Analyzing Malware in Memory

Andrew Case

@attrc

Who Am I?

- Develop forensics & RE curriculum for the Hacker Academy
- Volatility Core Developer and Registry Decoder Co-developer
- Former Blackhat, SOURCE, and BSides speaker
- Lead-investigator on many large-scale investigations

What is Memory Forensics?

- Memory forensics is the analysis of captures of physical memory (RAM) for artifacts relevant to an investigation
- Requires modeling of the operating system's data structures and algorithms offline in order to recreate state at the time of the capture

Why is it Important?

- Traditional forensics only looks at disk images
- This misses information never written to disk
  - Network connections, memory allocations, running processes, open file lists, and much more
- Skilled attackers know to avoid the disk and securely clean up on-disk artifacts

Why? Cont.

- Malware can trivially default live analysis
  - Live analysis is running tools built into the OS to gather volatile data (the general sysadmin/IR response)
  - Malware can lie to any and all userland tools and even in-kernel monitors
- Advanced malware only operates in memory
  - Never touches the disk, all network traffic encrypted
  - Good luck without memory forensics!

Volatility Overview

Volatility

- Most popular memory analysis framework
  - Written in Python
  - Open Source
  - Supports Windows {XP, Vista, 7, 2003, 2008}
  - Supports Linux on Intel and ARM (Android)
  - Supports OS X
- Allows for analysis plugins to be easily written
- Used daily in real forensics investigations

Volatility Architecture - Vtypes

- Python representation of data structures, unions, enumerations, bitfields, arrays, pointers, etc...
- Accessed as regular Python classes
- Auto generated from OS debugging information (PDB)
- PDBs may be incomplete or not exist at all
  - Reverse engineering
  - Apply overlays

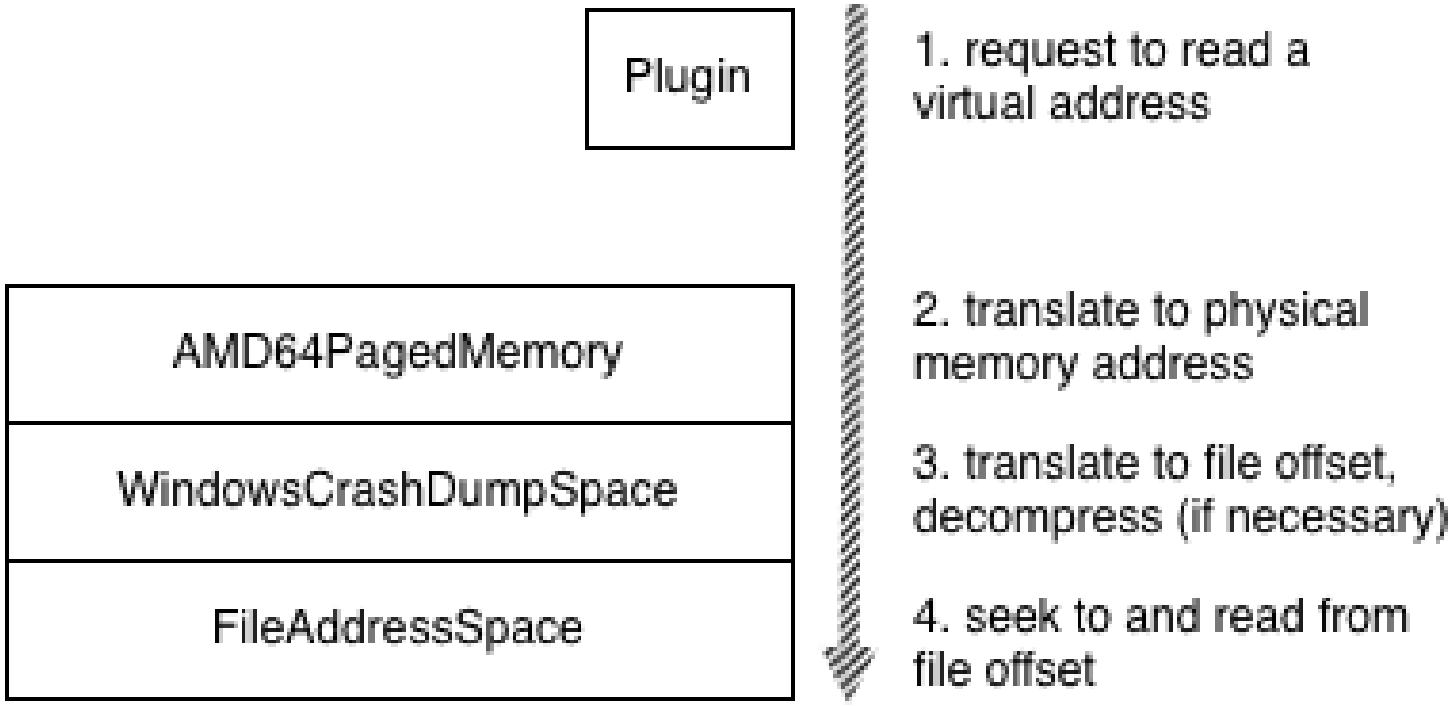
Volatility Architecture - Profiles

- Profiles are collections of VTypes, overlays, and object classes which are customized for a specific OS release
- Meta-data
  - os, major, minor, memory_model

Volatility Architecture - Address spaces

- Address spaces are used to model translations for non-raw files
- Two Types
  1. Memory managament
     - Perform virtual address to physical offset translations
  2. File based
     - Interperts physical offsets requests to reads inside a structured file format

Volatility Architecture - Address spaces



Volatility Architecture - Virtual / Paged Address Spaces

- Intel
  - x86
  - x86_64
  - PAE
  - 4KB/2MB/4MB pages
- ARM (Android)

Volatility Architecture - File Based Address Spaces

- Firewire
- Windows Hibernation Files
- Crash Dumps
- EWF Files
- LiME (Linux/Android)
- Mac Memory Reader

Volatility Architecture - Scanning

- Self-referential values
  - KPCRScan
- Pattern / signature matching
  - KDBGScan, DTBScan
- Object scanning
  - _DISPATCHER_HEADER
  - _POOL_HEADER

Pool Scanning

- Many types of objects that need to be allocated & de-allocated quickly are stored in pools
- Each object allocated from a pool is given a pool header
- This pool header contains a tag per-pool type
- Scanning memory for this tag leads to the recovery of the corresponding objects
- As we will see, pool scanning allows us to recover previously freed objects as well as ones rootkits try to hide

Per-Process Analysis

Processes

- Processes/threads are separate units of execution that have a number of unique attributes:
  - Open file handles
  - Memory mappings
  - Network connections
  - Privileges

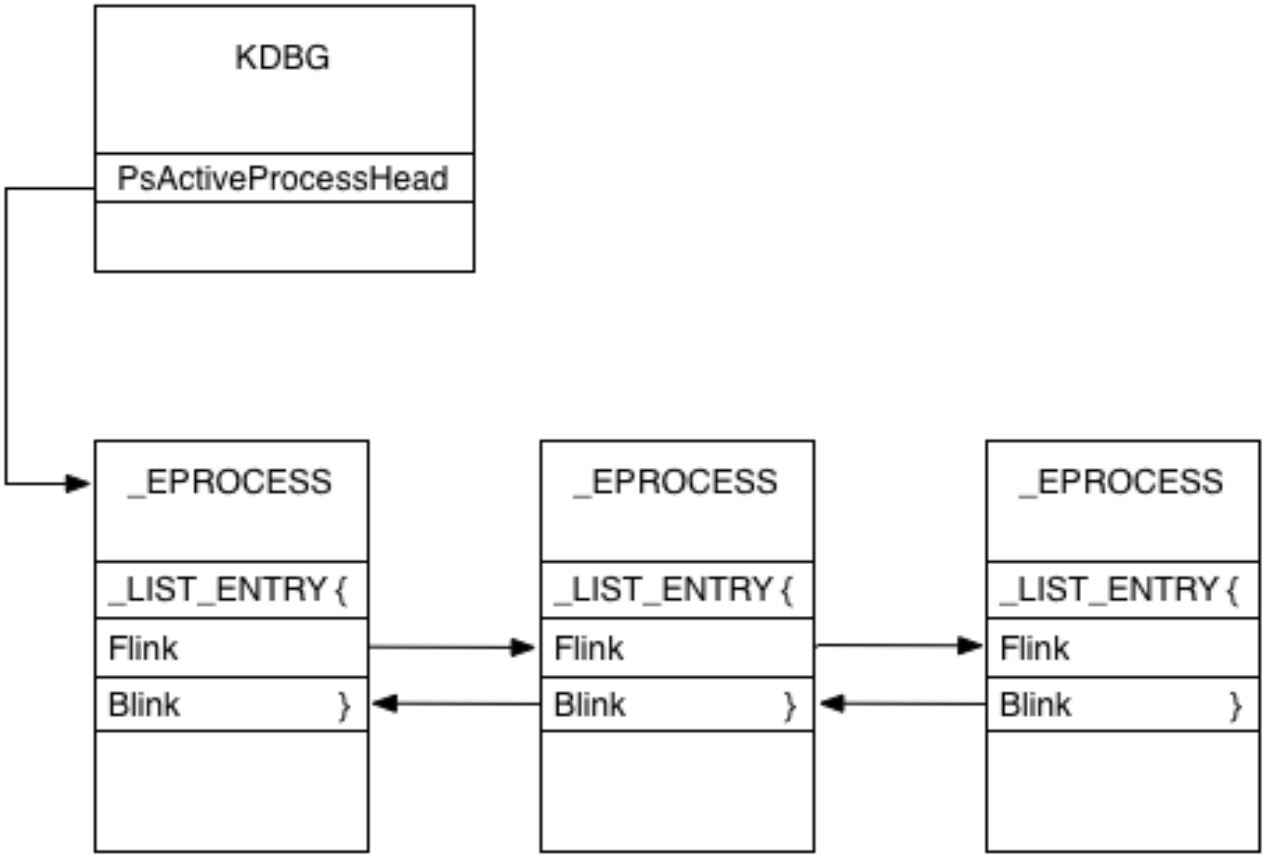
Finding Processes – PsActiveProcessHead

- Walk the doubly linked list pointed to by PsActiveProcessHead
- Implemented in the *pslist* plugin
- Live tools such as task manager, Process Explorer, etc rely on this list through userland APIs
- Malware often targets this list to hide processes

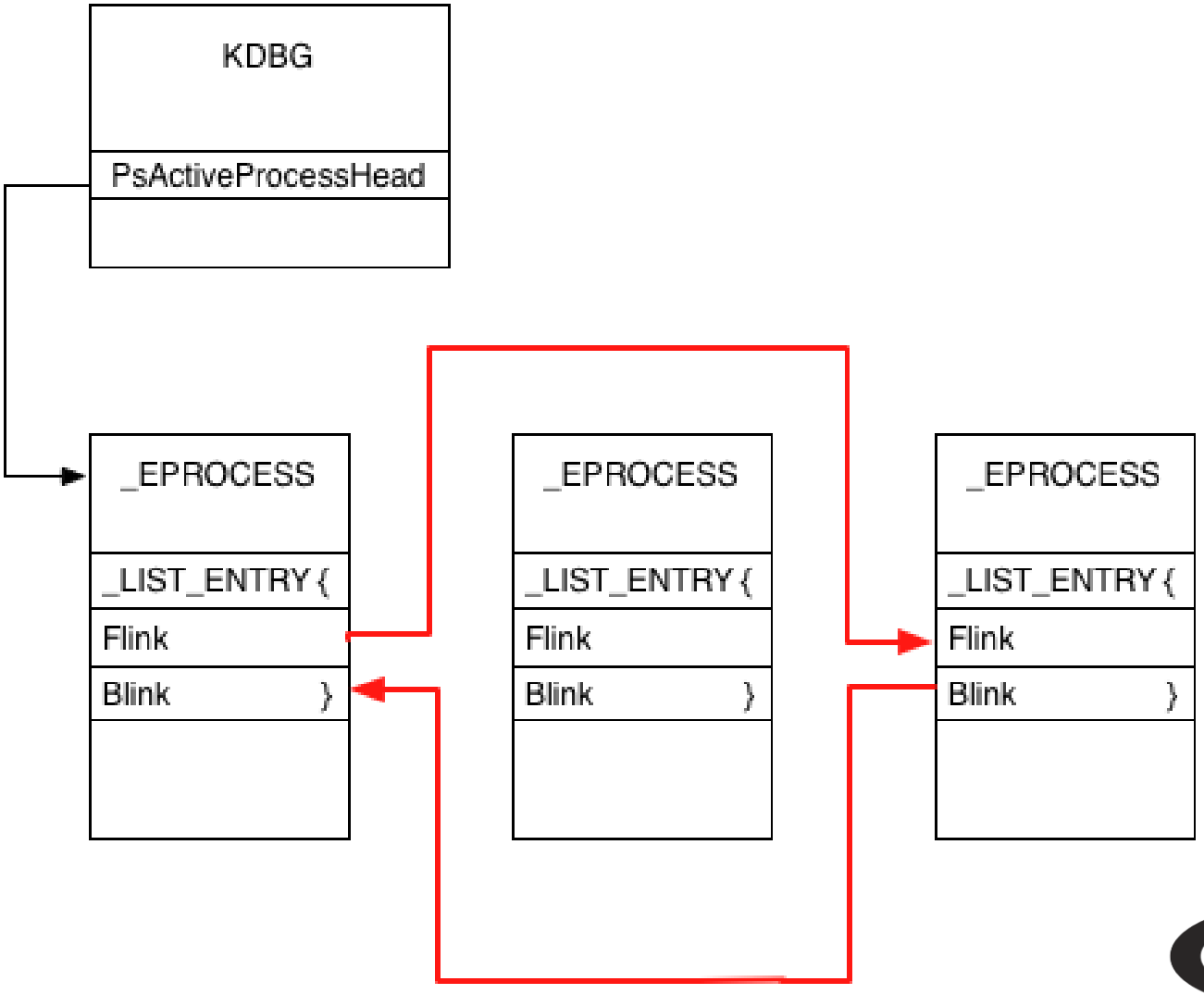
pslist output

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x823c8830	System	4	0	51	269	------	0		
0x8211a020	smss.exe	360	4	3	19	------	0	2012-04-28 01:56:37	
0x82129220	csrss.exe	596	360	11	340	0	0	2012-04-28 01:56:38	
0x82194020	winlogon.exe	624	360	17	535	0	0	2012-04-28 01:56:39	
0x82146460	services.exe	672	624	15	238	0	0	2012-04-28 01:56:39	
0x821497f0	lsass.exe	684	624	26	410	0	0	2012-04-28 01:56:39	
0x821d4500	svchost.exe	852	672	22	203	0	0	2012-04-28 01:56:40	
0x82147da0	svchost.exe	940	672	9	215	0	0	2012-04-28 01:56:41	
0x8211a880	svchost.exe	1024	672	75	1480	0	0	2012-04-28 01:56:41	
0x8217d020	svchost.exe	1072	672	5	82	0	0	2012-04-28 01:56:41	
0x82124020	svchost.exe	1124	672	14	193	0	0	2012-04-28 01:56:42	
0x822b0020	spoolsv.exe	1356	672	11	106	0	0	2012-04-28 01:56:43	
0x8202a020	alg.exe	1880	672	5	102	0	0	2012-04-28 01:56:53	
0x822b7a58	userinit.exe	1212	624	0	--------	0	0	2012-04-28 02:20:54	2012-04-28 02:21:21
0x8214a020	explorer.exe	1096	1212	13	317	0	0	2012-04-28 02:20:54	
0x820211d0	userinit.exe	1836	624	0	--------	0	0	2012-04-28 02:20:55	2012-04-28 02:22:05
0x82222268	reader_sl.exe	2008	1096	2	27	0	0	2012-04-28 02:20:56	
0x821f67e8	AdobeARM.exe	1796	1096	10	215	0	0	2012-04-28 02:20:56	
0x82247da0	cmd.exe	1120	1096	1	33	0	0	2012-04-28 02:21:15	
0x821ab3d0	mdd.exe	1396	1120	1	24	0	0	2012-04-28 02:23:20	

Non-Tampered with Process List



Tampered (DKOM'd) List



Scanning for Processes

- Need to be able to find processes even after DKOM
- The *psscan* plugin leverages the scanning framework to find EPROCESS structures
- Does not rely on the OS'es list so will find discrepancies from *pslist*

Hiding from *psscan*

- POC rootkit hides from *pslist* and modifies pool tag to hide from *psscan*
- The pool tag is non-essential, can be overwritten
- Still need to find it!

psxview

- Enumerates processes seven different ways
  - A rootkit is very unlikely to target all these sources
- Methods besides pslist & psscan:
  1. Pool scanning for threads
  2. Enumerating Process handles from PspCidTable
  3. Csrss maintains handles to processes (objects) while active
  4. enumerate per-session processes
  5. desktop thread scanning

psxview output

```
$ python vol.py -f ds_fuzz_hidden_proc.img psxview
Volatile Systems Volatility Framework 2.3_alpha
Offset(P)  Name                PID pslist psscan thrdproc pspcdid csrss session deskthrd
---------- ---------------- ------ ------ ------ -------- ------- ----- ------- --------
0x01a2b100 winlogon.exe        620 True   True   True     True    True  True    True
0x01a3d360 svchost.exe         932 True   True   True     True    True  True    True
0x018a13c0 VMwareService.e    1756 True   True   True     True    True  True    True
0x018e75e8 spoolsv.exe        1648 True   True   True     True    True  True    True
0x019dbc30 lsass.exe           684 True   True   True     True    True  True    True
0x0184e3a8 wscntfy.exe         560 True   True   True     True    True  True    True
0x018af860 VMwareTray.exe     1896 True   True   True     True    True  True    True
0x01a4bc20 network_listene    1696 False  False  True     True    True  True    True
0x01843b28 wuauclt.exe        1372 True   True   True     True    True  True    True
0x01a59d70 svchost.exe         844 True   True   True     True    True  True    True
0x018af448 VMwareUser.exe     1904 True   True   True     True    True  True    True
0x019f7da0 svchost.exe        1164 True   True   True     True    True  True    True
[snip]
```

Dumping Processes to Disk

- *procmemdump* and *procexedump* dump a process' main executable to disk
- Take the –p flag for processes that appear in pslist
- For hidden processes use –o/--offset with the address from psscan/psxview
- Binary can be reversed as normal
  - Cannot be executed!

VADs

- Virtual address descriptors are used to represent ranges of all virtual memory within a process
  - Start/end address
  - Protection
  - If the range maps a file or not
- Think:
  - Stack
  - Heap
  - Memory-mapped files
  - Executables

Interacting with VADs

- vadinfo
  - display detailed information about each VAD node
- vaddump
  - extract zero-padded VAD memory ranges to individual files
- vadtree
  - show parent, child relationships; useful with Graphviz diagrams

$ python vol.py -f test.vmem -p <PID> vadtree --output=dot --output-file=vad.dot

Scanning Memory with Yara

- Scan a virtual (kernel or process) address space for yara signatures
- Flexibility
  - ANSI or Unicode strings
  - Hex byte code sequences
  - Command-line or rules files

Searching Memory with Strings

- The *strings* plugin can take output of strings commands and map the strings back to their owning processes or kernel memory
  - The next version of *strings* will differentiate between kernel modules and the rest of the kernel
- Can use this to search for IP:
  - IP addresses / domain names
  - Filenames
  - SSN/CCN/PII
  - Passwords
  - Commands entered by an attacker
  - <many more>

DLLs

- The set of loaded DLLs in a process is recorded in three lists inside of a process' PEB
- Legitimate ways loaded:
  - Specified as needed by an executable in its IAT
  - Loaded through LoadLibrary or LdrLoadDll

Examining DLLs with Volatility

- dlllist
  - Lists DLLs in a process
- dlldump
  - Dumps DLLs from all processes by default
  - Can be filtered with --pid for specific processes
  - Can dump from hidden processes with –o
  - Can dump specific DLL/exe with –b
  - Filter on name of DLL/exe with -r

Hiding from dlllist & live tools

- The three DLL lists exists in userland so can be manipulated without kernel code
- Process explorer and other live tools focus on one of the lists (load order)
- Malware commonly breaks this list to avoid detection
  - Flame is a popular example [2]
- *ldrmodules* cross references the three lists with VAD information for mapped files

ldrmodules

$ python vol.py -f flame.raw -p 912 ldrmodules

Volatile Systems Volatility Framework 2.1_alpha

```
Pid      Process            Base       InLoad InInit InMem MappedPath
-------- ------------------ ---------- ------ ------ ----- ----------
   912 services.exe       0x7c900000 True   True   True  \WINDOWS\system32\ntdll.dll
   912 services.exe       0x7c9c0000 False  False  False \WINDOWS\system32\shell32.dll
[snip]
```

Can ldrmodules be defeated?

- VAD information can be manipulated too
  - This will bypass *ldrmodules* and *dlllist*
- MHL found a (buggy) POC that does this
- Wrote *memhole* plugin that compares all pageable addresses in a process with those found by walking the VAD tree
  - Any not found are suspicious

Code Injection

- We have shown how difficult it is to hide processes and DLLs from memory forensics
- We will now discuss four other ways that malware tries to hide injected code and how to detect the attacks
  - Remote library injection
  - Remote shellcode injection
  - Reflective DLL loading
  - Process hollowing

Code Injection - Remote Library Injection

- A malicious process loads a DLL from disk into another process
- Malicious process can now exit and have work performed in another processes' context
- The malicious DLL can be found using the same methods we just discussed

Code Injection - Remote Shellcode Injection

- Instead of loading a full DLL into another process, simply allocate page(s), copy shellcode in, and then have it executed

Code Injection - Reflective DLL Loading

- Loads & executes a DLL into a process without it touching disk
- Integrated into Metasploit

Detecting These Injections

- The *malfind* plugin looks for pages that have suspicious mappings such as being executable and writable
- Lists the address of the suspicious page along with a hexdump & disassembly of its contents
- Injected DLLs can be extracted with *dlldump* and injected shellcode with *vaddump*

malfind & Stuxnet

$ python vol.py -f stuxnet.vmem malfind

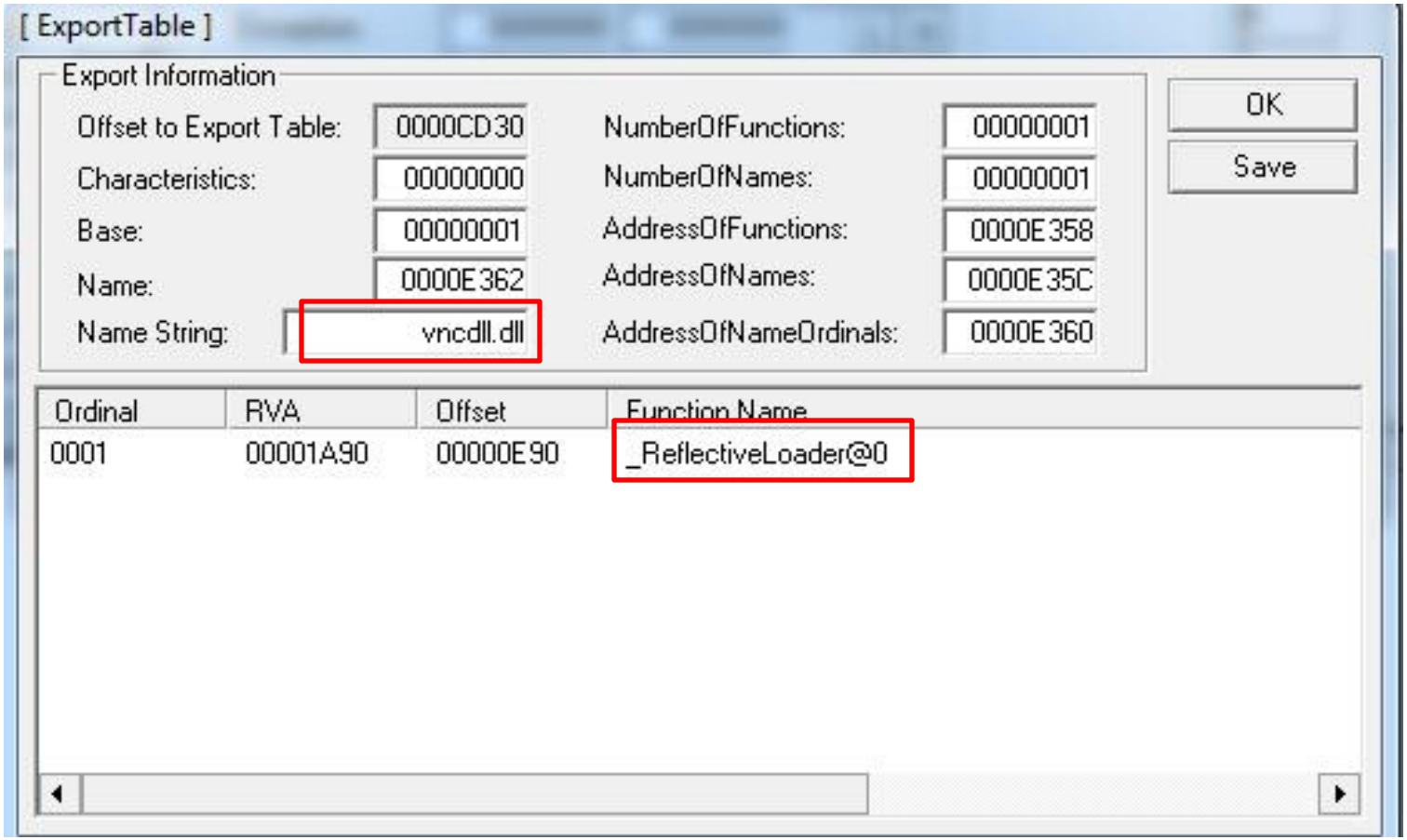
Process: lsass.exe Pid: 868 Address: 0x80000

Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE

Flags: Protection: 6

```
0x00080000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x00080010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x00080020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00080030  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00   ................
```

Analyzing Metasploit's Reflectively Loaded DLL



Code Injection – Process Hollowing

- Loads a thread into a suspended state, overwrites the main executable with malicious one, and then lets the thread begin execution
- To the system administrator, it looks like the overwritten process is actually the one running
- Volatility can be used to detect this by comparing the code in a non-overwritten process with the overwritten one

API Hooking

API Hooks

- Overwrite the beginning of API functions in order to redirect control flow
- Allows the malware to hide virtually any data from userland tools and even some in-kernel monitors

apihooks - Coreflood

```
$ python vol.py -f coreflood.vmem -p 2044 apihooks
Volatile Systems Volatility Framework 2.1_alpha
**************************************************
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 2044 (IEXPLORE.EXE)
Victim module: iexplore.exe (0x400000 - 0x419000)
Function: kernel32.dll!GetProcAddress at 0x7ff82360
Hook address: 0x7ff82360
Hooking module: <unknown>

Disassembly(0):
0x7ff82360 e8fbf5ffff      CALL 0x7ff81960
0x7ff82365 84c0            TEST AL, AL
0x7ff82367 740b            JZ 0x7ff82374
0x7ff82369 8b150054fa7f    MOV EDX, [0x7ffa5400]
0x7ff8236f 8b4250          MOV EAX, [EDX+0x50]
0x7ff82372 ffe0            JMP EAX
0x7ff82374 8b4c2408        MOV ECX, [ESP+0x8]
```

apihooks - Duqu

Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1176 (lsass.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9af000)
Function: ntdll.dll!ZwQuerySection at 0x7c90d8b0
Hook address: 0x980a02
Hooking module: <unknown>

Disassembly(0):
```
0x7c90d8b0 b8020a9800      MOV EAX, 0x980a02
0x7c90d8b5 ffe0            JMP EAX
0x7c90d8b7 03fe            ADD EDI, ESI
0x7c90d8b9 7fff            JG 0x7c90d8ba
0x7c90d8bb 12c2            ADC AL, DL
0x7c90d8bd 1400            ADC AL, 0x0
0x7c90d8bf 90              NOP
0x7c90d8c0 b8a8000000      MOV EAX, 0xa8
0x7c90d8c5 ba              DB 0xba
0x7c90d8c6 0003            ADD [EBX], AL
```

Misc. Process Data

Handles plugin

- Lists the open handles of a process
- Can be filtered by handle type:
  - Process
  - Thread
  - File
  - Key
  - <more>

Network Connections

- Volatility has the ability to list active network connections as well as to scan for previously active ones
- The network connection is mapped back to the PID that started it
  - This PID might not exist anymore

getsids

- Lists the SIDs associated with a particular process
- Lets you see which processes have elevated or unusual privileges

getsids - GrrCon Challenge

```
$ python vol.py -f grrcon.img getsids -p 624
Volatile Systems Volatility Framework 2.1_rc3
winlogon.exe (624): S-1-5-18 (Local System)
winlogon.exe (624): S-1-5-32-544 (Administrators)
winlogon.exe (624): S-1-1-0 (Everyone)
winlogon.exe (624): S-1-5-11 (Authenticated Users)

$ python vol.py -f grrcon.img getsids -p 684
Volatile Systems Volatility Framework 2.1_rc3
lsass.exe (684): S-1-5-18 (Local System)
lsass.exe (684): S-1-5-32-544 (Administrators)
lsass.exe (684): S-1-1-0 (Everyone)
lsass.exe (684): S-1-5-11 (Authenticated Users)

$ python vol.py -f grrcon.img getsids -p 1096
Volatile Systems Volatility Framework 2.1_rc3
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-500 (Administrator)
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-513 (Domain Users)
explorer.exe (1096): S-1-1-0 (Everyone)
explorer.exe (1096): S-1-5-32-545 (Users)
explorer.exe (1096): S-1-5-32-544 (Administrators)
explorer.exe (1096): S-1-5-4 (Interactive)
explorer.exe (1096): S-1-5-11 (Authenticated Users)
explorer.exe (1096): S-1-5-5-0-206541 (Logon Session)
explorer.exe (1096): S-1-2-0 (Local (Users with the ability to log in locally))
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-519 (Enterprise Admins)
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-1115
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-518 (Schema Admins)
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-512 (Domain Admins)
explorer.exe (1096): S-1-5-21-2682149276-1333600406-3352121115-520 (Group Policy Creator Owners)
```

cmdscan/consoles

- cmdscan
  - Recovers input to cmd.exe sessions
- consoles
  - Recovers input and output to cmd.exe sessions (the screen as the attacker/malware saw it)

Consoles Output

```
$ python vol.py -f rdp.mem --profile=Win2003SP2x86 consoles
**************************************************
ConsoleProcess: csrss.exe Pid: 7888
Console: 0x4c2404 CommandHistorySize: 50
HistoryBufferCount: 4 HistoryBufferMax: 4
OriginalTitle: Command Prompt
Title: Command Prompt
AttachedProcess: cmd.exe Pid: 5544 Handle: 0x25c
----
CommandHistory: 0x4c2c30 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 12 LastAdded: 11 LastDisplayed: 11
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x25c
Cmd #0 at 0x4c1f90: d:
Cmd #1 at 0xf41280: cd inetlogs
Cmd #2 at 0xf412e8: cd xxxxxxxxxx
Cmd #3 at 0xf41340: type xxxxxxxxxx.log | find " xxxxxxxxxx " | find "GET"
Cmd #4 at 0xf41b10: c:
Cmd #5 at 0xf412a0: cd\windows\system32\ xxxxxxxxxx
Cmd #6 at 0xf41b20: ftp xxxxxxxxxx.com
Cmd #7 at 0xf41948: notepad xxxxxxxxxx.log
Cmd #8 at 0x4c2388: notepad xxxxxxxxxx.log
Cmd #9 at 0xf43e70: ftp xxxxxxxxxx.com
Cmd #10 at 0xf43fb0: dir
Cmd #11 at 0xf41550: notepad xxxxxxxxxx.log
----
```

GUI Subsystem

Sessions [3]

- Every logged in user has a session object that tracks the processes, loaded kernel drivers, and its windows stations
- The ‘sessions’ column in *psxview* comes from enumerating every unique session along with its processes

Windows Stations [4]

- Each session has a set of windows stations
- These contain clipboard record information, atom tables, and a list of its Desktops
- Clipboard information can be used to determine which applications are set to receive related notifications

Desktops [5]

- Desktops are used to display distinct sets of graphical environments
- Malware analysis usefulness:
  - Finding hidden desktops were applications are active
  - The ‘desktopthrd’ column of *psxview* populates from linking running threads to their owning desktop


Anti-Child Porn Spam Protection (18 U.S.C. § 2252)
Warning! Access to your computer is limited. Your files has been decrypted.
We have detected spam advertises illegal sites with child pornography from your computer
This contradicts law and harm other network users and in this case we have to do next steps:
1. Block access to your desktop.
2. Totally block Safe-Mode and Network.
3. Encrypt your files using Advanced Encryption Standard and 256 symbols randomly generated password and delete source files using DOD 5220.22-M
(DOD 5220.22-M is the Department of Defense clearing and sanitizing standard - You cant recover your files - NEVER).
4. Sent this randomly generated password to our secure server and delete this password from your computer. (you cant get this password -NEVER)
This password is unique for each computer and stored on our secure server(and then erasing from this server and sending to us) and in each encrypted file.
If you think that you or some specialist can get this password from encrypted file - this is unreal even for specialist for goverment services, because here using
256-bit Advanced Encryption Standard.
To brute-force an AES-256-ECB encryption key in a known-plaintext attack, using all possible combinations,on a Cray XE6 with one million Opteron 6282 SE cores,
it would take up to ~66,282,862,563,751,221,625,826,507,369,649,000,000,000,000,000,000,000,000,000 years to complete the known-plaintext attack.
You have only two ways to decrypt your files:
1. Get Paid for decrypt password to us.
2. Wait ~66,282,862,563,751,221,625,826,507,369,649,000,000,000,000,000,000,000,000,000 years (who thinking that bruteforce can help read about AES online!).
Maybe you will remove locker but you have no other ways to decrypt or recover your files! No one person in the world can not decrypt your files or get the password to decrypt.
Forget about the fact that someone will help you.
Because your computer has been hacked or someone spamming from your computer. You must pay a penalty within 96 hours otherwise we will send report to FBI
with special password to decrypt some files wich contains spam software and child pornography files. (this special password is only for this files, not for all your files.
Password for all your files we will send you only after payment). If first 48 hours will be end you must pay 4000$ within next 48 hours.
To remove lock and decrypt your files you need to do next steps:
1. Buy Moneypak, Paysafecard or Ukash card 1000$ for first 48 hours and 4000$ after first 48 hours.
2. Send us email with your Id number and card code (you can use mobile internet from your cell phone or another PC to send email).
3. Wait 1-3 hours while we will send you reply email with two passwords to unlock and decrypt your files.
You can buy MoneyPak card at the nearest stores : Walgreens, Walmart, CVS/ pharmacy, Kmart, SevenEleven, Rite Aid or go to www.moneypak.com to find location stores near you
To find Paysafecard location stores near you visit www.paysafecard.com or Ukash at www.ukash.com
Our Guaranties:
If you dont trust us you can send any one file (no more 5mb, jpg, bmp or other picture, not a document) and your Id number to our email, then we wil decrypt it
and will send you reply with succesefully decrypted file.
Your ID Number and our contacts (please write down this data):
Your Id #: 1074470467 Our special service email: security11220@gmail.com
Send Code

Deskscan Plugin

```
$ python vol.py -f ACCFISA.vmem deskscan
Volatile Systems Volatility Framework 2.1_alpha
[snip]
**************************************************
Desktop: 0x24675c0, Name: WinSta0\My Desktop 2,
Next: 0x820a47d8
SessionId: 0, DesktopInfo: 0xbc310650, fsHooks: 0
spwnd: 0xbc3106e8, Windows: 111
Heap: 0xbc310000, Size: 0x300000, Base: 0xbc310000,
Limit: 0xbc610000
 652 (csrss.exe 612 parent 564)
 648 (csrss.exe 612 parent 564)
 308 (svchost.exe 300 parent 240)
```

ACCDFISA

$ **python vol.py -f ACCFISA.vmem wintree**
Volatile Systems Volatility Framework 2.1_alpha
[snip]
****

Window context: **0\WinSta0\My Desktop 2**
[snip]
.#100e2 csrss.exe:612 -
.#100e4 csrss.exe:612 -
#100de (visible) csrss.exe:612 -
.**Anti-Child Porn Spam Protection** (18 U.S.C. ? 2252) (visible) svchost.exe:300 WindowClass_0
..Send Code (visible) svchost.exe:300 Button
..#100ee (visible) svchost.exe:300 Edit
..Your Id #: 1074470467 Our special service email: **security11220@gmail.com** (visible) svchost.exe:300 Static
..Your ID Number and our contacts (please write down this data): (visible) svchost.exe:300 Static
..#100e8 (visible) svchost.exe:300 Static

Atoms [6]

- Atoms are strings that can be passed between processes without requiring full IPC
- Malware indirectly creates Atoms through a number of normal operations
- Volatility can find these instances to determine malware was active on a computer
  - Implemented in the *atomscan* plugin

atomscan & Clod

- Uses the *RegisterWindowMessage* API to register a *WM_HTML_GETOBJECT* window message
- This is part of its process to get scriptable control of Internet Explorer
- atomscan will show the *"WM_HTML_GETOBJECT"* atom

atomscan & laqma

- Using *SetWindowsHookEx* or *SetWinEventHook* to inject a DLL into another process will create an atom of the full DLL path
- A side effect probably not known by the author

atomscan & laqma

$ python vol.py –f laqma.vmem atomscan

```
AtomOfs(V)      Atom Refs   Pinned Name
---------- ---------- ------ ------ ----
0xe1000d10    0xc001      1      1 USER32
0xe155e958    0xc002      1      1 ObjectLink
0xe100a308    0xc003      1      1 OwnerLink
0xe2950c98    0xc19f      1      0 ControlOfs0142000000000FC
0xe11d6290    0xc1a0      1      0 C:\WINDOWS\system32\Dll.dll
0xe1106380    0xc1a1      1      0 BCGM_ONCHANGE_ACTIVE_TAB
```

GDI Timers [7]

- Userland applications can set timers to run functions at specified intervals
- The *timers* plugin lists the timers active in the memory capture
- Common uses:
  - Check for C&C updates
  - Scan for vulnerable hosts
  - <use your imagination>

gditimers & laqma

$ python vol.py -f laqma.vmem gditimers

Volatile Systems Volatility Framework 2.1_alpha

Thread	Process	nID	Rate(ms)	Countdown(ms)	Func
696	csrss.exe:660	0x7ffe	1000	734	0xbf8012b8
1648	explorer.exe:1624	0x15	60000	45109	0x00000000
1480	svchost.exe:1064	0x7476	60000	16234	0x74f51070
696	csrss.exe:660	0x7ffd	35000	25625	0xbf8f4d9a
1648	explorer.exe:1624	0x19	86400000	70004672	0x00000000
356	svchost.exe:1064	0x0	300000	131234	0x77532ebb
2024	lanmanwrk.exe:920	0x2eb	600000	589578	0x00401fc8
2024	lanmanwrk.exe:920	0x161	3000	1578	0x004010aa

More GUI Analysis

- Check the Month of Volatility Plugin posts from MHL:
- http://volatility-labs.blogspot.com/search/label/movp

Registry in Memory

Registry in Memory

- Portions of the registry that have been actively and recently used are cached in memory
- There are volatile keys in memory that are never written to disk
- Windows consults the cached version, and if the needed data exists, the on-disk version is never checked
- This leads to some interesting inconsistencies

Changing Hashes in Memory [8]

- An attacker can change the password hash of a user in memory and then log in to that account with the new password
- The changed hash is never written back to disk

Volatility Registry Analysis

- Volatility can query and report on all registry values in memory
- This lets you analyze the registry and its data in the way the operating system did while the machine was active

Callbacks

Kernel Callbacks

- Instead of actively hooking functions in the kernel to determine when certain events occur, there is a much safer, passive option
- You can "register" a function to be called by the system
- Every time the event occurs, your function is called

Process-Related Callbacks

- Activated on:
  - Process/thread creation and image (executable) loading
- Malware samples:
  - Mebroot, BlackEnergy, Rustock, TDL
- Uses:
  - Inject DLL into all new processes
  - Terminate a process before it can start

Filesystem-Related Callbacks

- Activated on:
  - New filesystem registration
- Malware samples:
  - TDL3, Stuxnet
- Uses:
  - TDL3 infects MBR, registers FS callback, and then is notified when main volume loads
  - Stuxnet attaches to the device stack for removable media plugged into the computer to hide its files

Bug-check Callbacks

- Activated on:
  - When machine is crashing (BSOD, crash dumping being written)
- Malware samples:
  - Rustock.C, Sinowal
- Uses:
  - Rustock.C cleans itself from memory before a crash dump is written
  - Sinowal ensures the MBR is still infected before shutting down after a BSOD

Registry Callbacks

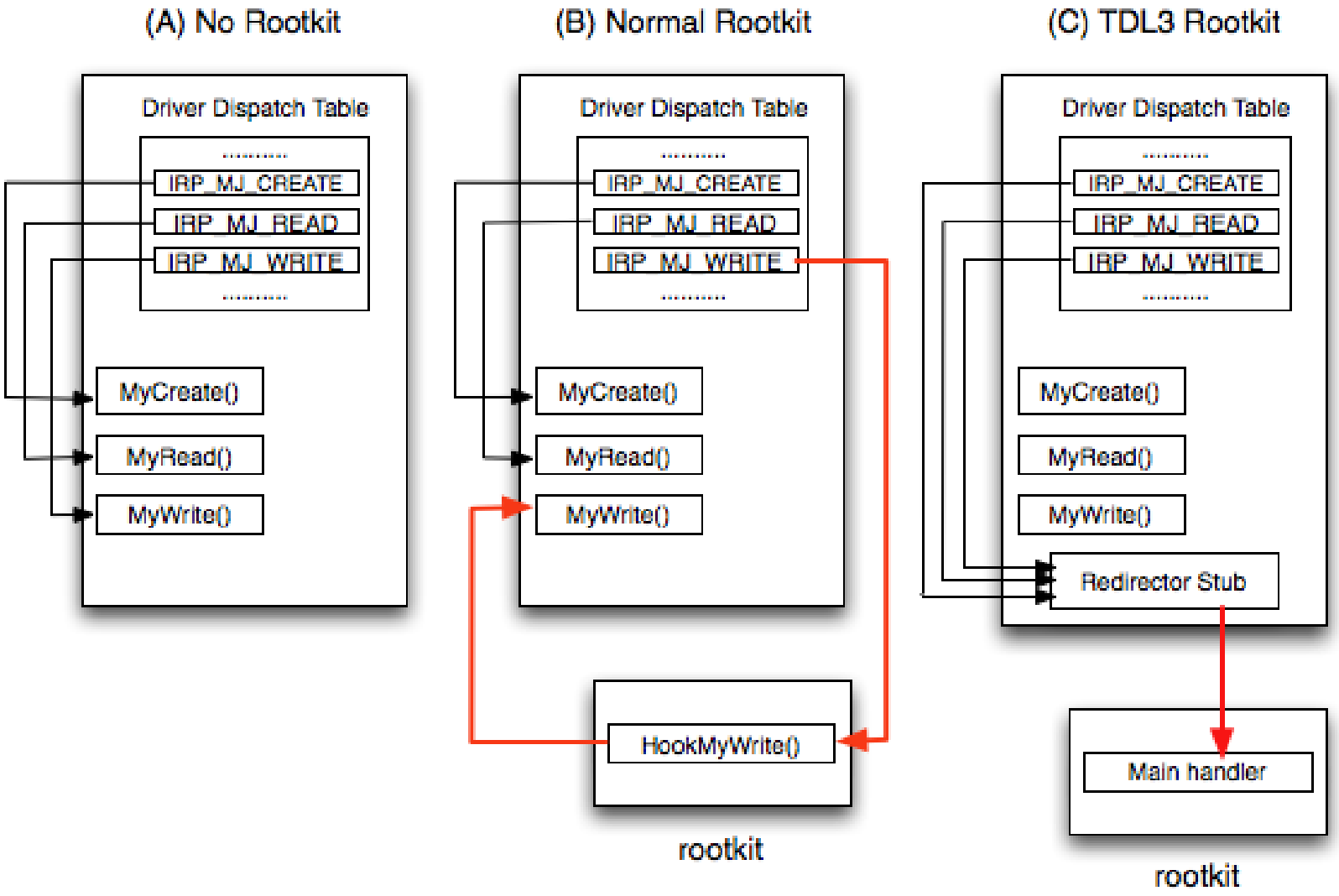
- Activated on:
  - Modifications to the registry, the callback can monitor, block, or modify the operation
- Malware Samples:
  - Various
- Uses:
  - Prevent persistence methods inside the registry (run keys, etc) from being modified/deleted

IRP Hooking

Driver IRPs

- I/O Request Packets
  - Carries details on a request to the responsible driver
  - Example: ReadFile() constructs an IRP behind the scenes and delivers it to the file system drivers
- Each driver has a list of IRP handlers called the major function table
  - These can be hooked by malware
  - *driverirp* plugin prints drivers' IRP tables

IRP Hooking


(A) No Rootkit
Driver Dispatch Table
.....
IRP_MJ_CREATE
IRP_MJ_READ
IRP_MJ_WRITE
.....
MyCreate()
MyRead()
MyWrite()
(B) Normal Rootkit
Driver Dispatch Table
.....
IRP_MJ_CREATE
IRP_MJ_READ
IRP_MJ_WRITE
.....
MyCreate()
MyRead()
MyWrite()
HookMyWrite()
rootkit
(C) TDL3 Rootkit
Driver Dispatch Table
.....
IRP_MJ_CREATE
IRP_MJ_READ
IRP_MJ_WRITE
.....
MyCreate()
MyRead()
MyWrite()
Redirector Stub
Main handler
rootkit

TDL3 IRP Hook

```
$ python vol.py -f mem.dmp driverirp -r vmscsi
Volatile Systems Volatility Framework 2.3_alpha
------------------------------------------------
DriverName: vmscsi
DriverStart: 0xf9db8000
DriverSize: 0x2c00
DriverStartIo: 0xf97ea40e
   0 IRP_MJ_CREATE                         0xf9db9cbd vmscsi.sys
   1 IRP_MJ_CREATE_NAMED_PIPE              0xf9db9cbd vmscsi.sys
   2 IRP_MJ_CLOSE                          0xf9db9cbd vmscsi.sys
   3 IRP_MJ_READ                           0xf9db9cbd vmscsi.sys
   4 IRP_MJ_WRITE                          0xf9db9cbd vmscsi.sys
   5 IRP_MJ_QUERY_INFORMATION              0xf9db9cbd vmscsi.sys
   6 IRP_MJ_SET_INFORMATION                0xf9db9cbd vmscsi.sys
   7 IRP_MJ_QUERY_EA                       0xf9db9cbd vmscsi.sys
   8 IRP_MJ_SET_EA                         0xf9db9cbd vmscsi.sys
   9 IRP_MJ_FLUSH_BUFFERS                  0xf9db9cbd vmscsi.sys
<snip>
```

TDL3 Hook Close-Up

```
$ python vol.py -f mem.dmp driverirp -r vmscsi -v
Volatile Systems Volatility Framework 2.3_alpha
------------------------------------------------
DriverName: vmscsi
DriverStart: 0xf9db8000
DriverSize: 0x2c00
DriverStartIo: 0xf97ea40e
   0 IRP_MJ_CREATE                          0xf9db9cbd vmscsi.sys

0xf9db9cbd a10803dfff      MOV EAX, [0xffdf0308]
0xf9db9cc2 ffa0fc000000    JMP DWORD [EAX+0xfc]
0xf9db9cc8 0000            ADD [EAX], AL
0xf9db9cca 0000            ADD [EAX], AL
0xf9db9ccc 0000            ADD [EAX], AL
```

Devices

Devices

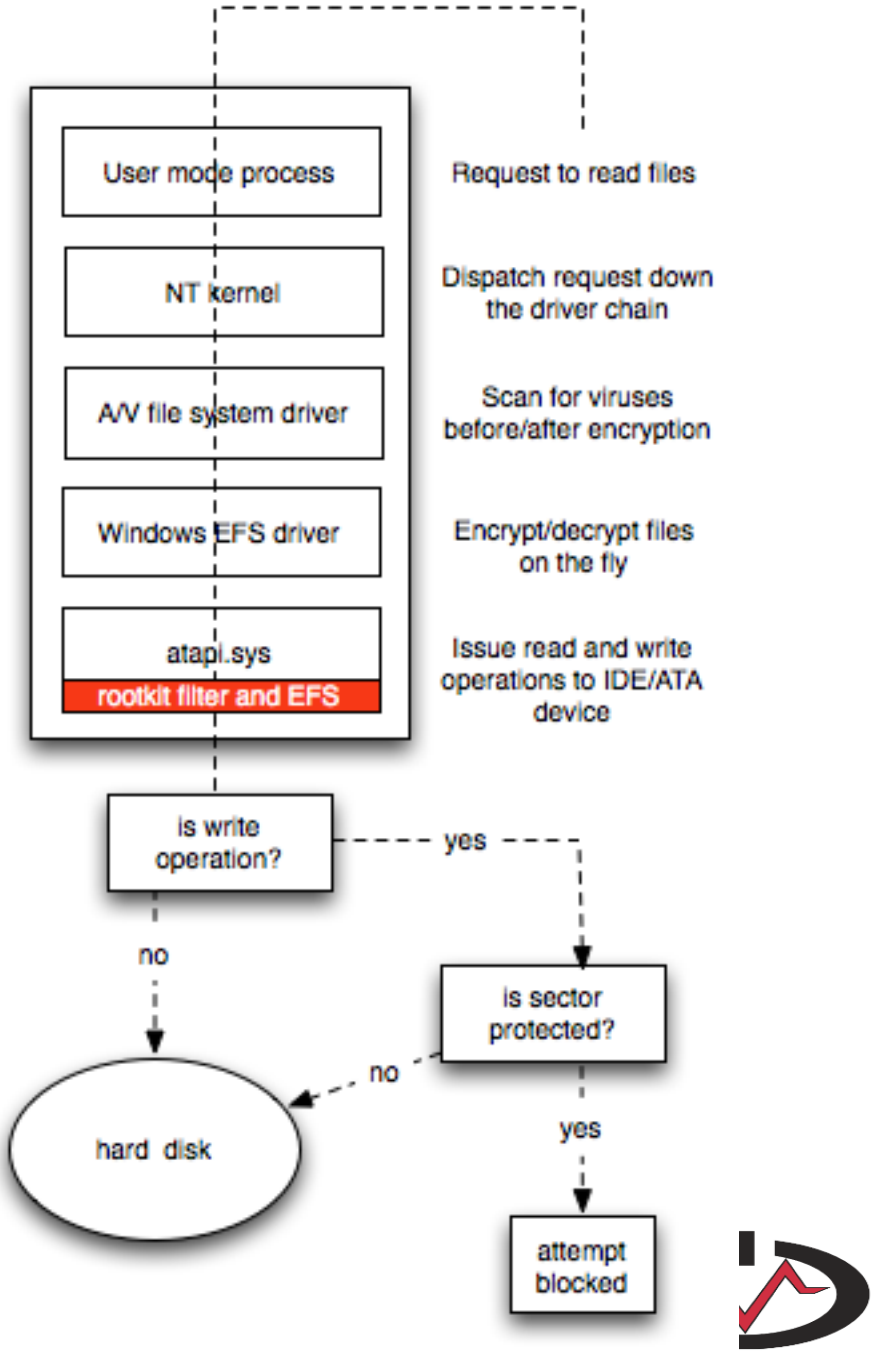
- Drivers wanting to communicate with user-mode applications create devices
  - IoCreateDevice("\\Device\\MyDevice", ...)
- Applications use CreateFile("\\\\.\\MyDevice") to open a handle to the device
  - Can then read/write/other driver-defined actions

Device Tree

- Multiple drivers can respond to or handle an IRP with the stacked, layered architecture
- The highest device in the stack gets the IRP first, and the lowest handles it last
- Examples
  - A firewall product may be inserted into the stack to filter network connections
  - A file system driver may be inserted into the stack for archiving, encryption, or compression
  - Malware inserts itself into stack to control operations

TDL3 Stacks

Prevents you from deleting the malware's own files



Stuxnet (*devicetree*)

```
DRV 0x0205e5a8 \FileSystem\vmhgfs
---| DEV 0x820f0030 hgfsInternal UNKNOWN
---| DEV 0x821a1030 HGFS FILE_DEVICE_NETWORK_FILE_SYSTEM
------| ATT 0x81f5d020 (?) - \FileSystem\FltMgr FILE_DEVICE_NETWORK_FILE_SYSTEM
---------| ATT 0x821354b8 (?) - \Driver\MRxNet FILE_DEVICE_NETWORK_FILE_SYSTEM

DRV 0x023ae880 \FileSystem\MRxSmb
---| DEV 0x81da95d0 LanmanDatagramReceiver FILE_DEVICE_NETWORK_BROWSER
---| DEV 0x81ee5030 LanmanRedirector FILE_DEVICE_NETWORK_FILE_SYSTEM
------| ATT 0x81bf1020 (?) - \FileSystem\FltMgr FILE_DEVICE_NETWORK_FILE_SYSTEM
---------| ATT 0x81f0fc58 (?) - \Driver\MRxNet FILE_DEVICE_NETWORK_FILE_SYSTEM

DRV 0x02476da0 \FileSystem\Cdfs
---| DEV 0x81e636c8 Cdfs FILE_DEVICE_CD_ROM_FILE_SYSTEM
------| ATT 0x81fac548 (?) - \FileSystem\FltMgr FILE_DEVICE_CD_ROM_FILE_SYSTEM
---------| ATT 0x8226ef10 (?) - \Driver\MRxNet FILE_DEVICE_CD_ROM_FILE_SYSTEM

DRV 0x0253d180 \FileSystem\Ntfs
---| DEV 0x82166020  FILE_DEVICE_DISK_FILE_SYSTEM
------| ATT 0x8228c6b0 (?) - \FileSystem\sr FILE_DEVICE_DISK_FILE_SYSTEM
---------| ATT 0x81f47020 (?) - \FileSystem\FltMgr FILE_DEVICE_DISK_FILE_SYSTEM
------------| ATT 0x81fb9680 (?) - \Driver\MRxNet FILE_DEVICE_DISK_FILE_SYSTEM
```

KLog

- Old proof of concept rootkit, but the technique is used in the wild

```
DRV 0x01f89310 \Driver\klog
---| DEV 0x81d2d730  FILE_DEVICE_KEYBOARD
```

```
DRV 0x02421770 \Driver\Kbdclass
---| DEV 0x81e96030 KeyboardClass1 FILE_DEVICE_KEYBOARD
---| DEV 0x822211e0 KeyboardClass0 FILE_DEVICE_KEYBOARD
------| ATT 0x81d2d730  - \Driver\klog FILE_DEVICE_KEYBOARD
```

Kernel Timers

- Kernel provides timer callbacks similar to userland
- Malware such as ZeroAccess and Rustock-C register timers that point to a hidden driver
- Volatility can detect this in the *timers* plugin

MBR & MFT

MBR Analysis

- Volatility has the ability to find MBRs in memory and to compare those in memory with MBRs from disk or previously extracted from memory
- Can find evidence/artifacts of MBR infectors
- See Jamie's presentation [9] and blog post [10]

Read these Blog Posts

- Solving the GrrCon Challenge [11]
- Malware In the Windows GUI Subsystem [12]
- Phalanx 2 Revealed [13]

Buy This Book

- Malware Analyst's Cookbook
  - Written by one of the Volatility developers
  - Goes in great detail over all the malware plugins

Questions/Comments?

- Contact Me:
  - andrew@madsecinc.com
  - @attrc
- Hacker Academy
  - http://www.hackeracademy.com
  - @hackeracademy
- Volatility:
  - http://volatility-labs.blogspot.com/
  - @volatility

References

[1] http://code.google.com/p/volatility/

[2] http://mnin.blogspot.com/2012/06/quickpost-flame-volatility.html

[3] http://volatility-labs.blogspot.com/2012/09/movp-11-logon-sessions-processes-and.html

[4] http://volatility-labs.blogspot.com/2012/09/movp-12-window-stations-and-clipboard.html

[5] http://volatility-labs.blogspot.com/2012/09/movp-13-desktops-heaps-and-ransomware.html

[6] http://volatility-labs.blogspot.com/2012/09/movp-21-atoms-new-mutex-classes-and-dll.html

[7] http://volatility-labs.blogspot.com/2012/10/movp-41-detecting-malware-with-gdi.html

[8] http://moyix.blogspot.com/2008/05/dfrws-2008-registry-forensics-in-memory.html

[9] http://volatility-labs.blogspot.com/2012/10/omfw-2012-reconstructing-mbr-and-mft.html

[10] http://volatility-labs.blogspot.com/2012/10/movp-43-recovering-master-boot-records.html

References Cont.

[11] http://volatility-labs.blogspot.com/2012/10/solving-grrcon-network-forensics.html

[12] http://volatility-labs.blogspot.com/2012/10/omfw-2012-malware-in-windows-gui.html

[13] http://volatility-labs.blogspot.com/2012/10/phalanx-2-revealed-using-volatility-to.html