Dump Memory Objects of Interest

Many Volatility™ 3 plugins have an option to "**--dump**" objects:
pslist, psscan, dlllist, modules, modscan, malfind

vol.py -f mem.img windows.pslist.PsList --pid 840 --dump

Extraction plugins also exist for other Windows memory objects:

windows.memmap.Memmap

windows.filescan.FileScan

windows.dumpfiles.DumpFiles

windows.mftscan.MFTScan

windows.svcscan.SvcScan

Memory Analysis with YARA

Volatility™ 3 VadYaraScan

--yara-file Text file of YARA rules

vol.py -f mem.img windows.vadyarascan --yara-file *rules*

MemProcFS YARA Integration

MemProcFS includes built-in YARA signatures from Elastic Security
Add to Command line: **-license-accept-elastic-license-2-0**
Built-in YARA hits: **M:\forensic\csv\findevil.csv**

Include Custom Signatures: **-forensic-yara-rules *rules***
Custom YARA hits: **M:\forensic\yara**

Memory Acquisition

Execute command terminal as Administrator

WinPmem

https://github.com/Velocidex/WinPmem

-d Output to <filename>
-l Load driver for live memory analysis

winpmem_mini_x64_<version>.exe -d D:\mem.img (64-bit)

Magnet DumpIt

https://for508.com/dumpit

/OUTPUT Image destination
/TYPE Memory output format (RAW | DMP)
/NOCOMPRESS Do not compress output when > 32GB

DumpIt.exe /TYPE DMP /OUTPUT D:\mem.img

Live Memory Scanning

Powerful capabilities exist to scan processes for anomalies on live systems. Useful for hunting and memory research.

Administrator command terminal is required

Moneta

Memory scanning tool looking for dynamic/unknown code, suspicious PE image regions, and advanced indicators of compromise
https://github.com/forrest-orr/moneta

-p Process IDs to scan (* for all)
-m ioc Scan only suspicious memory regions (**-m *** for all)
-d Dump selected process memory to local file system
--filter Limit scans to reduce false positives
(* | unsigned-module | metadata-modules | clr-heap | clr-prvx | wow64-init)

moneta64.exe -m ioc -p * --filter * -d

Hollows_Hunter

Identifies potential process implants, shellcode, hooks, and in-memory patches
https://github.com/hasherezade/hollows_hunter

/pname Scan specific processes by name
/pid Scan specific processes by PID
/dnet Set policy for skipping .NET processes
/hooks Detect code patches and inline hooks (noisy)
/dir Directory to save dumps and reporting

hollows_hunter64.exe /pid 1290;454 /dir .\Output

Get-InjectedThreadEx

Find suspicious threads (and associated processes) indicative of code injection
https://github.com/jdu2600/Get-InjectedThreadEx

Get-InjectedThreadEx.exe > .\output.txt

Alternate Windows Memory Locations

Hibernation File (Compressed)

C:\hiberfil.sys

Page and Swap Files

C:\pagefile.sys
C:\swapfile.sys (Windows 8+ \ Server 2012+)

Crash Dump

C:\Windows\MEMORY.DMP

In rare instances locations can differ from the defaults (except hiberfil.sys)

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

Memory Forensics Cheat Sheet v3.0

POCKET REFERENCE GUIDE

SANS Institute
http://dfir.sans.org

by Chad Tilbury
http://sans.org/for508

Purpose

This reference supports the SANS Institute FOR508 Advanced Incident Response, Threat Hunting, and Digital Forensics Course. It is not intended to be an exhaustive resource for MemProcFS, Volatility™, or any other tools. Volatility™ is a trademark of the Volatility Foundation. The SANS Institute is not sponsored, approved by, or affiliated with the Volatility Foundation.

How To Use This Document

Memory analysis is one of the most powerful tools available to forensic examiners. This guide aims to document and simplify the overwhelming number of tools and available capabilities.

Windows memory analysis can generally be split into six steps:

1. Identify Rogue Processes
2. Analyze Process Objects
3. Review Network Artifacts
4. Look for Evidence of Code Injection
5. Audit Drivers and Rootkit Detection
6. Dump Memory Objects of Interest

In this reference guide we outline the most useful MemProcFS and Volatility™ capabilities to support these six stages of memory forensics. Further information is provided for:

- Memory Acquisition
- Live Memory Scanning
- Using Indicators of Compromise
- Alternate Windows Memory Locations

MemProcFS

MemProcFS (Windows Memory Analysis)
https://github.com/ufrisk/MemProcFS

MemProcFS.exe [options] -device <memory image>

-device: Memory image (includes hibernation file support)
-v: Enable verbose auditing in console
-pagefile0: Specify pagefile.sys file (not required)
-pagefile1: Specify swapfile.sys file (not required)
-mount: Drive letter for analysis output (M:\ is default)
-forensic [0-4]: Start forensic scan of memory upon startup
0 = not enabled (default value)
1 = forensic mode with in-memory sqlite database
2 = forensic mode with temp sqlite database deleted upon exit
3 = forensic mode with temp sqlite database remaining upon exit
4 = forensic mode with static named sqlite database (vmm.sqlite3)

Processes:
Process Tree: **M:\sys\proc\proc.txt**
CSV (requires **-forensic**): **M:\forensic\csv\process.csv**

Process Objects:
Objects represented as files. Use a simple copy/paste for "dumping"
By PID: **M:\pid**
By Name: **M:\name**

Network Artifacts:
Text: **M:\sys\net\netstat.txt**
CSV (requires **-forensic**): **M:\forensic\csv\net.csv**

Code Injection and Anomaly Detection (requires **-forensic**):
Text: **M:\forensic\findevil\findevil.txt**
CSV: **M:\forensic\csv\findevil.csv**

Cached Files (requires **-forensic**):
Extracted files in virtualized file system: **M:\forensic\files**
List of available cached files: **M:\forensic\csv\files.csv**

Other Analysis Capabilities (most require **-forensic**):
Virtualized Registry: **M:\registry**
MFT Virtualized File System: **M:\forensic\ntfs**
Drivers: **M:\forensic\csv\drivers.csv**
Services: **M:\forensic\csv\services.csv**
Scheduled Tasks: **M:\forensic\csv\tasks.csv**
Forensic Timeline: **M:\forensic\csv\timeline_all.csv**

Getting Started with Volatility™ 3

Getting Help (Windows / Linux / Mac Memory Analysis)
https://github.com/volatilityfoundation/volatility3

vol.py -h (show options and supported plugins)
vol.py *plugin* -h (show plugin usage)

Sample Command Line
vol.py -f *mem.img plugin*

Query Memory Image Metadata (OS Profile & SystemTime)
vol.py -f mem.img windows.info.Info

Create and use JSON Config File to Accelerate Processing
vol.py --write-config -f mem.img windows.info.Info

vol.py -c config.json -f mem.img *plugin*

Output and Format Options
These options must precede the plugin within the command-line:
-r *<csv | pretty | json>* Output format
-o *folder* Output folder for extracted items (useful with **--dump**)
Plugin specific options must follow the plugin name:
--pid *PID1,PID2* Limit data to specific process IDs (most plugins)
Plugin names can be shortened if they still result in a unique match:

vol.py -f mem.img -r csv windows.pslist --pid 4

Identify Rogue Processes

pslist - High level view of running processes
--dump Extract process executables

vol.py -f mem.img windows.pslist.PsList

psscan - Deep scan of memory for EPROCESS blocks

vol.py -f mem.img windows.psscan.PsScan

pstree - Display parent-process relationships
--pid Display mini-process tree for single parent process

vol.py -f mem.img windows.pstree.PsTree

Review Network Artifacts

netstat - Display data from network tracking structures

vol.py -f mem.img windows.netstat.NetStat

netscan - Deep scan for network connections and sockets
--include-corrupt Relax validation for more results

vol.py -f mem.img windows.netscan.NetScan

Analyze Process Objects

dlllist - List of loaded DLLs by process
--dump Extract DLLs from the memory image

vol.py -f mem.img windows.dlllist.DllList --pid 840

cmdline - Display process command lines from PEB

vol.py -f mem.img windows.cmdline.CmdLine

getsids - Print process security identifiers

vol.py -f mem.img windows.getsids.GetSIDs

handles - List of open handles for each process
Pipe results to `egrep` to display only handles of a certain type:

vol.py -f mem.img windows.handles.Handles --pid 840 | egrep 'File|Key|Mutant'

Look for Evidence of Code Injection

malfind - Find suspicious RWX sections not mapped to disk
--dump Save suspicious memory sections to a folder

vol.py -f mem.img -o tmp windows.malfind.Malfind --dump

ldrmodules - Detect unlinked DLLs

vol.py -f mem.img windows.ldrmodules.LdrModules

Audit Drivers and Rootkit Detection

modules - View list of loaded kernel drivers
--dump Extract listed drivers
--name *driver* Info on named driver (can use with --dump)

vol.py -f mem.img windows.modules.Modules --name ks.sys

modscan - Scan for loaded, unloaded, and unlinked drivers
--dump Extract all available drivers

vol.py -f mem.img -o tmp windows.modscan.ModScan --dump

ssdt - Output System Service Descriptor Table

vol.py -f mem.img windows.ssdt.SSDT

driverirp - Print driver IRP (major function) tables

vol.py -f mem.img windows.driverirp.DriverIrp