analista de que debe averiguar la zona del equipo antes de correlacionar. El modo `COMPACT` existe porque los dos puntos de ISO-8601 **no son válidos en nombres de archivo** de Windows, y esas marcas se usan para nombrar imágenes y checkpoints.

```
set "_ts_t=!_ts:~11,8!"            :: extrae HH:MM:SS de un ISO en posición 11
if "!_ts_t!"=="" set "_ts_t=!_ts:~0,8!"   :: o del inicio si es formato %TIME%

for /f "tokens=1-3 delims=:" %%A in ("!_ts_t!") do (
    set /a "_sh=1%%A-100"          :: el truco del prefijo 1 …
    set /a "_sm=1%%B-100"
    set /a "_ss=1%%C-100"
)
set /a "_diff=_esec-_tsec"
if !_diff! lss 0 set /a "_diff+=86400"   :: cruce de medianoche
```

`TIMEDIFF`, la subrutina que calcula el campo `Duracion` del log central.

NOTA

Dos trucos de aritmética en lotes. **El prefijo 1:** `set /a` interpreta los números con cero inicial como **octales**, así que `08` y `09` producirían un error de sintaxis («operando no válido»). Prefijando un `1` y restando 100 se fuerza la interpretación decimal: `108-100 = 8`. Es el idioma estándar para manejar horas en lotes. **El ajuste de medianoche:** si la diferencia sale negativa, la operación cruzó las 00:00, y sumar 86.400 segundos (un día) da el resultado correcto. La función tolera además entradas no válidas devolviendo `N/D` en lugar de propagar un error, comprobando primero con `findstr ":"` que haya algo parseable.

16.5 LOG_CENTRAL: un no-op deliberado

```
:LOG_CENTRAL
:: No-op: el log JSONL encadenado fue eliminado. Se conserva como destino valido
:: para las llamadas existentes; no genera ningun archivo.
exit /b
```

NOTA

Una decisión de mantenimiento honesta. Versiones anteriores del framework escribían un log JSONL con encadenamiento de hashes entre entradas. Se retiró, y el capítulo 1 lo enmarca en el principio de «texto plano, sin formatos propietarios», pero las llamadas ya escritas se conservan repartidas por todo el script, con etiquetas semánticas útiles (`PRE_ACTION`, `POST_ACTION`, `acquisition_start`, `acquisition_aborted`, `system_caps`, `ram_dump_empty`, `dc3dd_index_overflow` y otras). Dejar la subrutina como no-op en lugar de borrar las llamadas tiene dos ventajas: no se toca código ya probado, y los puntos de instrumentación quedan marcados por si se decide reactivar el registro estructurado. La alternativa, eliminar cada llamada, habría introducido riesgo de regresión a cambio de nada.

CAPÍTULO 17

Utilidades, validaciones y peculiaridades de cmd.exe

Las subrutinas compartidas por todos los módulos, y un catálogo de los comportamientos no evidentes de cmd.exe que el código documenta y sortea.

17.1 Validación de rutas

```
set "_vd_in=%~1"
set "_VD_OK=0"
if not defined _vd_in goto VDD_BADFMT
if "!_vd_in:~0,2!"=="\\" goto VDD_OK              :: UNC \\servidor\recurso
if not "!_vd_in:~1,1!"==":" goto VDD_BADFMT       :: 2º carácter debe ser :
if not "!_vd_in:~2,1!"=="\" goto VDD_BADFMT       :: 3º debe ser \
set "_vd_drv=!_vd_in:~0,1!"
if not exist "!_vd_drv!:\" goto VDD_NODRV         :: la unidad debe existir
:VDD_OK
set "_VD_OK=1"
```

`VALIDATE_DEST_DIR`: comprobación de forma de la ruta de destino antes de tocar el disco.

> **POR QUÉ IMPORTA**
>
> Si el operador escribe `3` por error, confundiendo el prompt de ruta con un menú, o escribe `salida`, cmd crearía una carpeta con ese nombre **en el directorio de trabajo actual**, que en este framework puede ser `system32` o la carpeta del script. La evidencia acabaría en un lugar arbitrario, difícil de encontrar y peor de documentar. Exigir ruta absoluta (unidad con `X:\` o UNC con `\\`) y comprobar que la unidad existe elimina las dos formas del problema. Los mensajes de error son específicos: distinguen «unidad inexistente» de «formato inválido», e incluyen ejemplos válidos.

Es una subrutina **plana**, por el mismo motivo que `HASH_EVIDENCE`: los mensajes de error contienen paréntesis y un bloque `if (…)` se cerraría prematuramente.

`VALIDATE_SRC_DRIVE` es su equivalente para el origen: acepta `C`, `C:` o `C:\`, toma el primer carácter y comprueba que la raíz exista. Se usa en el volcado de memoria, la adquisición de particiones y el origen de KAPE.

17.2 Identificadores únicos

```
if defined PS_AVAILABLE (
    powershell -NoProfile -Command "[guid]::NewGuid().ToString('N')"
)
if not defined _g (
    set "_t=!TIME::=!"  &  set "_t=!_t: =!"  &  set "_t=!_t:,=!"  &  set "_t=!_t:.=!"
    set "_g=!RANDOM!!RANDOM!!_t!"
)
endlocal & set "%~1=%_g%"
```

`NEWGUID`: GUID de PowerShell con reserva nativa cuando PowerShell no está disponible.

El formato `'N'` produce el GUID sin guiones, apto para nombres de archivo. Sin PowerShell, la reserva combina **dos** llamadas a `%RANDOM%` más la hora con todos sus separadores eliminados. No es criptográficamente único, pero la probabilidad de colisión práctica es despreciable.

Usos en el framework: archivos de comando para el wrapper, scripts `.ps1` temporales, archivo de estado del Write Blocker, listado de `adb shell ls`, listas de VMs en ejecución de VMware y VirtualBox, nombre del clon temporal de snapshot de VirtualBox y su carpeta de trabajo. En todos los casos el objetivo es el mismo: que dos instancias concurrentes del framework no se pisen.

17.3 Catálogo de peculiaridades de cmd.exe documentadas en el código

El script anota, con la indicación de haber sido verificadas por reproducción aislada, los comportamientos no evidentes del intérprete con los que se topó el desarrollo. Se recogen aquí juntos porque explican decisiones de estilo que de otro modo parecerían arbitrarias.

COMPORTAMIENTO	CONSECUENCIA	SOLUCIÓN ADOPTADA
Un `)` en un valor expandido dentro de un bloque `if (…)`	Cierra el bloque prematuramente; cmd aborta **todo** el script con «X was unexpected at this time»	Subrutinas planas con `goto` y `exit /b` en lugar de bloques
Comillas de una ruta con espacios más pipe dentro de un `for /f`	Se pierde el pareo de comillas y cmd intenta ejecutar `C:\Proyecto` como comando	Envoltorio adicional de comillas dobles: `'"" … | … ""'`
Ancla `$` de `findstr /R` con entrada por pipe desde `echo`	No coincide nunca	Omitir el ancla; el prefijo del patrón ya es inequívoco
`for /r "!var!"` con directorio en expansión retardada	No itera en absoluto	`pushd` al directorio y `for /r` sin ruta explícita
`!var:,= !` dentro de un `for (…)`	El `for` tokeniza antes de que la sustitución retardada se resuelva	Sustitución previa en variable aparte con expansión inmediata `%var:,= %`
`set /a` con enteros de 32 bits	Desborda con tamaños de disco mayores de 2 GB: comparaciones erróneas	Delegar la comparación a PowerShell con `[long]`
Números con cero inicial en `set /a`	Se interpretan como octales: `08` y `09` son errores de sintaxis	Truco del prefijo: `1%%A-100`
`errorlevel` de una redirección fallida	No es fiable entre versiones de cmd	Comprobar la **existencia** del archivo escrito
Entrada «fantasma» en el buffer tras un proceso con mucha salida	El siguiente `set /p` lee una línea vacía sin intervención del operador	`choice /c YN /n /t 0 /d Y` para purgar el buffer
Etiquetas con bucle de reintento anidadas dentro de paréntesis	Fuente conocida de fallos de parseo	Mantener esos bloques de selección a nivel superior, fuera de cualquier `if (…)`

NOTA

Cada una de estas notas está escrita en el código junto al fragmento que la aplica, con el síntoma y la causa. Para un proyecto de tesis eso tiene un valor añadido: convierte el script en un documento de ingeniería, y no solo en una herramienta. Un revisor puede verificar que la decisión de estilo no es arbitraria, y quien mantenga el código después no revertirá una solución sin entender qué protegía.

17.4 Otras utilidades

SUBRUTINA	QUÉ HACE
`:BOOT_ROW`	Imprime una fila de estado del arranque con icono, etiqueta rellenada a 42 caracteres y detalle. Usa `setlocal` para aislar sus variables y **no reposiciona el cursor**, por lo que se comporta igual en cmd.exe clásico y en terminales modernos.
`:VBOX_LOAD_RUNNING`	Vuelca los UUID de las VMs encendidas de VirtualBox a un temporal con nombre GUID, para poder etiquetar el estado de cada VM en un listado sin invocar `VBoxManage` una vez por máquina.
`:VMDK_PRINT_CHILDREN`	Imprime recursivamente el árbol de snapshots de VMware con indentación por nivel. Usa `setlocal` en cada nivel de recursión para que las variables de un nivel no pisen las del anterior.
`:VMWARE_CLASSIFY_VMDK`	Clasifica un descriptor como base o snapshot leyendo `CID`, `parentCID` y `parentFileNameHint`. Devuelve tres valores con la técnica `endlocal & set var=%local%`.
`:VMWARE_FIND_SNAPSHOT_MEMORY`	Dos pasadas sobre el `.vmsd` para vincular un `.vmdk` de snapshot con su archivo de memoria y su nombre visible.
`:HVM_RESTORE_CKTYPE`	Restaura el `CheckpointType` original de una VM de Hyper-V. Se invoca en todas las rutas de salida del método de checkpoint.

CAPÍTULO 18

Índice de comandos y glosario de flags

Referencia rápida: todos los comandos externos que el framework puede ejecutar, agrupados por herramienta, y el significado de cada flag empleado.

18.1 Índice de comandos por herramienta

Cada línea es el comando tal como llega al archivo temporal que lee `forensic_wrapper.ps1`, con los placeholders entre ángulos en lugar de las rutas reales.

HERRAMIENTA	COMANDO TAL COMO SE EJECUTA	MÓDULO
DumpIt	`DumpIt.exe /Q /O <salida> /TYPE RAW`	RAM
WinPmem	`winpmem_mini_x64_rc2.exe <salida>` · `winpmem_mini_x86.exe <salida>`	RAM
KAPE	`kape.exe --tsource <u> --target <t> --tdest <d> --vhdx|--zip <n> --quiet`	Triaje
dc3dd	`dc3dd if=<dev> of=<img>.dd hash=sha256 log=<img>.log`	Discos
dd	`dd if=<origen> of=<img>.dd bs=512k --progress`	Discos
ewfacquire	`ewfacquire -u -c none [-S 80000G] [-f encase7|smart] -t <d> <origen>`	Discos
FTK Imager	`ftkimager <origen> <salida> [--e01|--s01] --case-number … --evidence-number … --examiner … [--frag 2000M] [--compress 6]`	Discos
certutil	`certutil -hashfile <archivo> MD5|SHA1|SHA256`	Verificación, dc3dd, QEMU
HashMyFiles	`HashMyFiles.exe /enable_hash 1|2|8 /file|/folder … [/wildcard … /subfolders 1] /stext <rep>`	Verificación
robocopy	`robocopy <o> <d> [archivos] /E /COPYALL|/COPY:DAT /NP /TEE /LOG: /J /R:1 /W:1 /NFL /NDL /NJH /NJS /NC /NS`	9 flujos distintos
vmrun	`vmrun list` · `vmrun suspend <vmx>` · `vmrun start <vmx>` · `vmrun stop <vmx> soft` · `vmrun listSnapshots <vmx>`	VMware
vmware-vdiskmanager	`vmware-vdiskmanager -r <origen> -t 0 <destino>`	VMware
VBoxManage	`list vms` · `list runningvms` · `list hdds` · `showvminfo <id> --machinereadable` · `snapshot <id> list [--machinereadable]` · `clonemedium disk <src> <dst> [--format …]` · `clonevm <id> --snapshot <s> --mode machine --name … --basefolder … --register` · `unregistervm <n> --delete` · `debugvm <id> dumpvmcore --filename <f>`	VirtualBox
Hyper-V (PowerShell)	`Get-VM` · `Get-VMHardDiskDrive` · `Get-VMSnapshot` · `Get-VHD` · `Set-VHD -ParentPath … -IgnoreIdMismatch` · `Merge-VHD -Force -Confirm:$false` · `Checkpoint-VM -SnapshotName … -Passthru` · `Set-VM -CheckpointType Standard` · `Save-VM` · `Start-VM` · `Stop-VM` · `Remove-VMSnapshot`	Hyper-V

HERRAMIENTA	COMANDO TAL COMO SE EJECUTA	MÓDULO
hv_savedstate_to_raw	`powershell -File hv_savedstate_to_raw.ps1 -SavedStatePath <f> [-VsvPath <f>] -OutputRaw <f>`	Hyper-V
qemu-img	`qemu-img info <archivo>` · `qemu-img convert -p -f <fmt> -O raw <o> <d>`	QEMU
docker	`docker version` · `docker ps -a --format …` · `docker export <id> -o <tar>` · `docker inspect <id>` · `docker inspect --format "{{.State.Running}}"` · `docker exec <id> sh -c "ls -1pL …"` · `docker exec <id> sh -c "test -d|-e …"` · `docker cp <id>:<ruta> <destino>`	Docker
adb	`adb devices [-l]` · `adb connect <ip:puerto>` · `adb disconnect` · `adb kill-server` · `adb -s <s> get-state` · `adb -s <s> shell ls -1p <ruta>` · `adb -s <s> shell test -d|-e <ruta>` · `adb -s <s> pull -a <o> <d>`	ADB
7zr	`7zr.exe a -t7z <contenedor>.7z <artefacto>`	Docker, ADB
Storage (PowerShell)	`Get-Disk` · `Set-Disk -Number <n> -IsReadOnly $true\|$false`	Write Blocker
WMI / WMIC	`wmic diskdrive get Index,Model,Size /value` · `wmic logicaldisk where … get Size,FreeSpace /value` · `wmic os get LocalDateTime` · `Get-WmiObject Win32_LogicalDisk` · `Get-WmiObject Win32_DiskDrive` · `Get-PhysicalDisk`	Transversal
Sistema	`reg query` · `sc query vmms` · `bcdedit /enum {current}` · `mode con` · `net session` · `findstr` · `choice` · `timeout` · `copy /y` · `move /Y` · `del` · `mkdir` · `rmdir /s /q` · `type` · `pushd/popd`	Transversal

18.2 Glosario de flags de robocopy

robocopy es la herramienta con más variantes de invocación del framework, y cada flag responde a un motivo concreto: preservar metadatos NTFS, no inundar un log o no bloquearse durante horas en un archivo que ya no está.

FLAG	SIGNIFICADO	DÓNDE SE USA
/E	Subdirectorios incluidos los vacíos	Adquisición lógica, volúmenes Docker, clones completos, config Hyper-V
/COPYALL	`/COPY:DATSOU`: datos, atributos, timestamps, ACLs, propietario, auditoría	Adquisición lógica de archivos y carpetas
/COPY:DAT	Datos, atributos y timestamps, sin ACLs	Clones completos y volúmenes Docker, para que la copia sea abrible en otra máquina
/J	E/S sin buffer (solo Windows 8+, variable `RBC_J`)	Archivos gigantes: `.vmem`, `.vhdx`, estados guardados
/NP	Sin porcentaje de progreso	Cuando se escribe a un log, para no inundarlo
/TEE	Salida a consola y a log simultáneamente	Adquisición lógica
/LOG:	Archivo de log de la copia	Adquisición lógica
/R:1 /W:1	Un reintento con un segundo de espera	Clones y volúmenes: evita el millón de reintentos por defecto
/NFL /NDL	Sin lista de archivos ni de directorios	Copias masivas donde el detalle sería ruido
/NJH /NJS	Sin cabecera ni resumen de trabajo	Copias en bucle, una por archivo o extensión
/NC /NS	Sin clase de archivo ni tamaños	Copias de memoria y discos individuales

18.3 Glosario de códigos y centinelas internos

Valores que viajan entre los módulos `.bat` y los scripts PowerShell. Los centinelas son cadenas literales: el llamador las compara tal cual, así que cualquier cambio de texto rompe el protocolo.

VALOR	ORIGEN	SIGNIFICADO
`0` / `1` / `2`	`forensic_wrapper.ps1`	Éxito / error de la herramienta / abortado por el operador
`99`	`launch_noctrlc.ps1`	El lanzador falló: continuar en modo degradado
`3`	`hv_savedstate_to_raw.ps1`	`vmsavedstatedumpprovider.dll` no encontrado
`≥ 8`	robocopy	Fallo real de copia; 0-7 son variantes de éxito
`@ACQ_CMD_STR`	`:ACQUIRE` → `:LOG_CMD`	Centinela: el comando viaja por variable, no por argumento
`__NOEXISTE__`	Write Blocker	El número de disco consultado no existe
`NONE`	Scripts PS de Hyper-V	No se encontraron discos o archivos de estado
`NO_VOLUMES`	Script PS de Docker	El contenedor no tiene volúmenes montados
`NO_CHAIN`	Consolidación Hyper-V	No hay `.avhdx` que consolidar
`MERGE_OK` / `MERGE_FAIL`	Consolidación Hyper-V	Resultado de `Merge-VHD` y de la copia posterior
`FILE|` / `OLDTYPE|` / `PSERROR|`	Checkpoint Hyper-V	Protocolo de tres prefijos entre el `.ps1` y el `.bat`
`base`	`VBoxManage list hdds`	Valor de `Parent UUID` que marca la raíz de la cadena
`ffffffff`	Descriptor VMDK	`parentCID` que indica disco base sin padre

CAPÍTULO 19

Anexos

Material de referencia: la estructura que la evidencia ocupa en disco, el inventario de herramientas embebidas, el laboratorio que quedó deliberadamente fuera del menú y las fuentes metodológicas que sostienen las decisiones.

19.1 Estructura completa de la evidencia generada

```
Evidencias\
├── Windows\
│   ├── Volcado\                        memdump.raw · memdump.raw_hashes.txt
│   ├── Adquisicion\<nombre>\           <img>.dd|.E01|.Ex01|.S01 · <img>.log (dc3dd)
│   │                                   <img>_Hashes.txt · <img>_VERIFICACION.txt
│   │                                   Forensic_Journal.log
│   └── AdquisicionLogica\<nombre>\     copia con metadatos NTFS
│                                       <nombre>_robocopy.log · <nombre>_hashes.txt
├── KAPE\<EQUIPO>_<timestamp>\          <ts>_<nombre>.vhdx|.zip · hashes_kape.txt
├── VMware\<VM>\
│   ├── Disks\                          <disco>.vmdk consolidado · <disco>_Hashes.txt
│   ├── Memory\                         *.vmem *.vmss *.vmsn · Memoria_VM_Hashes.txt
│   ├── Config\                         *.vmx *.vmxf *.vmsd *.nvram · hashes.sha256
│   ├── Snapshots\                      cadena diferencial preservada · hashes.sha256
│   ├── Reconstruccion_Snapshots\
│   │   ├── hashes_originales_PREcopia.sha256
│   │   ├── _trabajo_<snap>\            área de trabajo (copia de la cadena)
│   │   └── Consolidado_<snap>\         <snap>_reconstruido.vmdk
│   │       ├── hashes_evidencia_derivada.sha256
│   │       ├── Memoria\                *.vmsn *.vmem · hashes_memoria.sha256
│   │       └── Config\                 *.vmx … · hashes.sha256
│   └── Clon_Completo_VMware\
│       ├── hashes_originales_PREcopia.sha256
│       ├── hashes_copia_POSTcopia.sha256
│       └── VM\                         carpeta íntegra sin consolidar
├── VirtualBox\<VM>\
│   ├── Disks\                          clon .vdi|.vmdk|.vhd|.raw · hashes.sha256
│   │                                   <VM>_<snap>.<ext> (modo 4) · hashes_snapshot.sha256
│   ├── Memory\                         <VM>.elf · FORMAT_NOTE.txt · hashes.sha256
│   ├── Config\                         <VM>.vbox · hashes.sha256
│   ├── Snapshots\                      *.vdi *.vmdk *.vhd · hashes.sha256
│   └── Clon_Completo_VirtualBox\       VM\ + hashes PRE y POST
├── HyperV\<VM>\
│   ├── Disks\                          *.vhdx *.avhdx · hashes.txt
│   ├── Memory\                         <Id>.vmrs|.bin+.vsv|.vmgs · <Id>.raw
│   │                                   FORMAT_NOTE.txt · hashes.txt
│   ├── Config\                         *.vmcx *.vmrs *.xml · hashes.sha256
│   ├── Consolidation\                  <VM>_consolidated.vhdx · hashes.txt
│   │                                   (Working\ se elimina al terminar)
│   └── Clon_Completo_HyperV\           VM\ + hashes PRE y POST
├── Docker\<contenedor>\
│   ├── ContainerExport\                <nombre>_filesystem.tar · hashes.txt
│   ├── Volumes\<volumen>\              copia de cada mount · hashes.txt
│   └── Artifacts\                      <ruta_saneada>.7z (+ __N si duplicado)
│                                       <artefacto>.7z_hashes.txt
└── ADB\<serial>\
    └── Artifacts\                      <ruta_saneada>.7z · <artefacto>.7z_hashes.txt
```

```
Script\Logs\Forensic_Commands.log     registro central de comandos de toda la sesión
```

Árbol que resulta de una sesión que hubiera recorrido todos los módulos.

Cuando se usa QEMU, en la carpeta de destino aparecen además `<nombre>.raw|.img`, `destino_hashes.txt` y `conversion_report.txt`.

19.2 Inventario de herramientas embebidas

RUTA BAJO **SISTEMA\HERRAMIENTAS**	CONTENIDO
`Adquisicion\Discos\dc3dd`	`dc3dd.exe` + DLLs de Cygwin (`cygwin1.dll`, `cygiconv-2.dll`, `cygintl-8.dll`, `cyggcc_s-seh-1.dll`)
`Adquisicion\Discos\dd`	`dd.exe` + documentación de licencia y cambios
`Adquisicion\Discos\ewf`	Suite libewf: `ewfacquire`, `ewfacquirestream`, `ewfexport`, `ewfinfo`, `ewfverify` + `libewf.dll` y `zlib.dll`
`Adquisicion\Discos\ftk`	`ftkimager.exe`
`Volcados\RAM\comae\{x64,x86,ARM64}`	`DumpIt.exe` + conversores `Bin2Dmp`, `Dmp2Bin`, `Dmp2Json`, `Hibr2Bin`, `Hibr2Dmp`, `Z2Dmp`
`Volcados\RAM\winpmem`	`winpmem_mini_x64_rc2.exe`, `winpmem_mini_x86.exe`
`Volcados\Virtual\vm2dump`	`vm2dmp.exe` (conversor legacy de Hyper-V, ≤ 2008R2)
`Hashes\HashMyFiles\{hashmyfiles-x64,x32}`	`HashMyFiles.exe` en ambas arquitecturas
`Triaje\kape`	`kape.exe`, `gkape.exe`, `Targets\` (267 `.tkape`), `Modules\`, `Documentation\`, `Get-KAPEUpdate.ps1`
`Virtualizacion\VMware\vdiskmanager\bin`	`vmware-vdiskmanager.exe`, `vixDiskCheck.exe`, `vddkReporter.exe`
`Virtualizacion\VMware\vmrun`	`vmrun.exe`, `vnetlib.exe`, `vnetlib64.exe`
`Virtualizacion\VirtualBox`	`VBoxManage.exe`, `VBoxSVC.exe`, `VBoxDrvInst.exe`
`Virtualizacion\QEMU`	`qemu-img.exe` + bibliotecas y recursos
`Utilidades\7zip`	`7zr.exe` (build reducida, solo formato `.7z`)
`Utilidades\Sysinternals`	`pslist`, `logonsessions`, `PsLoggedon`, `sigcheck`, `tcpvcon`, `tcpview` (x86 y x64)
`platform-tools`	`adb.exe`, `fastboot.exe`, `sqlite3.exe`, `AdbWinApi.dll`, `AdbWinUsbApi.dll` y utilidades del SDK
`Terminal\terminal-1.24.11321.0`	`OpenConsole.exe`
`volatility3-develop`	Volatility 3 completo (`vol.py`, `volshell.py`, framework y plugins) para el análisis posterior

RUTA BAJO `SISTEMA\HERRAMIENTAS\`	CONTENIDO
`dwarf2json`	`dwarf2json.exe` + generador de símbolos de ejemplo para perfiles Linux de Volatility

El módulo `Sistema\lib\` contiene los tres scripts PowerShell del framework, y `Sistema\poc\` el laboratorio experimental que describe la sección siguiente.

19.3 El módulo de prueba de concepto no integrado

El proyecto incluye `Sistema\poc\vm_guest_ops_poc.ps1`, un laboratorio experimental marcado explícitamente como **no integrado** en `ForensicScript.bat` ni en su menú. No es código de producción: ninguna ruta del framework lo invoca. Su propósito es validar la viabilidad técnica de tres operaciones dentro del sistema invitado, ejecutadas desde el host:

TECNOLOGÍA	VÍA DE ACCESO AL GUEST	REQUISITO
VMware	`vmrun`: copiar archivos host↔guest, ejecutar programas	Credenciales válidas del SO invitado
VirtualBox	`VBoxManage guestcontrol`: las mismas tres operaciones	Credenciales válidas del SO invitado
Hyper-V	PowerShell Direct (`Invoke-Command -VMName -Credential`), comparado con el montaje de solo lectura `Mount-VHD`	Credenciales; el montaje `Mount-VHD` **no** las requiere ni exige encender la VM

ADVERTENCIA

1. Requiere credenciales, sin excepción. En VMware y VirtualBox, *todas* las operaciones de acceso al guest exigen usuario y contraseña válidos del sistema invitado: no existe excepción documentada ni bypass oficial. Eso limita su utilidad a escenarios donde el operador ya las conoce: investigación corporativa con cooperación del área de TI, laboratorio propio, o credenciales entregadas por orden judicial. En el escenario típico de incautación sin cooperación del sospechoso esta vía **no es aplicable**, y el framework principal debe seguir dependiendo exclusivamente de la copia bit a bit del disco virtual (RFC 3227, SANS FOR498).

2. Altera el sistema evaluado. Ejecutar comandos o programas dentro del guest genera artefactos nuevos: entradas en Prefetch y Amcache, registros en Event Logs, claves MRU y Run en el registro, y modificación de timestamps de acceso. Es una concesión metodológica deliberada, **no una técnica de preservación forense pura**. Cualquier uso real debe documentarse explícitamente en la cadena de custodia como «adquisición no forense pura / adquisición en vivo justificada».

Su inclusión en el proyecto (aislado, documentado y con sus limitaciones enunciadas) tiene valor metodológico: demuestra que la vía fue investigada y explica con fundamento por qué no se integró, en lugar de dejar la ausencia sin justificar.

19.4 Referencias metodológicas empleadas

REFERENCIA	APLICACIÓN CONCRETA EN EL FRAMEWORK
RFC 3227: Guidelines for Evidence Collection and Archiving	Orden de volatilidad: la memoria RAM es la opción 1 del menú, antes de cualquier operación sobre disco. Fundamenta también la preferencia por la copia bit a bit frente al acceso al guest.
NIST SP 800-86: Guide to Integrating Forensic Techniques into Incident Response	Citada explícitamente en el log al suspender una VM: justifica la *alteración mínima, controlada y documentada* cuando es necesaria para obtener evidencia válida. Se aplica igualmente al checkpoint estándar de Hyper-V y a `Save-VM`.
SANS FOR498: Battlefield Forensics & Data Acquisition	Marco de referencia para la adquisición de discos virtuales y la distinción entre evidencia primaria y derivada. Los targets `!SANS_Triage` de KAPE siguen esta metodología.
Documentación de Microsoft: `VmSavedStateDumpProvider`	API oficial usada para convertir el estado guardado de Hyper-V a memoria RAW, siguiendo la secuencia del ejemplo `rawmemtofile.cpp`.
Especificación *VMware Virtual Disk Format* 1.1/5	Campos `CID`, `parentCID` y `parentFileNameHint` para clasificar descriptores por contenido y reconstruir la cadena de snapshots.

19.5 Resumen: el framework en una tabla

DIMENSIÓN	CIFRA
Script principal	Un único `ForensicScript.bat` que concentra los flujos y las subrutinas compartidas
Menús y pantallas de módulo	13
Flujos de adquisición completos	22
Herramientas externas invocadas	19 binarios embebidos + utilidades del sistema
Tecnologías de virtualización cubiertas	4 (VMware, VirtualBox, Hyper-V, QEMU)
Comprobaciones en la pantalla de arranque	21
Cobertura del wrapper anti-CTRL+C	Toda adquisición que pueda tardar horas
Cobertura de la verificación de integridad	Toda evidencia adquirida, a decisión del operador
Módulos PowerShell de apoyo	3 (+1 PoC no integrado)
Targets de KAPE disponibles	267
Sistemas operativos soportados	Windows 7 a Windows 11, en x86, x64 y ARM64