Lecture 14
Memory Forensics

Joshua Reynolds
Spring 2024
NMSU RE

Memory Forensics

Examining infected system memory to learn how malware operates.

- Goals:
  - Find the attacker code for static analysis
  - Find open files, network connections, etc.
  - Identify indicators of compromise and C2
  - Learn how infections started

Dumping Memory

The first step in memory forensics is getting a dump of the RAM.

If you are in a VM, some hypervisors let you simply freeze the VM and the RAM will be saved to disk automatically, because it will be restored when you “resume” the VM

For bare-metal machines or VMs in some hypervisors, you need a **Memory Acquisition** tool to copy RAM to disk.

On servers with large amounts of RAM (think 1TB+) you may need to compress this dump to take it in a timely manner.

Enumerating Windows Processes

The way standard APIs and tools like Task Manager list processes is by using API's to ask the OS for process info.

These API's give access to an internal circularly linked list of process structs called _EPROCESS.

Memory forensics tools can find this data structure, and enumerate running processes.

Even without a running VM, you can see what processes were running when the memory snapshot was taken

Malware Process Hiding with DKOM

Malware doesn’t want to be listed in the _EPROCESS data structure.

Some rootkits (using their power to affect kernel-space memory) alters the linked list to skip past their process.

This is called **Direct Kernel Object Manipulation** (DKOM)

Ultimate futility of full hiding from memory forensics

There are other ways to find hidden processes.

Ultimately, if a process wants to be scheduled on the CPU and allocate memory, it needs to be known to the **OS scheduler** and **memory management** system

They will also have a **virtual address space** with associated **stack** and **heap.**

On systems with DEP, the OS also keeps track of **memory protections**

Where is it theoretically possible to hide from memory forensics?

- A CPU core not controlled by the scheduler, that only uses the cache
- GPU firmware that only uses VRAM
- SSD/HDD controller firmware
- Motherboard firmware (Secure Boot and UEFI protect somewhat)
- Network card controllers
- USB-C devices
- SFPs
- RGB/Fan/Pump controller
- Coprocessors
- ASICs/FPGAs
    - Application-Specific Integrated Circuit
    - Field-Programmable Gate Arrays

Beyond the _EPROCESS structure: Pool Tag Scanning

In Windows, every process, open file, thread, socket, etc. is an **object** that is kept track of by the **object manager** in the kernel.

Each process also needs to ask for and use memory from the **kernel pool** of available managed memory. (This may be **paged** or **non paged**)

- Generally, the kernel and its drivers use non paged memory
- Generally, user-space processes are given paged memory

Looking for process objects, forensics tools can recover process objects and where there memory is. In many cases, info about terminated processes is still available

Volatility’s psxview

Some malware will even break it’s own memory pool tags to hide.

Volatility has a tool with 7 different methods of identifying processes.

Any process that is not seen by all 7 methods is probably trying to hide and deserves further analysis.

In this way, with the right tools, the effort to hide becomes a useful clue to the reverse engineer.

Process Relationships

Processes started by another process have their parent process tracked. Using this linked list, memory forensics can help reconstruct how an attack worked:

→ Explorer.exe

→ Excell.exe

→ PS.exe

→ windows.exe

What does it look like happened here?

Open Handles

The OS keeps track of open handles to files, sockets, ect.

Listing these can help show what a process was doing.

Does it have an open TCP connection with some IP address?

Does it have on open TCP or UDP listener? What port?

What files does it have open? For reading or writing?

What registry keys does it have open?

Imported DLLs

All the imported DLLs into a process can be shown.

This would list any DLL loaded with LoadLibraryA, as well as its file location.

But how, if the handles to these libraries are closed after they are loaded?

It does this by reading the **Process Environment Blocks** that journal this and other information (in user space)

Hidden DLLs

Malware can alter the **PEBs** to hide the loading of a malicious DLL.

But, they cannot as easily hide the traces of this load in the kernel memory manager, which maintains a list of “**Virtual Address Descriptors**”

Volatility has a plugin called ldrmodules that compares what it sees in the three PEBs with the VAD list and finds hidden DLLs

Handy Registry Keys

userassist – contains programs run recently and when

shimcache – similar, but not limited to a user

shellbags – records of directory traversal

“Services” (like Linux daemons)

Services are tracked separately from processes, and so must be looked for specifically.

Volatility can tell you where the .EXE or .DLL was that launched the service.

Recovering recent commands

Volatility has tools to recover recent commands issued in cmd.exe, conhost.exe

Powershell stores a command history in text files at:

%userprofile%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline

Dumping a single process

Once other tools have found where a process's memory resides in actual memory (not virtual address space), you can carve out that process's memory by specifying at what physical address offset to start.

Some tools will automate this, giving you a list of available DLLs for a process, or allowing you to dump by giving a PID.

Detecting Process Injection

Process injection of various types leaves discrepancies in the VAD, PEBs, or both.

Hollow Process Injection: Check .exe location, memory protections, hidden DLLs, and parent processes

API Hooks: List all hooks with Volatility “apihooks”

Kernel Space Callbacks, Hooks, Timers, Callbacks:

- System Service Descriptor Table (and its shadow)
- Interrupt Descriptor Table
- Deferred Procedure Calls

Detecting Malicious Drivers

Rootkits generally run as kernel-mode drivers.

Windows tries to stop them by requiring cryptographic code signatures attested by trusted signers.

Windows also attempts to watch its core kernel data structures with PatchGuard to prevent random changes.

Without Secure Boot, or with a compromised UEFI instance, the most powerful attackers can sometimes get an unsigned driver running.

Listing Kernel Modules

Also vulnerable to drivers hiding themselves with DKOM

Also can be found with Pool Tag Scanning

Kernel-Mode Driver IPC

Drivers communicate with user-space components of the rootkit usually using I/O operations.

Just like a network card, hard drive, USB device, ect.