Welcome!

- This hour will focus on analysis of Linux systems
- We will focus on artifacts and analysis related to a compromised system
- This will include a mix of lecture and hands-on exercises
- To end the hour, we will discuss different memory acquisition techniques and approaches for Linux
What is Memory Forensics?

- Memory forensics is the process of acquiring and analyzing physical memory (RAM) in order to find artifacts and evidence.
- Usually performed in conjunction with disk and network forensics (one component of the digital crime scene).
- Often can be performed alone and solely used to solve complex investigations.

Volatility Framework

- Implemented in Python under the GPLv2.
- Extracts digital artifacts from volatile memory (RAM) samples.
- Extraction techniques are performed completely independent of the system being investigated.
- Offers visibility into the runtime state of the system.
- Over 200 plugins in the 2.4 release.
- Supports analysis of Windows, Linux (+ Android), and OS X samples.
Scenario

- A Linux server was compromised by a remote attacker
- The attacker gained access and installed information-stealing malware on the system
- After detecting suspicious network activity from the compromised system, your client acquired a sample of memory from the system
- You were then given the resulting memory sample and asked to investigate for signs of suspicious activity

The Malware to Investigate

- Contains userland (process) and kernel components
- Userland component injects code to steal user login credentials
  - Credentials are exfiltrated over the network
  - The kernel mode components hide activity related to the userland components
To start the investigation, we should look for the injected code responsible for the credential gathering.

To accomplish this, we need to know two things:

1. What does injected code look like in memory?
2. How do we find it with Volatility?
Process Analysis – Injected Code

• Three methods of code injection:
  1. Shellcode injection
  2. On-disk library injection
  3. Memory-only library injection

• Two plugins to find them:
  • `linux_malfind` (methods #1 and #3)
  • `linux_proc_maps` (method #2)

Process Analysis – Process Listing

• To list processes on a Linux system, you can use the `linux_pslst` plugin
• This plugin walks the active lists of processes kept within the kernel
• This lists can be manipulated by malware
  • Use `linux_psxview` to find unlinked/hidden processes
Process Analysis - Memory Mappings

- To view the memory mappings of each process, the `linux_proc_maps` plugin is used
- This lists each mapping, along with its path, permissions, starting and ending address, and other metadata
- Libraries injected from disk using the system APIs will appear in the output of this plugin
  - Along with all the legitimate libraries
  - This can be an overwhelming amount of data without a whitelist

Process Analysis - `linux_malfind`

- The `linux_malfind` plugin attempts to automate detection of injected code
- Looks for the following anomalies:
  - Sections mapped rwx (readable, writable, and executable)
  - Sections mapped executable that are not backed by a file
- For each suspicious region the following is listed:
  - The process name and ID
  - The starting and ending virtual address of the region
  - A hex dump of the data at the beginning of the region
Process Analysis - Extracting Memory

• Once a suspicious region is found, we will want to dump it to disk
  • For injected libraries, this should include the entire executable
  • For shellcode, this should include the memory region containing the shellcode
• The `linux_dump_map` plugin will extract particular regions to disk
• The `linux_librarydump` plugin will reconstruct ELF files from the given starting address and address space

Investigating Network Connections

• The malware capabilities list included the ability to automatically exfiltrate credentials
• Through memory forensics, we can examine both the currently active network connections as well as historical ones
  • We will use this to find the data exfiltration traces
• The `linux_netstat` plugin will list currently active connections and map them back to their owning process
• The `linux norscan` plugin will carve through memory looking for historical network connection structures
LAB

- Hands-on - Analyzing the Userland Components

Kernel Analysis
Kernel Analysis – Listing Kernel Modules

• To start the kernel analysis, we need to find regions of code in the kernel
• The best place to start is the kernel module list
• The *linux lsmod* plugin walks the list of active modules and reports each one
  • This mimics the exact behavior of *lsmod* on a live system
• Unfortunately, the malware unlinks its LKM from the list
• The *linux_check_modules* plugin can be used to find LKMs that hide from the module list, but not from /sys/
• The *linux_hidden_modules* plugin can be used to find modules that hide from both of the previous plugins

Kernel Analysis – Hidden Network Connections

• Kernel-level malware can trivially hide network connections from all userland tools
• These tools rely on the accuracy of the /proc subsystem to report accurate data
  • In particular, tools like *netstat* rely on the files found under /proc/net/*
LAB

• Hands-on - Analyzing the Kernel Components

Acquisition Notes

• When you can acquire a VM guest from the host, always take that approach
  • No need to load third-party software
  • No need to enter credentials
  • No chance for detection/cleanup by attackers
• The most reliable software tool for Linux memory acquisition is LiME [1]
  • Open source, GPLv2
  • Loads a kernel module that can dump memory to disk (e.g., attached USB drive) or to the network
  • You must compile a LiME module for each kernel that you want to analyze
Memory Forensics Resources

- Volatility
  - https://github.com/volatilityfoundation/volatility
- LiME
  - https://github.com/504ensicsLabs/LiME
- The Art of Memory Forensics
- Community Documentation
  - https://github.com/volatilityfoundation/volatility/wiki/Volatility-Documentation-Project

Thank You

Andrew Case | Volexity
andrew@dfir.org | @attrc
END SESSION 1

Memory Forensics of Linux and Mac Systems
Part 2 - Mac
Welcome!

- This hour will focus on analysis of Mac systems
- We will focus on artifacts and analysis related to a compromised system
- This will include a mix of lecture and hands-on exercises
- To end the hour, we will discuss different memory acquisition techniques and approaches for Mac

What is Memory Forensics?

- Memory forensics is the process of acquiring and analyzing physical memory (RAM) in order to find artifacts and evidence
- Usually performed in conjunction with disk and network forensics (one component of the digital crime scene)
- Often can be performed alone and solely used to solve complex investigations
Volatility Framework

- Implemented in Python under the GPLv2
- Extracts digital artifacts from volatile memory (RAM) samples
- Extraction techniques are performed completely independent of the system being investigated
- Offers visibility into the runtime state of the system
- Over 200 plugins in the 2.4 release
- Supports analysis of Windows, Linux (+ Android), and OS X samples

Scenario

- An executive’s Mac laptop was compromised
- A keylogger that steals keystrokes from all processes was installed
- After the victim had many accounts, all with different passwords, compromised, IT quarantined his system and acquired a memory sample
- You have been given the memory sample and need to locate the keylogger and recover the logged keystrokes
The Malware to Investigate

- Userland and kernel components
- The userland component runs as a standalone process that abuses OS X APIs to globally log keystrokes
  - The keylogger saves the logged keystrokes to a hidden file
- The kernel component hides the keylogger process and save file

---

Userland Analysis
Hidden Process Analysis – Approach

- To start the investigation, we should enumerate processes to attempt to find the hidden one
- Since the process is hidden from the live system, only memory forensics will be able to find it

Hidden Process Analysis – Volatility

- The *mac_pslist* plugin enumerate processes using the method the live system does
  - This means the plugin will not report the process if malware has unlinked it from the process list
  - If malware simply hooks the process enumeration APIs, then it will still see it
- *mac_tasks, mac_pgrp_hash_table, mac_pid_hash_table* enumerate processes without relying on the standard process list
  - The discrepancy between these plugins and *mac_pslist* can immediately point to malware
- *mac_psxview* immediately performs this cross comparison
Hidden Process Analysis – Logfile

- Two approaches:
  - If the keylogger keeps its handle to the logfile opened the entire time it is running, then the `mac_lsof` plugin will show the full path to the file.
  - If the keylogger only keep its handle opened to write out the current buffer, then we can extract process memory and look for filenames.
    - Less precise but often still works.

Hidden Process Analysis – File Extraction

- The `mac_list_files` plugin can enumerate all cached files and recover their contents.
  - In the most ideal situations.
- Caveats:
  - If a file hasn’t been used recently (days? weeks? months?) it may no longer be cached.
  - If certain portions of a file have not been read or written to recently then the contents may not be in main memory.
  - In both cases Volatility will either not be able to recover the file at all or will only be able to recover certain portions.
LAB

- Hands-on - Analyzing the Userland Component

Kernel Analysis
Kernel Analysis – Listing Kernel Modules

- To start the kernel analysis, we need to find regions of code in the kernel
- The best place to start is the kernel extension list
- The `mac_lsmod` plugin walks the list of active kernel extensions and reports each one
  - This mimics the exact behavior of `kextstat` on a live system
- Unfortunately, the malware unlinks its extension from the list
  - The same behavior that is often seen from Windows and Linux malware as well
- `mac_lsmod_iokit` & `mac_lsmod_kext_map` enumerate extensions without relying on the list

Kernel Analysis – Hidden Files

- We already found how the process was hidden, now we need to determine how the file was hidden
- A common method for this is system call table hooking
  - Each time a process wants to interact with the file system or the network it must do so through a system call
- The `mac_check_syscall` & `mac_check_trap_table` plugins enumerate each system call table handler and reports if they are malicious or benign
LAB

• Hands-on - Analyzing the Kernel Component

Acquisition Notes

• When you can acquire a VM guest from the host, always take that approach
• Which tool to use on OS X is very version dependent
  • Mac Memory Reader was the standard tool until 10.7
  • Since then no stable, open source tool has been developed for OS X acquisition
  • MacAquistion from Black Bag provides stable support until 10.11 (El Capitan)
  • No current tool, whether commercial or open source, fully supports El Capitan
Memory Forensics Resources

- Volatility
  - https://github.com/volatilityfoundation/volatility
- The Art of Memory Forensics
- Community Documentation
  - https://github.com/volatilityfoundation/volatility/wiki/Volatility-Documentation-Project

Thank You

Andrew Case | Volexity
andrew@dfir.org | @attrc