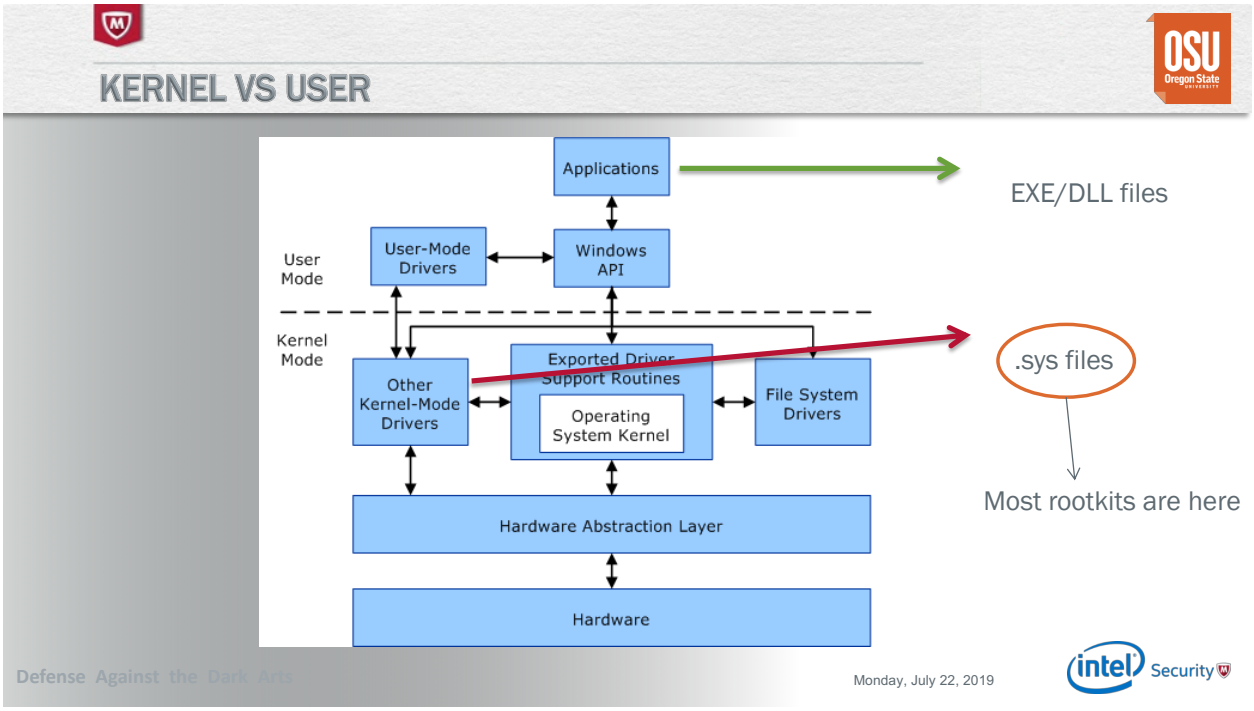
Johnny Zhong
CS373 – Defense Against the Dark Arts
Week 5 – Windows Internals

Focus for the week

- protection of malware IP.
- Understand concepts via memory manipulation.
- Rootkits

Rootkits prominent in 2005.
Used to hide malware on machine.
Use rootkits to learn windows kernel security.
Signatures introduce with 64-bit Windows.



Applications are not writing files to disk; kernel is doing it.
Example -> Typing into a document -> application takes keystrokes from keyboard -> calls Windows APIs -> tells kernel how to respond.

User Layer contains .exe and .dll files. Kernel contains .sys files. All these files are PE files (portable executable files).
Most rootkits like to be in kernel mode (more capability to manipulate the system).
Flat memory model makes it so that malware in kernel mode has the same permissions as kernel applications.
Lab Notes:

Sample – Agony (Rootkit)

Goals:
- Use Cuckoo to determine what files this malware has created.
- What processes are running after execution of the bad sample.

Tools to use:
- LiveKD (download and keep)
- RegShot (system diff tool)

3 folders made:
- Bin files
- Tzero.dll
- Last folder has malware

Other observations:
- File on desktop
- File named sparams

Go to CAnalyzer folder. Use cmd to look for hidden files via "dir *.sys"
In Tools folder, find Tuluka.
Tuluka finds rootkits.
Agony seems to reroute particular requests to the kernel, overriding requests to find it.

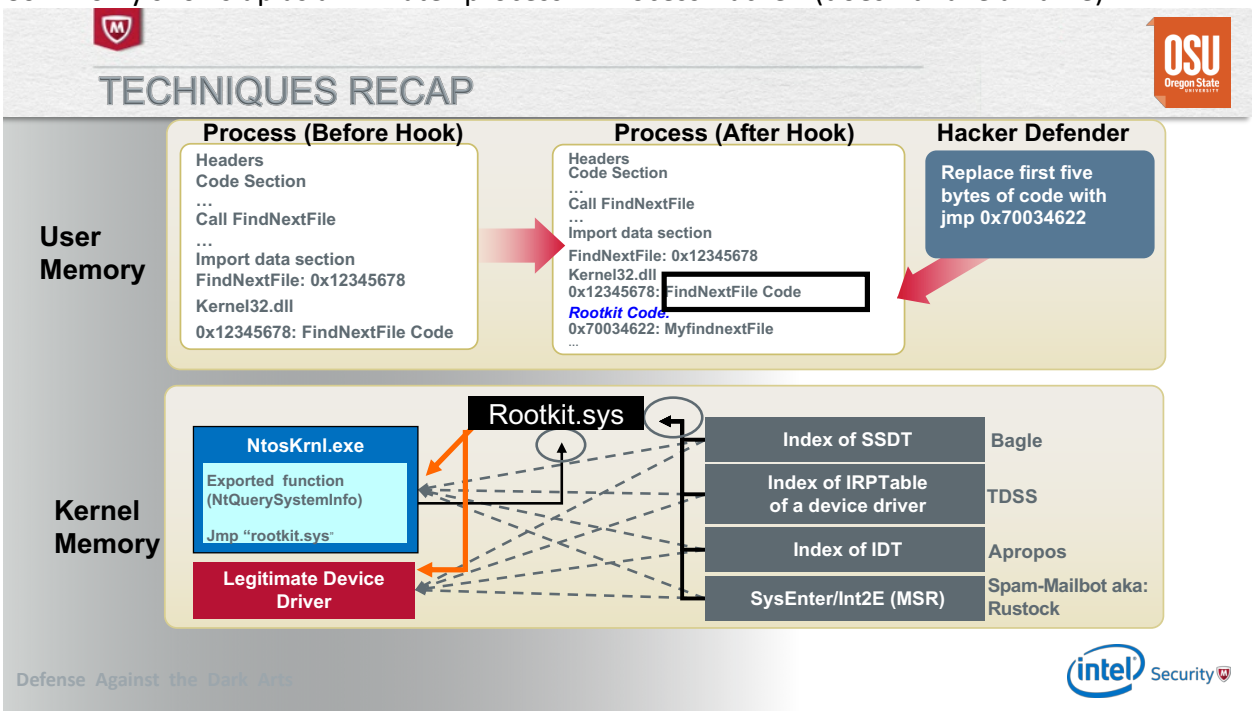
Part 1 Lecture 3

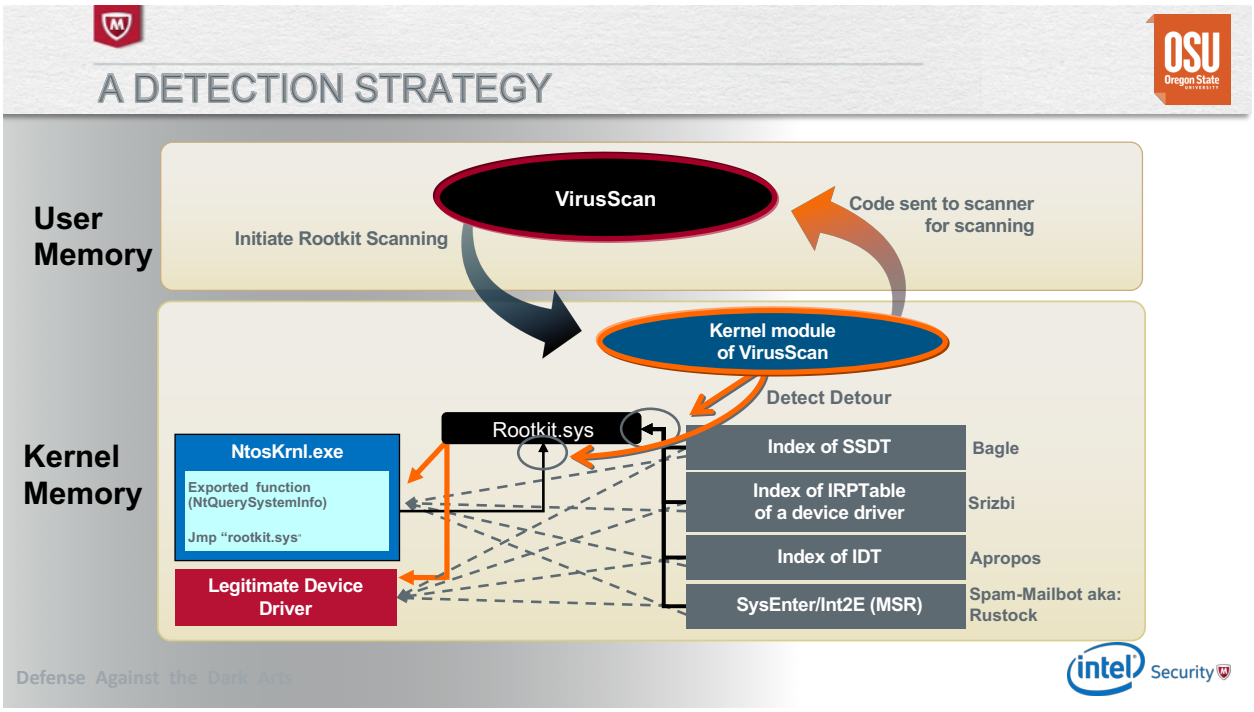
Basics – Threads
Discussion on round robin, thread priority, in the thread scheduler.
Hyperthreading – multiple instruction pipelines in order to manage CPU time, helped manage load for a while.
Multicore Processors – duplicate (or more) performance for multithreaded requests/applications.
Thread context – a saved state of the thread.
DT (command used in WinDBG/LiveKD) – describe/data type, provides information on all things in a thread structure.
Each thread has own stack.

Data is copied into kernel memory from the user memory.
Processes:
Needs at least 1 thread to execute.
Has object table which has handles to threads, files, registry keys that are being accessed.

Lab2 Notes

Exploring Process Memory via Process Hacker
Deeper dive into memory than Process Explorer.
Can walk through virtual memory of processes.
Image Area: meat and potatoes of the process, holds the binary image of the .exe.
Issue with memory: there is no difference between executable code and data. If the block of memory is located within a section that has RX permissions, it can be executed.
Tool can show us memory contents.
Process Injection – code injection from malware that is used to intercept application starting.
Commonly shows up as a "Private" process in Process Hacker. (doesn't have a name)





Detection of rootkits relies on checking on pointers. Determine if the pointers are going outside of the kernel, determine where the pointer is going and look for the malware this way.
Proactive measures use detection logic based reported malware.
Can also detect based on malware behavior.
Faking the memory has been an issue, but can use the hypervisor to view the memory.
Can find the thread as well.

Lab3
Run DPS command, create 2 breakpoints.
This is to be done via the additional Agony Lab Instructions

Lecture Day 2
Terms:

- MBR: Master Boot Record

84 bytes?
Inline Hook – modify function as opposed to the table

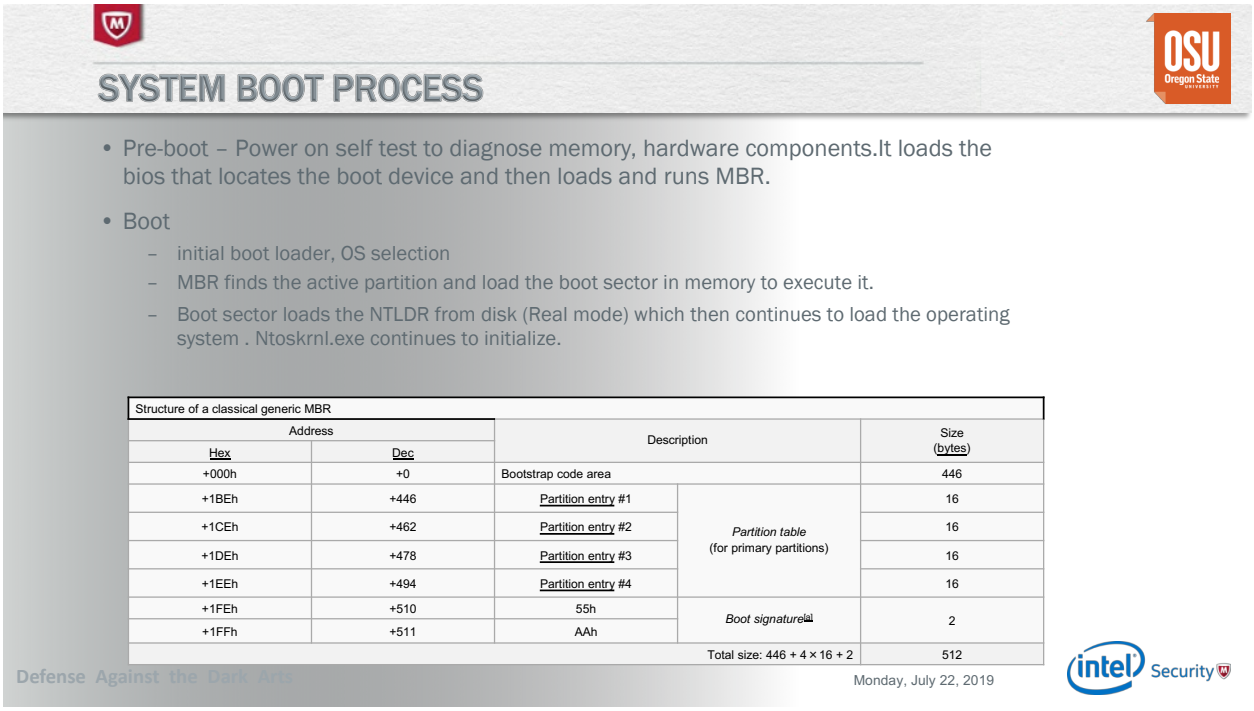
Bootkits

- Began with the Brain virus (propelled AV and virus industry)
- Floppy virus

Ransomware

- Modifies the first sector of your disk
- Triggers a password screen to lock

- Not true encryption



SYSTEM BOOT PROCESS

- Pre-boot – Power on self test to diagnose memory, hardware components.It loads the bios that locates the boot device and then 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 (Real mode) which then continues to load the operating system . Ntoskrnl.exe continues to initialize.

Structure of a classical generic MBR

Address		Description		Size (bytes)
Hex	Dec			
+000h	+0	Bootstrap code area		446
+1BEh	+446	Partition entry #1	*Partition table* (for primary partitions)	16
+1CEh	+462	Partition entry #2		16
+1DEh	+478	Partition entry #3		16
+1EEh	+494	Partition entry #4		16
+1FEh	+510	55h	*Boot signature*[a]	2
+1FFh	+511	AAh		
			Total size: 446 + 4 × 16 + 2	512

Defense Against the Dark Arts

Monday, July 22, 2019

intel Security

System Boot Process

- If bootkit is loaded, it can insert itself into the steps prior to the OS loaded
- Can then affect the OS prior to being loaded
- Secure Boot confirms signatures in order to boot properly

Stealth MBR:

- Sector 0, on first disk new code is written.
- Sector 0-62 on VMWare is empty.
- Sector 61: rootkit creates a small installer blob
- Sector 62: original MBR
- On boot, infected MBR gives control to rootkit installer, which loads rootkit into memory, calls original MBR. Rootkit here is stored in memory.

IRP Dispatch Table Malware (rootkit)

- Similar to ssdd
- All drivers have one
- Can hide rootkit code from disk in this table
- Can redirect requests to read disk sectors, providing fake data or the original MBR
-

TDSS Rootkit Demo

- Look at device object and device stack (what are the drivers that are serving this?)
- Inspect disk device via “dt <address>”
- Sect Editor: sector editing tool
  - Used to read MBR sectors

    - MBR ends with 55 88
    - MBR filled with instructions, can inspect instructions via GDB
    - Export first section (Sector 0)
- Save the exported sector somewhere (sector 0)
- Run DDSS rootkit/bootkit, then diff the current sector 0 vs the saved sector 0
- Modifies MBR, modifies memory to hide that it's modified the MBR
- Look at device object location (738)

Disassociating Memory from File-On-Disk
- Modified API
- Remove API names from loaded modules, can't map module address back to disk

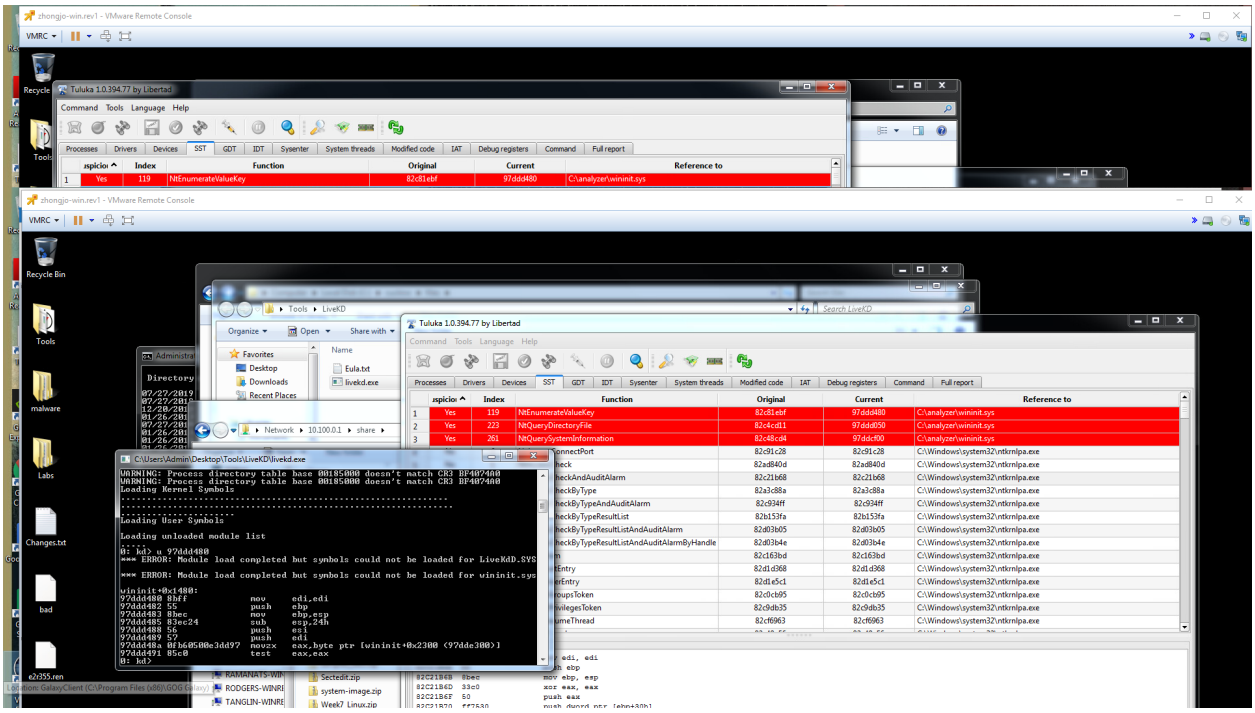
File Content Forging:
- Modify files on disk
- If attempt to read files, malware provides false file contents
- Remove malware by removing thread that provides the fake view

Detection Engine that attacks AV
Whitelisted applications on limited capability machines (ATM/POS)

Brazilian Banker Rootkit
- OS provides callbacks
- Callback notifies those drivers registered for the callback (even when notifying for applications getting ready to run)
- When an application has been loaded into memory, the malware is notified and patches its entry point and returns an error (in cases of AV).
- This allows malware to prevent the starting of AV to avoid detection.

To find agony:
- Write memory to file
- Run yara on file on unique byte sequence
- We know address, so can find byte sequence
- Use lm to find address of winnit.sys and size of winnit.sys
- Or on winnit.sys, locate functions responsible for hooking
- Can use file insight
- Best method is to look for hooking functions
- .writemem <start_address> <end_address>
- lm to to find the range
- Baw1 – break on access write on 1 byte – create a breakpoint that has the program break when the malware attempts to write to a particular location
    - Allows user to get to the rootkit
    - By way of looking for behavior

Extra Lab Work:



The hooks for the kernel APIs were found at the addresses provided in the screenshot provided. The modified values are below:

- NtEnumerateValueKey: 97ddd480
- NtQueryDirectoryFile: 97ddd050
- NtQuerySystemInformation: 97ddcf00

The callbacks to the original APIs were found at the following addresses:

- NtEnumerateValueKey: 97ddd4d4
- NtQueryDirectoryFile: 97ddd086
- NtQuerySystemInformation: 97ddcf1a

The offset of these values are as follows:

- NtEnumerateValueKey: 0x54 -> 84 bytes
- NtQueryDirectoryFile: 0x36 -> 54 bytes
- NtQuerySystemInformation: 0x1a -> 26 bytes