## **Windows Memory Acquisition**

Windows WinPmem (Open cmd.exe as Administrator)

As of winpmem 2.0.1, the default output file format is AFF4

#### **Creating an AFF4**

C:\> winpmem <version>.exe -o output.aff4

#### Extracting the raw memory image from the AFF4

C:\> winpmem<version>.exe output.aff4 --export
PhysicalMemory -o memory.img

### **Other WinPmem Options:**

view aff4 metadata (-V) | elf output (--elf)

# **Live Windows Memory Analysis**

Windows WinPmem (Open cmd.exe as Administrator)

#### Loading the kernel driver on target system

C:\Program Files\Rekall> winpmem<version>.exe -1

#### **Creating Rekall session referencing live memory**

C:\Program Files\Rekall> Rekal -f \\.\pmem
[1] pmem 11:14:35> pslist

## Creating an image (AFF4) with aff4acquire

Linux PMEM (to create profile)

# tar vxzf linux pmem 1.0RC1.tqz

[1] pmem 11:14:35> aff4acquire output="w.aff4"

# **Linux Memory Acquisition**

```
# cd linux
# make
LinPMEM (to create image via /proc/kcore)
# gzip -d linpmem_2.0.1.gz
# chmod 755 linpmem_2.0.1
# ./linpmem_2.0.1 -o linux.aff4
# cd linux
# rekal convert_profile 3.11.0-26-generic.zip
Ubuntu.zip
# rekal --profile=Ubuntu.zip -f ../linux.aff4
```

## **Registry Analysis Plugins**

## Enumerate and Extract Registry Hives

hives-Find and list available registry hives
\$ rekal -f image.img hives

\_\_\_\_\_

**regdump**- Extracts target hive

--hive\_regex Regex Pattern Matching

- D "<dir>" Dump directory

\$ rekal -f image.img regdump --hive\_regex="SAM" -D
"/cases"

**printkey**- Output a registry key, subkeys, and values

-K "Registry key path"

userassist - Find and parse userassist key values

# **Additional Functionality**

analyze\_struct - Interprets and identifies windows
memory structures when given a virtual offset
[1] image.img 11:15:35> analyze\_struct 0x8180e6f0
vmscan - Allows for the identification of virtual machines
certscan - Dumps RSA private and public keys
dump dir= Dumps output to a specified directory

## **Mac OSX Memory Live Analysis &Acquisition**

<u>Mac OSXPmem</u> (Run commands with Root privileges) Extract osxpmem.zip and ensure file/dir permissions are root:wheel

#### Creating an AFF4

\$ sudo kextload MacPmem.kext

\$ sudo ./osxpmem --output test.aff4

\$ sudo kextunload MacPmem.kext/

<clean up by removing driver>

#### **Live OSX Memory Analysis**

\$ sudo kextload MacPmem.kext/

\$ rekal -f /dev/pmem

<begin interactive session>

\$ sudo kextunload MacPmem.kext/
<clean up by removing driver>



POCKET REFERENCE GUIDE

by Alissa Torres

## **Purpose**

The Rekall Memory Forensic Framework has unique syntax and plugin options specific to its features and capabilities. This cheatsheet provides a quick reference for memory analysis operations in Rekall, covering acquisition, live memory analysis and parsing plugins used in the 6-Step Investigative Process. For more information on this tool, visit rekall-forensic.com.

# **Rekall Memory Forensic Framework**

Memory analysis is one of the most powerful investigation techniques available to forensic examiners. Rekall auto-detects the target system's profile, using a repository of more than 100 kernel versions available either online or stored locally.

When launching Rekall, you can run single commands or drop into an interactive session to take advantage of caching, preventing the need to obtain the same data with subsequent plugin runs. This cheatsheet shows command line examples using both techniques.

# **Getting Started with Rekall**

Single Command Example

\$ rekal -f image.img pslist

Starting an Interactive Session

\$ rekal -f image.img



## **Memory Analysis Essentials**

#### Getting Help

[1] image.img 11:14:35> plugins.<tab> (lists plugins applicable for use for this image)

[1] image.img 11:14:35> pslist? (lists options available for specific plugin)

## Common Options in Interactive Session

-Short description of available plugins specific to image info - Specify amount of output (1-10, default=1) verbosity=# proc regex="process name"- regex to select process by name pid=<PID> - Filter plugin on a specific process by identifier dump dir="path to directory"- path to output directory output="path to output dir\file"-Required if outputting to file Exit interactive session quit

## Image Details

[1] image.img 11:14:35> imageinfo (list OS version, physical layout, uptime, time of collection)

## **Step 1. Enumerating Processes**

#### pslist - Enumerate processes

Rekall uses 5 techniques to enumerate processes by default (PsActiveProcessList, sessions, handles, CSRSS, PspCidTable)

[1] image.img 11:14:35> pslist

Narrow the process enumeration using "method="

[1] image.img 11:14:35 pslist method= "PsActiveProcessHead"

psinfo -Display detailed process & PE info [1] image.img 11:14:35> psinfo pid=<PID>

desktops - Enumerate desktops and desktop threads

[1] image.img 11:14:35> desktops verbosity=<#>

- Enumerate sessions and associated processes sessions

[1] image.img 11:14:35 > sessions

- Enumerates process threads threads

[1] image.img 11:14:35> threads proc regex= "chrome"

# Step 2. Analyze Process DLLs and Handles

dlllist - List of loaded dlls by process

Show information only for specific process identifiers (PIDs) [1] image.img 11:14:35> dlllist pid=[1580,204]

- List of open handles for each process handles pid=<pid> - Show information only for specific PIDs object types="TYPE" - Limit to handles of a certain type {Process, Thread, Key, Event, File, Mutant, Token, Port}

[1] image.img 11:14:35> handles pid=868, object types="Mutant"

**filescan** - Scan memory for FILE OBJECT handles [1] image.img 11:15:35> filescan output="filescan.txt"

# **Step 3. Review Network Artifacts**

netscan -Scan for connections and sockets in Vista+

[1] memory.aff4 11:14:35> netscan

ipconfig -ID TCP connections, including closed

[1] memory.aff4 11:14:35> ipconfig

- Dumps ARP cache

[1] memory.aff4 11:14:35> arp

dns cache-Dumps dns cache

[1] memory.aff4 11:14:35> dns cache

## **Step 4. Look for Evidence of Code Injection**

- Find injected code and dump sections malfind - Show information only for specific PIDs pid=<pid> - Provide physical offset of process to scan phys eprocess= - Provide virtual offset for process to scan eprocess= - Directory to save memory sections [1] be.aff4 11:14:35> malfind eprocess=0x853cf460 ldrmodules - Detect unlinked DLLs

**verbosity=** - Verbose: show full paths from three DLL lists

[1] be.aff4 11:14:35> ldrmodules pid=1936

# Step 5. Check for Signs of a Rootkit

- Find hidden processes using cross-view psxview - Scan memory for loaded, unloaded, and modscan

unlinked drivers

- Enumerates services from in-memory registry services

hive

- Scans for **SERVICE RECORD** records svcscan

hooks inline - Detects API hooks

> Filters by virtual address EProcess eprocess=

phys eprocess= Filters by physical address of EProcess

hooks eat - Detects Export Address Table hooks

[1] image.img 11:14:35> hooks\_eat pid=6764

**hooks iat** - Detects Import Address Table hooks

- Hooks in System Service Descriptor Table ssdt

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

regex="drivername" - Filter on REGEX name pattern

object tree - Tracks named objects

[1]image.img 11:15:35> object tree type regex="Driver"

# **Step 6. Dump Suspicious Processes and Drivers**

- Hexdump data starting a specified offset

[1] image.img 11:14:35> dump <virtual offset>

dlldump - Extract DLLs from specific processes

[1] image.img 11:14:35> dlldump pid=1004 dump dir=.

- Extract kernel drivers moddump

procinfo -Dump process to executable sample

[1] image.img 11:14:35> dlldump pid=1004 dump dir=.

procdump - Dump process to executable sample

pid= Dump only specific PIDs

offset=Specify process by physical memory offset

dump-dir=Directory to save extracted files

[1] image.img 11:14:35> procdump proc regex="csrss" dump-dir="/tmp"

- Dump every memory section into a file memdump (command line options shown below)

- p <PID>- Dump memory sections from these PIDs

-D /cases -Directory to save extracted files

# rekal -f image.aff4 memdump -D ./output -p 1004