Arthur Liou
CS373

Week 5 Writeup

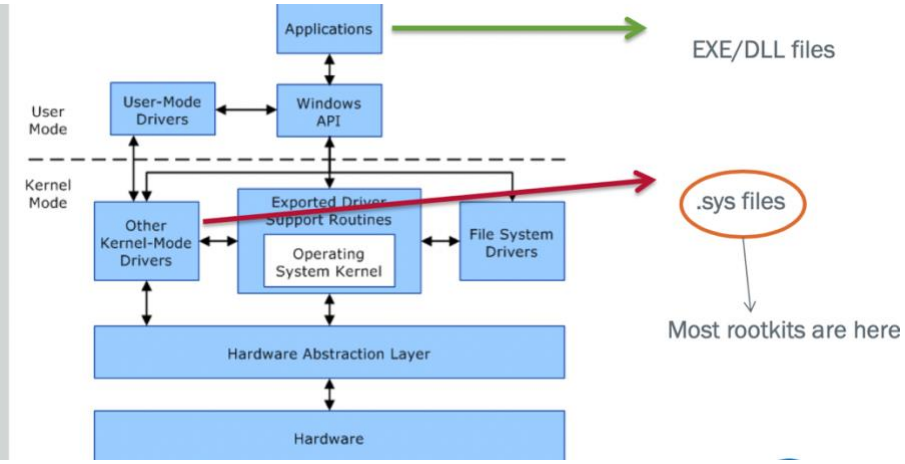
Prompt: Submitting a write-up of your thoughts, impressions, and any conclusions based on the material from the week. Each week will have its own assignment in the grades page.

For this week's writeup, I'm reflecting on the topic – Memory Manipulation. While I found the content to be mildly interesting, I particularly enjoyed the depth of information that we were given and exposed to this week. I really appreciated the diagrams in the lecture slides and all the tidbits of concepts and information that were shared. They really helped solidify my understanding of the material this week. Unfortunately for HW5, I will not be attempting to complete it during this week, due to a project and a midterm due for another OSU CS class. However, I took a glance at the requirements. While it looks straightforward, it has a lot of leeway and I'm wondering if there is a rubric for it – my thoughts for this week. Thanks for reading. I look forward to come back to next week's material!

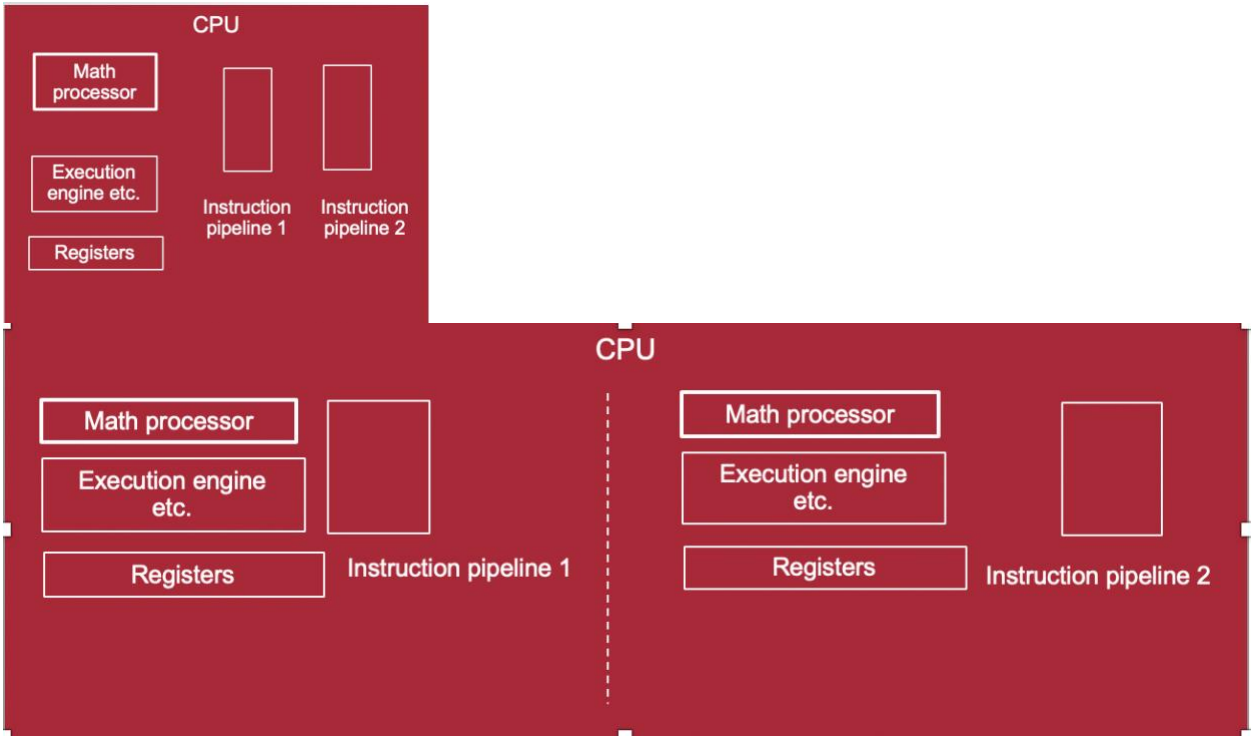
Lecture Notes

Windows Memory Manipulation

- W1-4 Recap = Learned basics of malware analysis, explored PE files and properties, stared at hex code and data, learned how to hunt for malicious traces, memory basics, various stages of attack, reviewed malware persistence, defense, exploit world
- Today: stealth and persistence from malware code
- Rootkits – malware that actively conceals its existence and actions from users and system processes
- ~10% of malware use rootkit. Most prevalent in 32-bit Windows, handful in 64-bit
- Challenges once malicious code enters kernel
    - Harder for rootkits to enter 64 bit kernel
    - Rootkits can infiltrate 64 bit OS Kernal by
        - Bypassing driver signing check (eg. Using testsigning mode)
        - Modifying the windows boot path (MBP) – Secure boot prevents this
        - Kernal exploits in Windows kernel or third party drivers
        - Stealing valid digisigs (similar to Stuxnet)

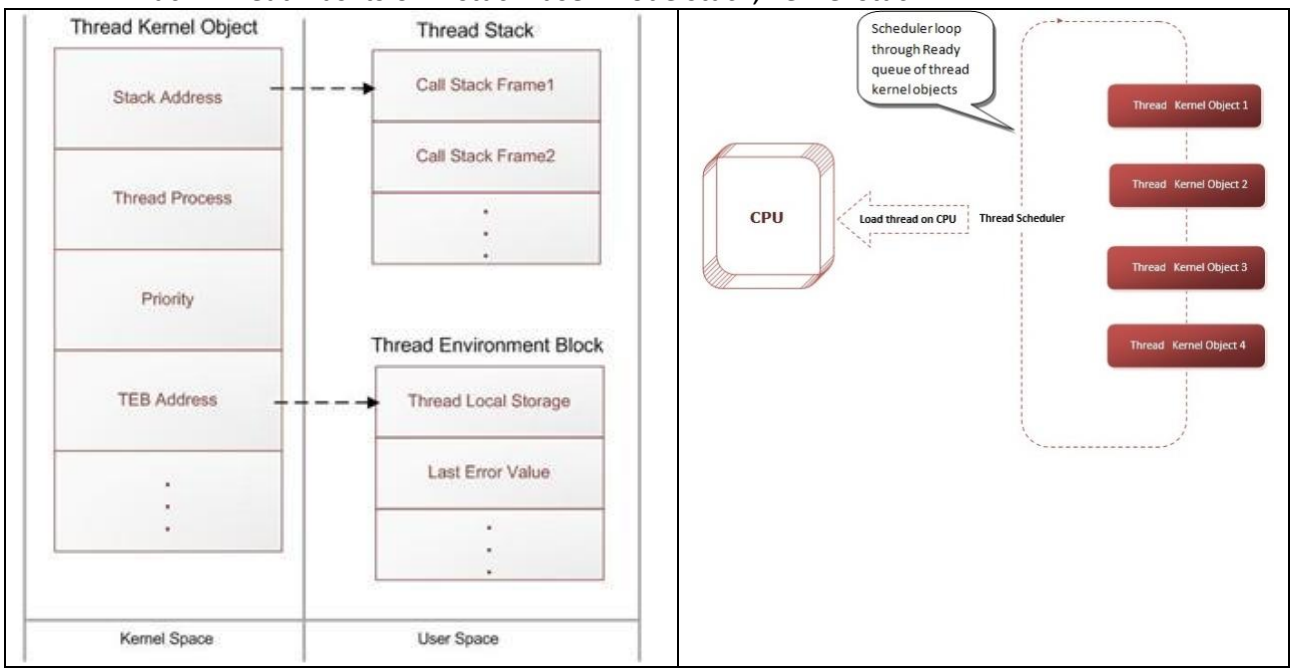


- Kernal Memory is a flat memory model with no security separation
- Any kernel driver can access any part of memory
- Composed of windows kernel (ntoskmi.exe) as well as driver code
- Many important structures that are prime targets for stealth – SSDT, IRP, IDT
- Windbg commands: .process command, lm, !devobi, !drvobj, !devstack, tirp, etc. Virtual to physical memory
- Lab 1 – Agony – Quick & high-level analysis. Cuckoo and memory malipulation
- Thread Basics: HLL-Assembly-Machine Code (HEX)-RAM-BUS-instruction pipeline
- CPU Hyperthreading

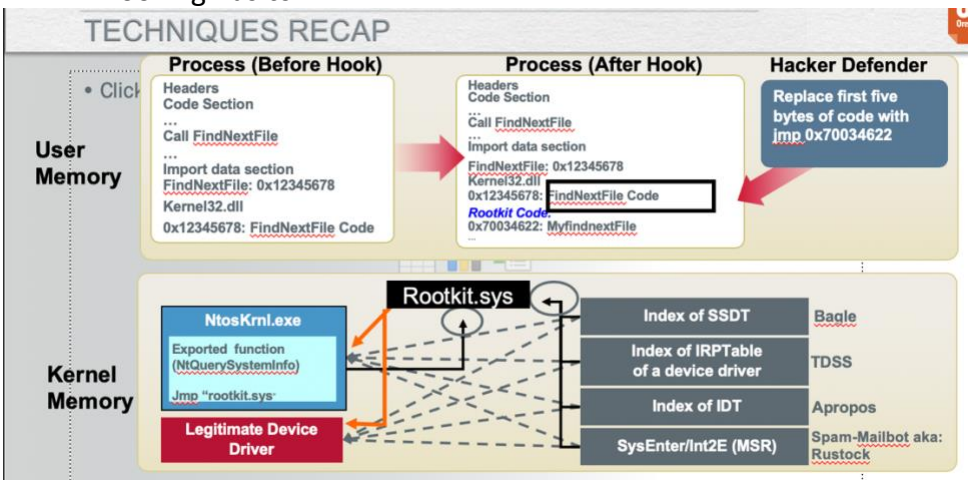


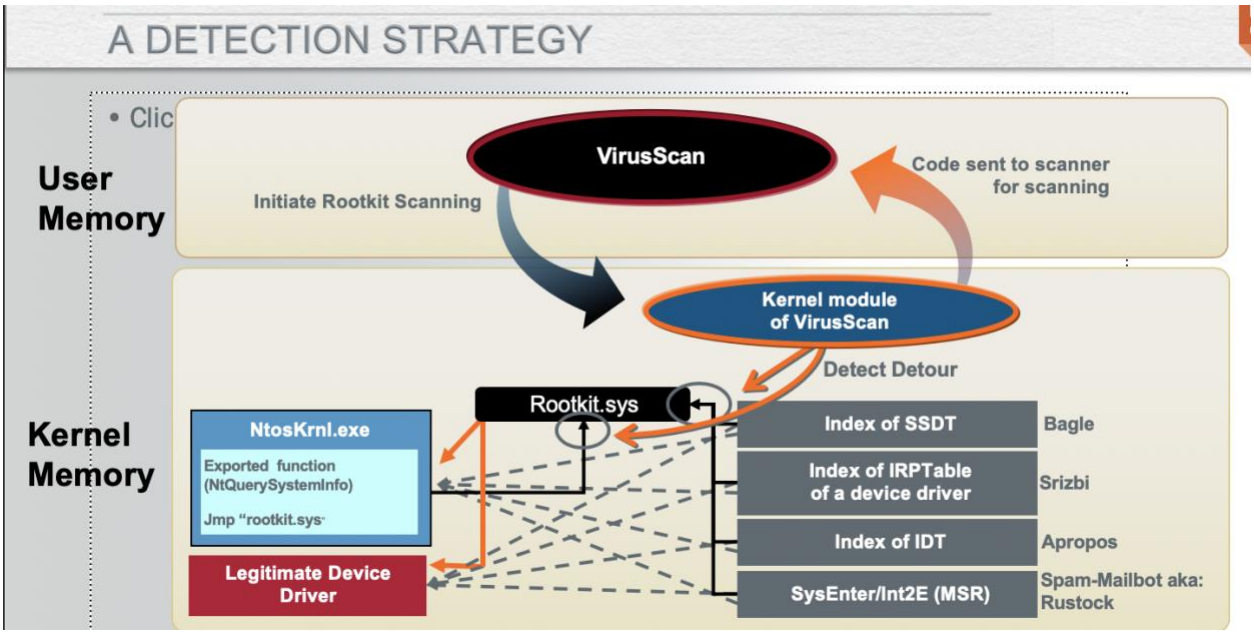
- Threads is the smallest 'unit' of execution that can execute code
- Thread context, stack, environment block (TEB), scheduling, Thread-process relationship
- Thread object defines a thread: Kernal objects are data structure defined by OS to describe various OS constructs, thread being one such construct

- Thead context stores all the related register values of the thread. During execution register values are stored in CPU, else they are in Memory.
- Each Thread has its own stack: user mode stack, kernel stack



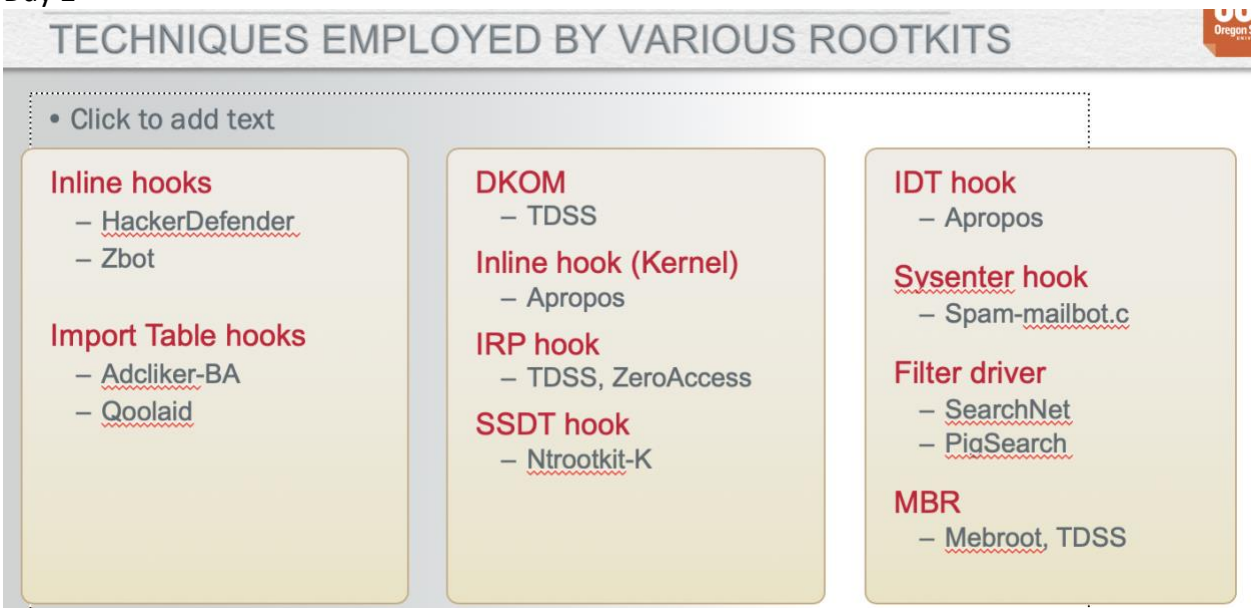
- Process memory: Tool (Process hacker)
- Processes are implemented as objects
- An executable process may contain one or more threads
- Process includes an object table that has handles to other objects known to this process.
- A process needs at least one thread to execute
- Lab 2 – Peering into Process Memory
- Hooking Basics





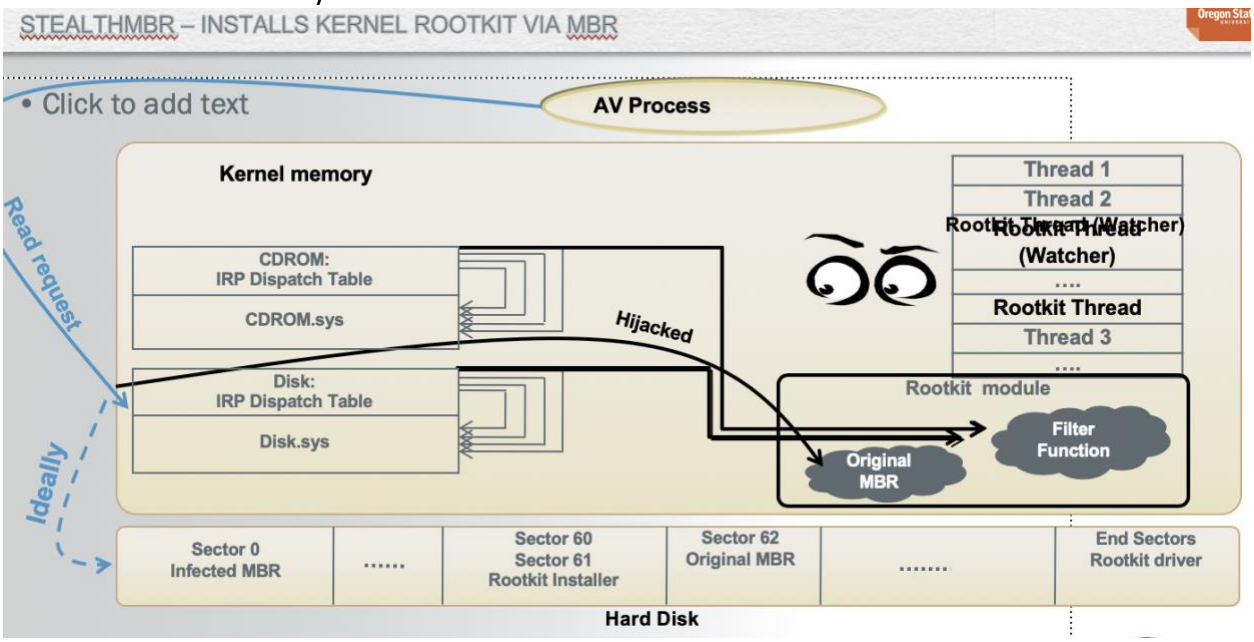
- Lab 3 – Kernal debugging 101
- Cekno Analysis & Detection
- Kernal debugging is more powerful than merely using liveKD
- Finding memory patterns and following the pointers in memory
- Kernal debugging is like any other program debugging you do in your HLL code
- Two machines – one a debugger and another debugee – talking to each other via virtual serial ports

Day 2



- System boot process
- Pre-boot – power on self test to diagnose memory, hardware components, it loads the BIOS that locates the boot devices an dthne loads and runs MBR

- Boot – initial boot loader, OS selection. MBR finds the active partition and load the boot sector in memory to execute it. Boot sector loads the NTLDR from disk



- Lab 4 – Kernal Structure Analysis
- Trends in Stealth Analysis – File forging, memory forging, self protection, attack AV, disassociating memory from file-on-disk, removing dependency on files, untrusting the trusted
- Disassociating memory from file-on-disk. Rootkit's memory can give-away its associated file on disk. Memory scanner needs to know the file. But can be difficult to track kernel memory.
- File Content Foraging – overwrite existing files and forge the 'view; such as the AV gets the clean view instead of the malicious. Better than hiding files
- Removing dependency on files – scanners based on direct file-system (FS) parses worked well. So having no file in the FS helps rootkits, so move malicious code to boot process or BIOs and move encrypted malicious code to raw sectors or as a file. Advantages in 64-bit Windows
- From limited self preservation – defend components and/or attack security components. Watcher threads (monitor and protect memory hooks and disk changes, rewrite registry, files, gain exclusive locks from SYSTEM), and attachs from kernel (callback registration to attack during process, module load)
- Holistic Self Defenses – Behavior ID of AV, untrusting the AV and whitelisting of legitimate applications
- Lab 5 – Analyze banker rootkit

WINDOWS ARCHITECTURE