Rekall Memory Forensic Framework

The Rekall Memory Forensic Framework is a library of memory analysis and attack tools implemented in Python and used to support advanced memory analysis for digital forensics. It includes the Rekall Memory Forensic Framework, Python, and the Rekall Memory Forensic Framework for Windows. The Rekall Memory Forensic Framework offers extensive documentation on how to use it, visit rekall-framework.org.

Getting Started with Rekall

```bash
$ rekal -f image.img
$ rekal -f image.img pslist
```

Single Command Example

More information on this tool, visit analysis, and parsing plugins used in the Six-Step Investigative Process. For memory analysis operations in Rekall, covering acquisition, live memory

General Public License. This cheatsheet provides a quick reference for...artifacts, and handles

Identifying Rogue DLLs

Tips

- Use the `pslist` command to enumerate processes
- Use the `handle` command to identify handles
- Use the `dlllist` command to list DLLs

Rekall's `ptov` command can be used for virtual to physical address translation. Both Rekall and Volatility offer plugins that provide this `ptov` functionality. With `ptov`, we can identify the `strings` plugin. Rekall has two different plugins that offer physical to virtual address translation: `gptov` and `pcaov`. These plugins employ different methods in determining which process has been allocated the frame in physical memory where the keyword lies. Regardless of the method used, the end result is a reverse lookup of keywords to owning process.

```
$ rekal -f test.img ptov 21732272
```

Extracting Process Details

```
$ rekal -f image.img process_create_time
```

Malicious Code Detection

```
$ rekal -f image.img key
```

Windows Memory Forensic Acquisition (winpmem)

```
$ winpmem -f image.img
```

Extracting the relevant contents of applications with Edit controls, such as notepad was a difficult challenge until the introduction of the `editbox` plugin. Based on the research of Adam Bridge, we can now uncover url fields, undo buffers, and undo text entered in the Run dialogue box.

```
$ vol.py -f memory.img --profile=profile psinfo -p <pid>
```

Recover Memory-Resident Evidence of Execution: Shimcochemen

By Fred House, Andrew Davis, and Claudia Torenick

The use of shimcochemen artifacts in many investigations has been limited because data is not updated in the registry until the system is shut down. As a winning submission to the 2015 Volatility plugin contest, these researchers authored a parsing plugin that extracts these entries from the Application Compatibility Cache database in memory. For memory. Despite changes over the years and the method of organization of these entries across versions of Windows, shimcochemen has become a plugin in the Rekall Memory Forensic Framework. In this chapter, we discuss how to use Shimcochemen to uncover evidence of execution.

```
$ vol.py -f test.img -g shimcochemen
```

Decompress Win+H HiberFile and Carve Hibernation Slack: Hibernation Recon

Hibernation Recon by Annual Tech

HiberWin by Comas Technologies

Hibernation data can be a wealth of forensic artifacts in investigations of all kinds. We encountered a hurdle to our analysis when Windows 8 introduced the LZ Huffman compression method for storing the contents of physical memory for a hibernating machine. Our tools at the time could not decompress, let alone produce data from this dataset. As a result, we decided to develop a tool that could unpack the `HDF` to Windows.

```
$ vol.py -f test.img --profile=winpmem HDF
```

Recover Text from Windows Edit Controls

editbox by Adam Bridge

Extracting the relevant contents of applications with Edit controls, such as notepad was a difficult challenge until the introduction of the `editbox` plugin. Based on the research of Adam Bridge, we can now uncover url fields, undo buffers, and undo text entered in the Run dialogue box.

```
$ vol.py -f memory.img --editbox=winpmem
```

Identify Known Malware Based on Import API Fuzzy Hashing: impfuzzy

impfuzzy by IPR/TECCT

Signatures for malicious binaries extracted from the file system are not applicable to memory analysis, due to changes that occur when a PE file is loaded into memory. By using fuzzy hash of the Import API table, performed by `impfuzzy`, we can identify the presence of previously signed malicious files in new memory samples.

```
$ vol.py -f memory.img --profile=winpmem impfuzzy
```

Comprehensive Process and VAD Analysis

paideia by Monnappa K A

Often during memory analysis, an examiner will enumerate processes multiple ways in order to glean insights into its functions and characteristics. Instead of requiring multiple runs of different plugins, paideia cross-references Windows and VAD analysis in one.

```
$ vol.py -f memory.img --profile=winpmem paideia
```

Counters to Memory Forensics: Modern Anti-Analysis Techniques

Subverting Memory Acquisition

Dementia by Luke Wills

In a recent article, “Dementia” research was presented by Luke Wills at the 29th Chaos Communication Congress in December 2015. The tool, Dementia, creates memory capture by intercepting Win32/16 calls through the use of online filtering and a full system(sys Ministério. The hallmark of a malware acquisition tool is manipulated so that any reference to the target process and its kernel objects is removed and the resultant memory image file has no evidence of this running process.

```
$ vol.py -f test.img --profile=winpmem psinfo -e <pid>
```

Anti-Analysis: Spinning the Wheels of the Forensic Examiner

Armageddon by Fred House

Broader and memory analysis PVC 04:10 (Movement Disks Disordered), written by John Williams. This tool can perform PSINFO, TCP endpoint, and FILE OBJECT structures in memory that load the sauerkraut driven shell code. Other files may appear to be loaded into system memory or on other networks.

```
$ vol.py -f memory.img --profile=winpmem opting -p <pid>
```

Evasive of Malicious Code Detection Techniques

Garpyte by Josh Comer

One of the methods we use to identify code Tmatics (two lines below) is to look for memorable memory that is not mapped to disk. Garpyte implements a signature of a code Tmatics (two lines below) and detects the code resident in memory. The use of Shimcache artifacts in many investigations has been limited because data is not updated in the registry until the system is shut down. As a winning submission to the 2015 Volatility plugin contest, these researchers authored a parsing plugin that extracts these entries from the Application Compatibility Cache database in memory. Despite changes over the years and the method of organization of these entries across versions of Windows, Shimcache has become a plugin in the Rekall Memory Forensic Framework. In this chapter, we discuss how to use Shimcache to uncover evidence of execution.

```
$ vol.py -f test.img -g Shimcache
```

Recovery Memory-Resident Evidence of Execution: Shimcochemen

Shimcochemen is intended to expose the contents of physical memory for a hibernating machine. Our tools at the time could not decompress, let alone produce data from this dataset. As a result, we decided to develop a tool that could unpack the HDF to Windows. In this chapter, we discuss how to use Shimcochemen to recover evidence of execution.

```
$ vol.py -f test.img --profile=winpmem HDF
```

Recover Text from Windows Edit Controls

editbox by Adam Bridge

Extracting the relevant contents of applications with Edit controls, such as notepad was a difficult challenge until the introduction of the `editbox` plugin. Based on the research of Adam Bridge, we can now uncover url fields, undo buffers, and undo text entered in the Run dialogue box.

```
$ vol.py -f memory.img --editbox=winpmem
```

Identify Known Malware Based on Import API Fuzzy Hashing: impfuzzy

impfuzzy by IPR/TECCT

Signatures for malicious binaries extracted from the file system are not applicable to memory analysis, due to changes that occur when a PE file is loaded into memory. By using fuzzy hash of the Import API table, performed by `impfuzzy`, we can identify the presence of previously signed malicious files in new memory samples.

```
$ vol.py -f memory.img --profile=winpmem impfuzzy
```

Comprehensive Process and VAD Analysis

paideia by Monnappa K A

Often during memory analysis, an examiner will enumerate processes multiple ways in order to glean insights into its functions and characteristics. Instead of requiring multiple runs of different plugins, paideia cross-references Windows and VAD analysis in one.

```
$ vol.py -f memory.img --profile=winpmem paideia
```

Memory Forensics Analysis Poster

Memory analysis is the decisive victory on the battlefield between offense and defense, giving the upper hand to incident responders by exposing injection and hooking techniques that would otherwise remain undetected.

Memory Analysis will prepare your examiner to:
- Discover zero-day malware
- Detect compromises
- Uncover evidence that others miss

Windows Memory Forensic Analysis

Operating System & In-depth

Incident Response & Threat Hunting

Smartphone Forensic Analysis for In-Depth"
Security Protections

Kernel Patch Protection (aka PatchGuard)
Windows 10 has expanded a functionality called Kernel Patch Protection (sometimes referred to as PatchGuard). KPP checks key system structures, including (but not limited to) the doubly-linked lists that track most objects on Windows. In particular, KPP makes the _EPROCESS structure of an unloaded process from the process list double-byte. When KPP detects an unauthorized modification, it causes a BSOD to halt the system. As a result, Windows kernel-mode modules now use kernel callbacks, Automatic Procedure Calls (APC), and Deferred Procedure Calls (DPC) to call code instead of this old “launch a process and use DWindows to hide it” technique.

Kernel Object Obliteration
Just as we do in memory forensics, many modules can now be found by searching key operating system structures. As of Windows 8, the KEK continues to prevent modules from easily locating it. This does not impact operations since the KEK is not used during normal system operation. If the system crashes, the KernelBase object decrypts the KEK before storing the crash dump data in the page. If the memory forensics analyst wishes to re-encrypt the KEK, the kernel object header is also encrypted when the system crashes. If the kernel is booted without it, this also has the effect of inhibiting some scanning plugins.

FOR526: Memory Forensics In-Depth

AUTHORS:
Alissa Torres @sa_bertor
Jake Williams @maloayload

In today’s enterprise investigations, memory forensics plays a crucial role in unraveling the details of what happened on the system. Recent large-scale malware infections have involved attackers implementing advanced anti-analysis techniques, making the system memory the battleground between offense and defense. Skilled incident responders use memory forensics skills to reveal “ground truth” of malicious activity and move more swiftly to remediation.

Learn more about FOR526: Memory Forensics In-Depth at www.sans.org/FOR526

1) PsLoadedModuleList
The PsLoadedModuleList structure of the KDSG points to the list of loaded kernel modules (often referred to as the LKD). Many malicious modules mimic loaded modules because they require low-level access to the system. Routines, packet sniffers, and other keyloggers are typically loaded to the standard module list. The members of a list are _LDR_DATA_TABLE_ENTRY structures. Standard, Blue, Black, Blue, Flame, etc., have used variants of kernel memory module component—this is a good place to look for advanced (supposed) nation-state malware. Notice that many malware modules can actually be found inside the list, so scanning for structures may also be necessary.

REKALL PLUGINS: modules, evilware

2) Unloaded Modules
The Windows OS always has a list of currently unloaded kernel modules (a.k.a. drivers). This is useful for finding modules (and misbehaving legitimate device drivers).

REKALL PLUGINS: evilware, unloaded modules

3) VAD
VADs (Virtual Address Descriptors) are used by the memory manager to track all memory allocated by a process. Malware and other advanced techniques often use VADs, but hiding from the memory manager is unwise. If it can’t see your memory, it will give it away! (Though most malware does not hide from the memory manager, it could be used to help detect it.)

REKALL PLUGINS: extra memory

4) _EPROCESS
_EPROCESS is perhaps the most important structure in memory forensics. The _EPROCESS structure has a much larger memory footprint than Process Environment Block (PEB). For more information, see the SANS REKALL PLUGINS: vad, vaddump

malware. However, note that some malware has the ability to unlink itself from this list, so scanning for structures may also be necessary.

REKALL PLUGINS: modules, evilware

5) Process Environment Block (aka PEB)
Process Environment Block (PEB) is a data structure that provides a complete view of processes and threads. It contains all important information about a process. For a process in Windows to use any resource (registry key, file, directory, process, etc.), it must first have a handle to that object. We can talk a bit about a process just by looking at its open handles. For instance, you could understand all the types of things that a file handle is opening or privileges that it’s using by examining handles.

REKALL PLUGINS: handles, object types

7) ThreadListHead
Where are the thread list structures in the process? Sorry, we just don’t have room to do them justice here. Threading is still important, though. In Windows, a process is a thread the execution of the Windows kernel must be executed within, and each thread has its own stack and program counter. The Windows kernel provides thread management services, including scheduling and thread creation.

REKALL PLUGINS: threads

8) _LDR_DATA_TABLE_ENTRY
This structure contains all structures of a loaded module. Unloaded modules can be two types: the kernel modules (aka device drivers) and dynamic link libraries (DLLs), which are loaded into user mode processes.

REKALL PLUGINS: modules, evilware

9) PE Loader Data
This structure contains pointers to three linked lists of loaded modules in a given process. Each list is ordered differently (order of loading, order of initialization, and order of memory addresses). Sometimes evidence will expose a DLL into the Windows memory, but then try to hide. Malware can do better than all three lists, or it will exit without its true

REKALL PLUGINS: evilware

Note that many internal OS structures are doubly-linked lists. The pointers in the lists actually point to the pointer in the next structure. However, for clarity of illustration, we have chosen to depict the type of structure they point to. Also, note that the PsActiveProcessHead member of the KDSG structure points to AllProcessesModules member of the _EPROCESS structure. For clarity, we thereby point the pointer to the base of the _EPROCESS structure. We feel that this depiction illustrates this more clearly.