SANS DFIR Memory Forensics Cheat Sheet v2.1

This guide was created by by Chad Tilbury | http://forensicmethods.com

Purpose

This cheat sheet supports the SANS **FOR508: Advanced Incident Response, Threat Hunting, and Digital Forensics** course. It is not intended to be an exhaustive resource for Volatility™ or other highlighted tools. Volatility™ is a trademark of Verizon. The SANS Institute is not sponsored, approved by or affiliated with Verizon.

How To Use This Document

Memory analysis is one of the most powerful tools available to forensic examiners. This guide hopes to simplify the overwhelming number of available options.

Analysis can generally be accomplished in six steps:

1. Identify Rogue Processes
2. Analyze Process DLLs and Handles
3. Review Network Artifacts
4. Look for Evidence of Code Injection
5. Check for Signs of a Rootkit
6. Extract Processes, Drivers, and Objects

We outline the most useful Volatility™ plugins supporting these six steps here. Further information is provided for:

- Memory Acquisition
- Alternate Memory Locations
- Converting Hibernation Files and Crash Dumps
- Memory Artifact Timelining
- Registry Analysis Plugins

Getting Started with Volatility™

Getting Help

`# vol.py -h`	Show options and supported plugins
`# vol.py plugin -h`	Show plugin usage
`# vol.py plugin --info`	Show available OS profiles

Sample Command Line

```
# vol.py -f image --profile=profile plugin
```

Identify System Profile

`imageinfo` Display memory image metadata

```
# vol.py -f mem.img imageinfo
```

Using Environment Variables

Set name of memory image Takes place of `-f`

```
# export VOLATILITY_LOCATION=file:///images/mem.img
```

Set profile type Takes place of `--profile=`

```
# export VOLATILITY_PROFILE=Win10x64_14393
```

Identify Rogue Processes

`pslist` High-level view of running processes

```
# vol.py pslist
```

`psscan` Scan memory for EPROCESS blocks

```
# vol.py psscan
```

`pstree` Display parent-process relationships

```
# vol.py pstree
```

Identify Rogue Processes

`dlllist` List of loaded dlls by process

`-p` Show information only for specific processes (PIDs)

```
# vol.py dlllist -p 1022,868
```

`getsids` Print process security identifiers

`-p` Show information only for specific PIDs

```
# vol.py getsids -p 868
```

`handles` List of open handles for each process

`-p` Show information only for specific PIDs

`-t` Display only handles of a certain type {Process, Thread, Key, Event, File, Mutant, Token, Port}

```
# vol.py handles -p 868 -t File,Key
```

Review Network Artifacts

`netscan` Scan for TCP connections and sockets

```
# vol.py netscan
```

Note: Use connscan and sockscan for XP systems

Look for Evidence of Code Injection

`malfind` Find injected code and dump sections

`-p` Show information only for specific PIDs

`-v` Verbose: show full paths from three DLL lists

```
# vol.py ldrmodules -p 868 -v
```

`hollowfind` Detect process hollowing techniques

`-p` Show information only for specific PIDs

`-D` Directory to save suspicious memory sections

```
# vol.py hollowfind -D ./output_dir
```

Check for Signs of a Rootkit

`psxview` Find hidden processes using cross-view

```
# vol.py psxview
```

`modscan` Scan memory for loaded, unloaded, and unlinked drivers

```
# vol.py modscan
```

`apihooks` Find API/DLL function hooks

`-p` Operate only on specific PIDs

`-Q` Only scan critical processes and DLLS

```
# vol.py apihooks
```

`ssdt` Hooks in System Service Descriptor Table

```
# vol.py ssdt | egrep -v '(ntoskrnl|win32k)'
```

`driverirp` Identify I/O Request Packet (IRP) hooks

`-r` Analyze drivers matching REGEX name pattern

```
# vol.py driverirp -r tcpip
```

`idt` Display Interrupt Descriptor Table

```
# vol.py idt
```

Extract Processes, Drivers, and Objects

dlldump Extract DLLs from specific processes

- **-p** Dump DLLs only for specific PIDs
- **-b** Dump DLL using base offset
- **-r** Dump DLLs matching REGEX name

```
# vol.py dlldump --dump-dir ./output -r metsrv
```

moddump Extract kernel drivers

- **-b** Dump driver using offset address (from modscan)
- **-r** Dump drivers matching REGEX name
- **--dump-dir** Directory to save extracted files

```
# vol.py moddump --dump-dir ./output -r gaopdx
```

procdump Dump process to executable sample

- **-p** Dump only specific PIDs
- **-o** Specify process by physical memory offset
- **-n** Use REGEX to specify process
- **--dump-dir** Directory to save extracted files

```
# vol.py procdump --dump-dir ./output -p 868
```

memdump Extract every memory section into one file

- **-p** Dump memory sections from these PIDs
- **-n** Use REGEX to specify process
- **--dump-dir** Directory to save extracted files

```
# vol.py memdump --dump-dir ./output -p 868
```

filescan Scan memory for FILE_OBJECT handles

```
# vol.py filescan
```

dumpfiles Extract FILE_OBJECTs from memory

- **-Q** Dump using physical offset of FILE_OBJECT
- **-r** Extract using a REGEX (add -i for case insensitive)
- **-n** Add original file name to output name
- **--dump-dir** Directory to save extracted files

```
# vol.py dumpfiles -n -i -r \\.exe --dump-dir=./
```

svcscan Scan for Windows Service record structures

- **-v** Show service DLL for svchost instances

```
# vol.py svcscan -v
```

cmdscan Scan for COMMAND_HISTORY buffers

```
# vol.py cmdscan
```

consoles Scan for CONSOLE_INFORMATION output

```
# vol.py consoles
```

Memory Acquisition

Remember to open command prompt as Administrator

winpmem

- **-o** Output file location
- **-p** **<path to pagefile.sys>** Include page file
- **-e** Extract raw image from AFF4 file
- **-l** Load driver for live memory analysis

```
C:\> winpmem_<version>.exe -o F:\mem.aff4
C:\> winpmem_<version>.exe F:\mem.aff4 -e
  PhysicalMemory -o mem.raw
```

DumpIt

- **/f** Output file location
- **/s** **<value>** Hash function to use
- **/t** **<addr>** Send to remote host (set up listener with /l)

```
C:\> DumpIt.exe /f F:\mem.raw /s 1
```

Alternate Memory Locations

Hibernation File

Compressed RAM Image; available in Volume Shadow Copies

```
%SystemDrive%\hiberfil.sys
```

Page and Swap Files

```
%SystemDrive%\pagefile.sys
%SystemDrive%\swapfile.sys (Win8+\2012+)
```

Memory Dump

```
%WINDIR%\MEMORY.DMP
```

Converting Hibernation Files and Crash Dumps

imagecopy Convert alternate memory sources to raw

- **-f** Name of source file
- **-o** Dump DLL using base offset
- **--profile** Source OS from **imageinfo**

```
# vol.py imagecopy -f hiberfil.sys -O hiber.
  raw --profile=Win7SP1x64
# vol.py imagecopy -f MEMORY.DMP -O crashdump.
  raw --profile=Win2016x64_14393
```

How To Use This Document

The **timeliner** plugin parses time-stamped objects found in memory images. Output is sorted by:

- Process creation time
- Thread creation time
- Driver compile time
- DLL/EXE compile time
- Network socket creation time
- Memory resident registry key last write time
- Memory resident event log entry creation time

timeliner

- **--output-file** Optional file to write output
- **--output=body** Bodyfile format (also **text,xlsx**)
- **--type=Registry** Extract registry key last write times

```
# vol.py -f mem.img timeliner --output-file out.
  body --output=body --profile=Win10x64
```

Registry Analysis Plugins

hivelist Find and list available registry hives

```
# vol.py hivelist
```

hivedump Print all keys and subkeys in a hive

- **-o** Offset of registry hive to dump (virtual offset)

```
# vol.py hivedump -o 0xe1a14b60
```

printkey Output a registry key, subkeys, and values

- **-K** Registry key path

```
# vol.py ol.py printkey -K "Microsoft\Windows\
  CurrentVersio
```

dumpregistry Extract all available registry hives

- **-o** Extract using virtual offset of registry hive
- **--dump-dir** Directory to save extracted files

```
# vol.py printkey -K "Microsoft\Windows\
  CurrentVersion\Run"
```

hashdump Dump user NTLM and Lanman hashes

```
# vol.py hashdump
```

autoruns Map ASEPs to running processes

- **-v** Show everything

```
# vol.py autoruns -v
```