Unmasking Careto through Memory Analysis

Andrew Case
@attrc

Who Am I?

- A Core Developer of Volatility and Registry Decoder
- Co-Author “Art of Memory Forensics”
- Lead-investigator on large-scale investigations
- Performed many RE efforts, pentests, and source code audits
- www.dfir.org

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
- Rapid triage/analysis leads

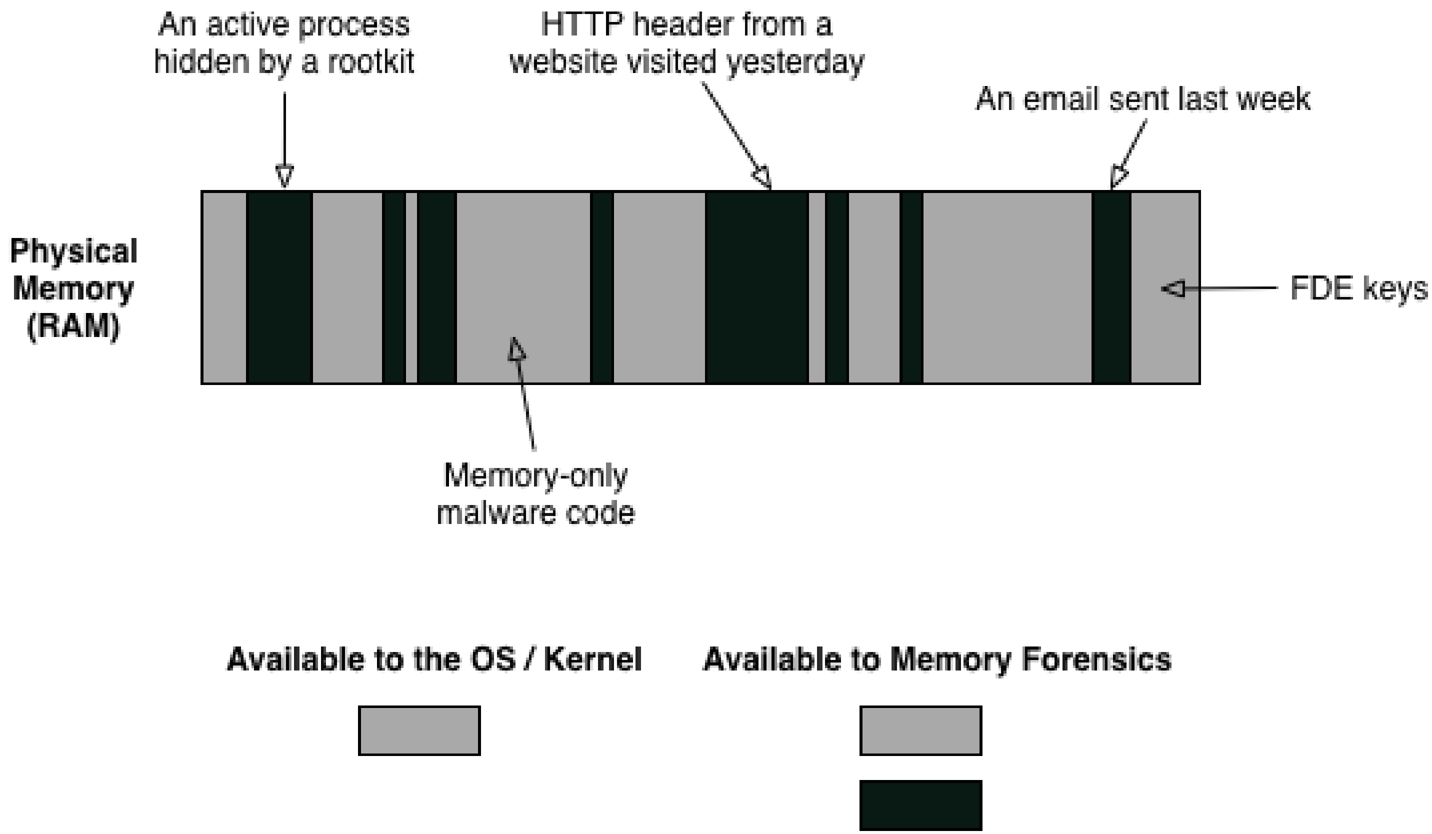
What can be recovered?

- Running processes
- Active network connections
- Loaded kernel drivers
- Console input and output
- Malware-created artifacts
- Application information (URL history, chat logs, emails)
- Disk encryption keys
- A whole lot more…

Why does it matter?

- Can recover the entire state of the operating system and running applications at the time of the capture
- Can also uncover historical information
- Advanced malware operates only in memory
- Sandbox, honeypot, automated analysis
- Much of the information recovered from memory is never written to disk or the network

Live IR vs Memory Forensics



Volatility Framework

- Implemented in Python under the GPL
- 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 latest release

Why use Volatility?

- Single, cohesive framework
  - x86 (PAE, non-PAE) and x64 Windows XP/2003 – 8.1/2012 R2
  - x86/x64 Linux kernels 2.6.11 – 3.16
  - x86/x64 Mac 10.5 (Leopard) – 10.9 (Mavericks)
  - Android
  - Modular: add new OS and architectures
- Open source, GNU GPLv2
  - read it, learn from it, extend it
  - understand how it works
- Python
  - RE and forensics language
  - distorm3, pycrypto, yara

Mapping Processes to Network Connections

```
$ python vol.py -f win764bit.raw --profile=Win7SP1x64 netscan
Volatility Foundation Volatility Framework 2.4
Offset(P)  Proto Local Address  Foreign Address State     Pid
Owner
0xf882a30  TCPv4   0.0.0.0:135       0.0.0.0:0  LISTENING 628
svchost.exe
0xfc13770  TCPv4 0.0.0.0:49154       0.0.0.0:0  LISTENING 916
svchost.exe
0xfdda1e0  TCPv6      :::49154            :::0  LISTENING 916
svchost.exe
0x1121b7b0 TCPv4   0.0.0.0:135       0.0.0.0:0  LISTENING 628
svchost.exe
0x1121b7b0 TCPv6        :::135            :::0  LISTENING 628
svchost.exe
0x11431360 TCPv4 0.0.0.0:49152       0.0.0.0:0  LISTENING  332
wininit.exe
0x11431360 TCPv6      :::49152            :::0  LISTENING  332
wininit.exe

[snip]

0x17de8980 TCPv6      :::49153            :::0  LISTENING  444
lsass.exe
0x17f35240 TCPv4 0.0.0.0:49155       0.0.0.0:0  LISTENING  880
svchost.exe
0x17f362b0 TCPv4 0.0.0.0:49155       0.0.0.0:0  LISTENING  880
```

```
$ 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)
```

Grrcon: What type of access did the attacker gain?

```
$ python vol.py -f rdp.mem --profile=Win2003SP2x86 consoles

ConsoleProcess: csrss.exe Pid: 7888
Console: 0x4c2404 CommandHistorySize: 50
AttachedProcess: cmd.exe Pid: 5544 Handle: 0x25c
Cmd #6 at 0xf41b20: ftp xxxxx.com
Cmd #7 at 0xf41948: notepad xxxxx.log
Cmd #8 at 0x4c2388: notepad xxxxx.log
Cmd #9 at 0xf43e70: ftp xxxxx.com
Cmd #10 at 0xf43fb0: dir
Cmd #11 at 0xf41550: notepad xxxxx.log

C:\WINDOWS\system32\xxxxx\sample>ftp xxxxx.com
Connected to xxxxx.com.
220 Microsoft FTP Service
User (xxxxx.com:(none)): xxxxx
331 Password required for xxxxx.
Password:
230 User xxxxx logged in.
ftp> cd logs
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
05-22-12  09:34AM       <DIR>          xxxxx
226 Transfer complete.
ftp: 51 bytes received in 0.00Seconds 51000.00Kbytes/sec.
```

Attacker commands are recoverable

Recovering TrueCrypt Artifacts

```
$ python vol.py -f Win8SP0x86-Pro.mem --profile=Win8SP0x86 truecryptsummary
Volatility Foundation Volatility Framework 2.3

Registry Version     TrueCrypt Version 7.1a
Process              TrueCrypt.exe at 0x85d79880 pid 3796
Kernel Module        truecrypt.sys at 0x9cd5b000 - 0x9cd92000
Symbolic Link        Volume{ad5c0504-eb77-11e2-af9f-8c2daa411e3c} ->
\Device\TrueCryptVolumeJ mounted 2013-10-10 22:51:29 UTC+0000
File Object          \Device\TrueCryptVolumeJ\ at 0x6c1a038
File Object
\Device\TrueCryptVolumeJ\Chats\GOOGLE\Query\modernimpact88@gmail.com.xml at 0x25e8e7e8
File Object          \Device\TrueCryptVolumeJ\Pictures\haile.jpg at 0x3d9d0810
File Object          \Device\TrueCryptVolumeJ\Pictures\nishikori.jpg at 0x3e44cc38
File Object          \Device\TrueCryptVolumeJ\$RECYCLE.BIN\desktop.ini at 0x3e45f790
File Object          \Device\TrueCryptVolumeJ\ at 0x3f14b8d0
File Object
\Device\TrueCryptVolumeJ\Chats\GOOGLE\Query\modernimpact88@gmail.com.log at 0x3f3332f0
Driver               \Driver\truecrypt at 0x18c57ea0 range 0x9cd5b000 - 0x9cd91b80
Device               TrueCryptVolumeJ at 0x86bb1728 type FILE_DEVICE_DISK
Container            Path: \??\C:\Users\Mike\Documents\lease.pdf
Device               TrueCrypt at 0x85db6918 type FILE_DEVICE_UNKNOWN
```

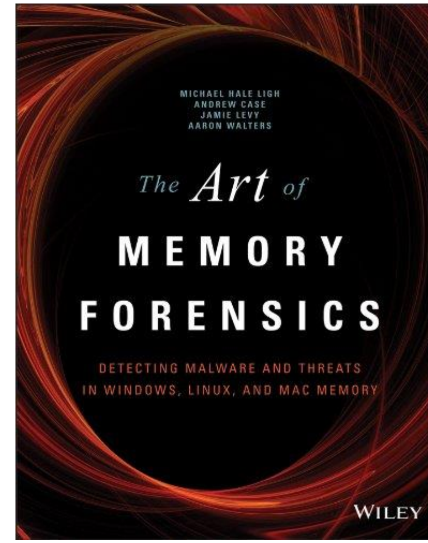
Extracting the Master Key

```
$ python vol.py -f WIN-QBTA4959AO9.raw --profile=Win2012SP0x64 truecryptmaster -D .
Volatility Foundation Volatility Framework 2.3

Container: \Device\Harddisk1\Partition1
Hidden Volume: No
Removable: No
Read Only: No
Disk Length: 7743733760 (bytes)
Host Length: 7743995904 (bytes)
Encryption Algorithm: SERPENT
Mode: XTS
Master Key
0xfffffa8018eb71a8  bb e1 dc 7a 8e 87 e9 f1 f7 ee f3 7e 6b b3 0a 25   ...z.......~k..%
0xfffffa8018eb71b8  90 b8 94 8f ef ee 42 5e 51 05 05 4e 32 58 b1 a7   ......B^Q..N2X..
0xfffffa8018eb71c8  a7 6c 5e 96 d6 78 92 33 50 08 a8 c6 0d 09 fb 69   .l^..x.3P......i
0xfffffa8018eb71d8  ef b0 b5 fc 75 9d 44 ec 8c 05 7f bc 94 ec 3c c9   ....u.D.......<.
Dumped 64 bytes to ./0xfffffa8018eb71a8_master.key
```

Documentation

- The Art of Memory Forensics
- Malware Analyst's Cookbook
- Websites
  - volatility-labs.blogspot.com
  - volatilityfoundation.org
  - memoryanalysis.net
- Volatility wiki
  - github.com/volatilityfoundation/volatility/wiki
  - community documentation: 200+ docs from 60+ different authors



Careto [1, 2]

- Discovered by Kaspersky in early 2014
  - Also called the "The Mask", which is the translation from Spanish
- Was in the wild for seven years before being detected
- One of the most "sophisticated" malware ever seen

Careto Capabilities

- Hiding files and network connections from the live system
- Data exfiltration
- Credential stealing from 20+ apps
- Anti-forensics
- Extensible through plugins

Windows Components

- Kernel
  - Known as SGH
  - Operates as a kernel driver
  - Main focus of this presentation
- Userland
  - “Careto”
  - Operates in userland through code/DLL injection

Encrypted Stores

- Everything used by Careto is encrypted on disk
- Drivers, DLLs, and configuration data are all decrypted as needed at runtime
- Through memory forensics we can automatically locate and extract these decrypted stores and code without any manual unpacking

Analysis Overview

- Setup
  - Snapshot virtual machine
  - Load the malware
  - Snapshot virtual machine again
- This gives us a baseline to compare with because each snapshot has a full capture of RAM
- Testing was performed on Windows XP and Windows 7 VMs

Analysis Goals

- Find the malware without any knowledge of its operation by analyzing the "after" snapshot
- Use differencing between each snapshot to build indicators of compromise
- Extract decrypted components that can be reverse engineered for deeper examination

Analysis without Pre-Knowledge

- Mimics most real-world environments that HIPS and security tools run inside
- Cannot use specific knowledge of malware samples if we hope to detect new ones
  - Must instead detect system state anomalies

“Floating” Drivers

- Drivers should generally be associated with a kernel module and a service
  – Those that aren’t - treat as suspicious
- Careto, like other malware, has floating drivers due to it reading them from disk and directly loading them into memory
  – Decryption happens in this phase as well

```
$ python vol.py -f xpafter.vmem drivermodule
Volatility Foundation Volatility Framework 2.4
Module                Driver
--------------------- ------------------
ndistapi.sys          NdisTapi
wdmaud.sys            wdmaud
sysaudio.sys          sysaudio
hidusb.sys            hidusb
vga.sys               VgaSave
CmBatt.sys            CmBatt
<snip>
```

```
$ python vol.py -f xpafter.vmem drivermodule|grep UNK
Volatility Foundation Volatility Framework 2.4
UNKNOWN                     \Driver\storage
UNKNOWN                     \Driver\PGPsdkDriver
UNKNOWN                     \Driver\loaddll
UNKNOWN                     \Driver\TdiFlt2
UNKNOWN                     \Driver\cipher
UNKNOWN                     \Driver\stopsec
UNKNOWN                     \Driver\cmprss
UNKNOWN                     Win32k
UNKNOWN                     \Driver\config
UNKNOWN                     \Driver\TdiFlt
UNKNOWN                     \Driver\fileflt
UNKNOWN                     RAW
UNKNOWN                     WMIxWDM
UNKNOWN                     ACPI_HAL
UNKNOWN                     PnpManager
```

```
$ python vol.py -f xpafter.vmem driverscan
Offset(P)  #Ptr #Hnd Start      Size    Service  Key Name  Driver Name
---------- ---- ---- ---------- ------- -----    --------  ----------------
0x4c10098     3    0 0x91fa6000 0x4f000 srv2     srv2      \FileSystem\srv2
0x6cfc6b0     4    0 0x91f11000 0x23000 mrxsmb   mrxsmb    \FileSystem\mrxsmb
0x6cfca90     3    0 0x91eff000 0x12000 mpsdrv   mpsdrv    \Driver\mpsdrv
0x6d352d0     3    0 0x91ee6000 0x19000 bowser   bowser    \FileSystem\bowser
0x6d356b0     2    0 0x91f6f000 0x1b000 mrxsmb20 mrxsmb20  \FileSystem\mrxsmb20
0x9848e8      2    0 0x82062000  0x4480                    \Driver\TdiFlt
0x9849f8      2    0 0x82061000   0xa80                    \Driver\stopsec
0x70c6d10     2    0 0x81fe7000  0x1c00                    \Driver\cipher
0x12410720    3    0 0x82067000  0xa000                    \Driver\TdiFlt2
0x1e7638d0    2    0 0x8203a000   0xa00                    \Driver\cmprss
028364030     2    0 0x8203b000  0x3680                    \Driver\loaddll
0x283642b8    3    0 0x8203f000  0x1d00                    \Driver\PGPsdkDriver
0x367e2378   12    0 0x82045000  0x7d00                    \Driver\fileflt
0x3bf4c308    2    0 0x81fd0000  0x2f80                    \Driver\storage
0x3bf4c418    2    0 0x81fea000  0x2000                    \Driver\config
```

Orphan Threads

- Kernel drivers can start threads to run independent units of code
- Many kernel malware samples will copy blocks of code into executable regions and then start a kernel thread at the copied address
- The thread data structure tracks this starting address and is leveraged inside Volatility

```
$ python vol.py -f xpafter.vmem threads -F OrphanThread
ETHREAD: 0x81d12870 Pid: 4 Tid: 204
Tags: OrphanThread,SystemThread
Created: 2014-10-09 19:16:53 UTC+0000
Exited: 1970-01-01 00:00:00 UTC+0000
Owning Process: System
Attached Process: System
State: Waiting:Executive
BasePriority: 0x8
Priority: 0x8
TEB: 0x00000000
StartAddress: 0x822cf6f0 UNKNOWN
ServiceTable: 0x80552fa0
  [0] 0x80501b8c
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_SYSTEM
0x822cf6f0 8b4c2404         MOV ECX, [ESP+0x4]
0x822cf6f4 e84bffffff       CALL 0x822cf644
0x822cf6f9 50               PUSH EAX
0x822cf6fa ff15a4162d82     CALL DWORD [0x822d16a4]
0x822cf700 c20400           RET 0x4
```

```
$ python vol.py -f xpafter.vmem threads -F OrphanThread | grep StartAddress
| cut -f 2 -d ' '
0x822cf6f0
0x820881c2
0x81ca6ada
<snip>
$ python vol.py -f xpafter.vmem drivermodule -a 0x822cf6f0
Volatility Foundation Volatility Framework 2.4
Module                                   Driver
------------------------------------ ------
UNKNOWN                                  \Driver\TdiFlt
$ python vol.py -f xpafter.vmem drivermodule -a 0x820881c2
Volatility Foundation Volatility Framework 2.4
Module                                   Driver
------------------------------------ ------
UNKNOWN                                  \Driver\PGPsdkDriver
$ python vol.py -f xpafter.vmem drivermodule -a 0x81ca6ada
Volatility Foundation Volatility Framework 2.4
Module                                   Driver
------------------------------------ ------
UNKNOWN                                  \Driver\fileflt
```

Callbacks

- Callbacks can be registered for system events:
  - Process creation
  - Registry key/value read/write/create/delete
  - File system registration
  - Bug checks (BSOD)
  - … many more
- The handler registered is called each time the event occurs
- Malware utilizes malicious callbacks for hiding, data stealing, and system manipulation

```
$ python vol.py -f xpafter.vmem callbacks
IoRegisterShutdownNotification 0xb2f07c96 vmhgfs.sys    \FileSystem\vmhgfs
IoRegisterShutdownNotification 0xf8307c6a VIDEOPRT.SYS  \Driver\mnmdd
IoRegisterShutdownNotification 0xf8307c6a VIDEOPRT.SYS  \Driver\VgaSave
IoRegisterShutdownNotification 0xf8bae5be Fs_Rec.SYS    \FileSystem\Fs_Rec
<snip>

$ python vol.py -f xpafter.vmem callbacks | grep UNKNOWN
Volatility Foundation Volatility Framework 2.4
IoRegisterShutdownNotification      0x81ca98ca UNKNOWN              -
GenericKernelCallback               0x81cad1fa UNKNOWN              -
GenericKernelCallback               0x81ca7d88 UNKNOWN              -
GenericKernelCallback               0x81ca7c88 UNKNOWN              -
GenericKernelCallback               0x81cad4ec UNKNOWN              -
IoRegisterFsRegistrationChange      0x81ca4d08 UNKNOWN              -
PsSetLoadImageNotifyRoutine         0x81cad1fa UNKNOWN              -
PsSetLoadImageNotifyRoutine         0x81ca7d88 UNKNOWN              -
PsSetCreateProcessNotifyRoutine     0x81cad4ec UNKNOWN              -
PsSetCreateProcessNotifyRoutine     0x81ca7c88 UNKNOWN              -
```

Driver IRPs

- Each driver sets handlers for its operations
  - Read, write, create, attributes, etc.
- Malicious drivers can setup handlers for requests from other components
  - For malicious drivers this can lead directly to the malware's code
- Malware also hooks IRP handlers of other drivers in order to control its operations

```
$ python vol.py --profile=Win7SP1x86 -f win7snap5.vmem driverip
DriverName: mrxsmb20
DriverStart: 0x91f6f000
DriverSize: 0x1b000
DriverStartIo: 0x0
   0 IRP_MJ_CREATE                          0x828cbda3 ntoskrnl.exe
   1 IRP_MJ_CREATE_NAMED_PIPE               0x828cbda3 ntoskrnl.exe
   2 IRP_MJ_CLOSE                           0x828cbda3 ntoskrnl.exe
   3 IRP_MJ_READ                            0x828cbda3 ntoskrnl.exe
   4 IRP_MJ_WRITE                           0x828cbda3 ntoskrnl.exe
   5 IRP_MJ_QUERY_INFORMATION               0x828cbda3 ntoskrnl.exe
   6 IRP_MJ_SET_INFORMATION                 0x828cbda3 ntoskrnl.exe
   7 IRP_MJ_QUERY_EA                        0x828cbda3 ntoskrnl.exe
   8 IRP_MJ_SET_EA                          0x828cbda3 ntoskrnl.exe
   9 IRP_MJ_FLUSH_BUFFERS                   0x828cbda3 ntoskrnl.exe
  10 IRP_MJ_QUERY_VOLUME_INFORMATION        0x828cbda3 ntoskrnl.exe
  11 IRP_MJ_SET_VOLUME_INFORMATION          0x828cbda3 ntoskrnl.exe
  12 IRP_MJ_DIRECTORY_CONTROL               0x828cbda3 ntoskrnl.exe
  13 IRP_MJ_FILE_SYSTEM_CONTROL             0x828cbda3 ntoskrnl.exe
  14 IRP_MJ_DEVICE_CONTROL                  0x828cbda3 ntoskrnl.exe
  15 IRP_MJ_INTERNAL_DEVICE_CONTROL         0x828cbda3 ntoskrnl.exe
  16 IRP_MJ_SHUTDOWN                        0x828cbda3 ntoskrnl.exe
  17 IRP_MJ_LOCK_CONTROL                    0x828cbda3 ntoskrnl.exe
  18 IRP_MJ_CLEANUP                         0x828cbda3 ntoskrnl.exe
  19 IRP_MJ_CREATE_MAILSLOT                 0x828cbda3 ntoskrnl.exe
  20 IRP_MJ_QUERY_SECURITY                  0x828cbda3 ntoskrnl.exe
<snip>
```

```
$ python vol.py --profile=Win7SP1x86 -f win7snap5.vmem driverirp|grep -c Unknown
Volatility Foundation Volatility Framework 2.4
280
$ python vol.py --profile=Win7SP1x86 -f win7snap5.vmem driverirp
DriverName: \Driver\TdiFlt
DriverStart: 0x82062000
DriverSize: 0x4480
DriverStartIo: 0x0
   0 IRP_MJ_CREATE                          0x820649b2 Unknown
   1 IRP_MJ_CREATE_NAMED_PIPE               0x820649b2 Unknown
   2 IRP_MJ_CLOSE                           0x820649b2 Unknown
   3 IRP_MJ_READ                            0x820649b2 Unknown
   4 IRP_MJ_WRITE                           0x820649b2 Unknown
<snip>
DriverName: \Driver\stopsec
DriverStart: 0x82061000
DriverSize: 0xa80
DriverStartIo: 0x0
   0 IRP_MJ_CREATE                          0x82354230 Unknown
   1 IRP_MJ_CREATE_NAMED_PIPE               0x82354230 Unknown
   2 IRP_MJ_CLOSE                           0x82354230 Unknown
   3 IRP_MJ_READ                            0x82354230 Unknown
   4 IRP_MJ_WRITE                           0x82354230 Unknown
<snip>
```

Finding Injected DLLs

- The loaddll driver injects DLLs into browser processes
- Volatility's malfind can automatically find the shellcode used for injection
  - It looks for suspicious process mappings, such as RWX regions and non-file backed PE files

```
Process: IEXPLORE.EXE Pid: 1164 Address: 0x270000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE

0x00270000  00 00 00 00 00 00 00 00 68 00 00 26 00 e8 69 1d   ........h..&..i.
0x00270010  59 7c 6a ff e8 2d 24 59 7c 83 c4 04 c3 00 00 00   Y|j..-$Y|.......
0x00270020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00270030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x270000 0000             ADD [EAX], AL
0x270002 0000             ADD [EAX], AL
0x270004 0000             ADD [EAX], AL
0x270006 0000             ADD [EAX], AL
0x270008 6800002600       PUSH DWORD 0x260000
0x27000d e8691d597c       CALL 0x7c801d7b ← This is LoadLibrary
0x270012 6aff             PUSH -0x1
0x270014 e82d24597c       CALL 0x7c802446
0x270019 83c404           ADD ESP, 0x4
0x27001c c3               RET

Process: IEXPLORE.EXE Pid: 1164 Address: 0x260000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x00260000  53 79 73 74 65 6d 33 32 5c 76 63 68 77 39 78 2e   System32\vchw9x.
0x00260010  64 6c 6c 00 00 00 00 00 00 00 00 00 00 00 00 00   dll.............
```

Extraction

- In Careto, all of the kernel drivers and DLLs are encrypted on disk
- They get decrypted at runtime by the main driver (scsimap.sys) and then loaded
- We can automatically extract the drivers we previously found with Volatility
- No IDA needed!

```
$ python vol.py -f xpafter.vmem driverscan | grep TdiFlt2
0x000000000209b388 2  0 0x82074000     0xa000
\Driver\TdiFlt2
$ python vol.py -f xpafter.vmem moddump -b 0x82074000 -D .
Volatility Foundation Volatility Framework 2.4
Module Base Module Name          Result
----------- -------------------- ------
0x082074000 UNKNOWN              OK: driver.82074000.sys
$ file driver.82074000.sys
driver.82074000.sys: PE32 executable (native) Intel 80386, for MS
Windows
```

Differencing for Indicators

- We compare three common sources of indicators
  - Services
  - Mutexes
  - Symlinks

A New Service Appears…

```
$ diff xpbefore/svcscan.txt xpafter/svcscan.txt
> Offset: 0x38b260
> Order: 265
> Start: SERVICE_SYSTEM_START
> Process ID: -
> Service Name: scsimap
> Display Name: scsimap
> Service Type: SERVICE_KERNEL_DRIVER
> Service State: SERVICE_RUNNING
> Binary Path: \Driver\scsimap
```

Marking with a Mutex

```
$ diff xpbefore/mutantscan.txt xpafter/mutantscan.txt
> 0x0000000001eb1818 2  1 -1 0x820c2960 1164:1684
{36A900E5-0AE5-4ca6-84B4-45A05B42E705}_262144_124160
```

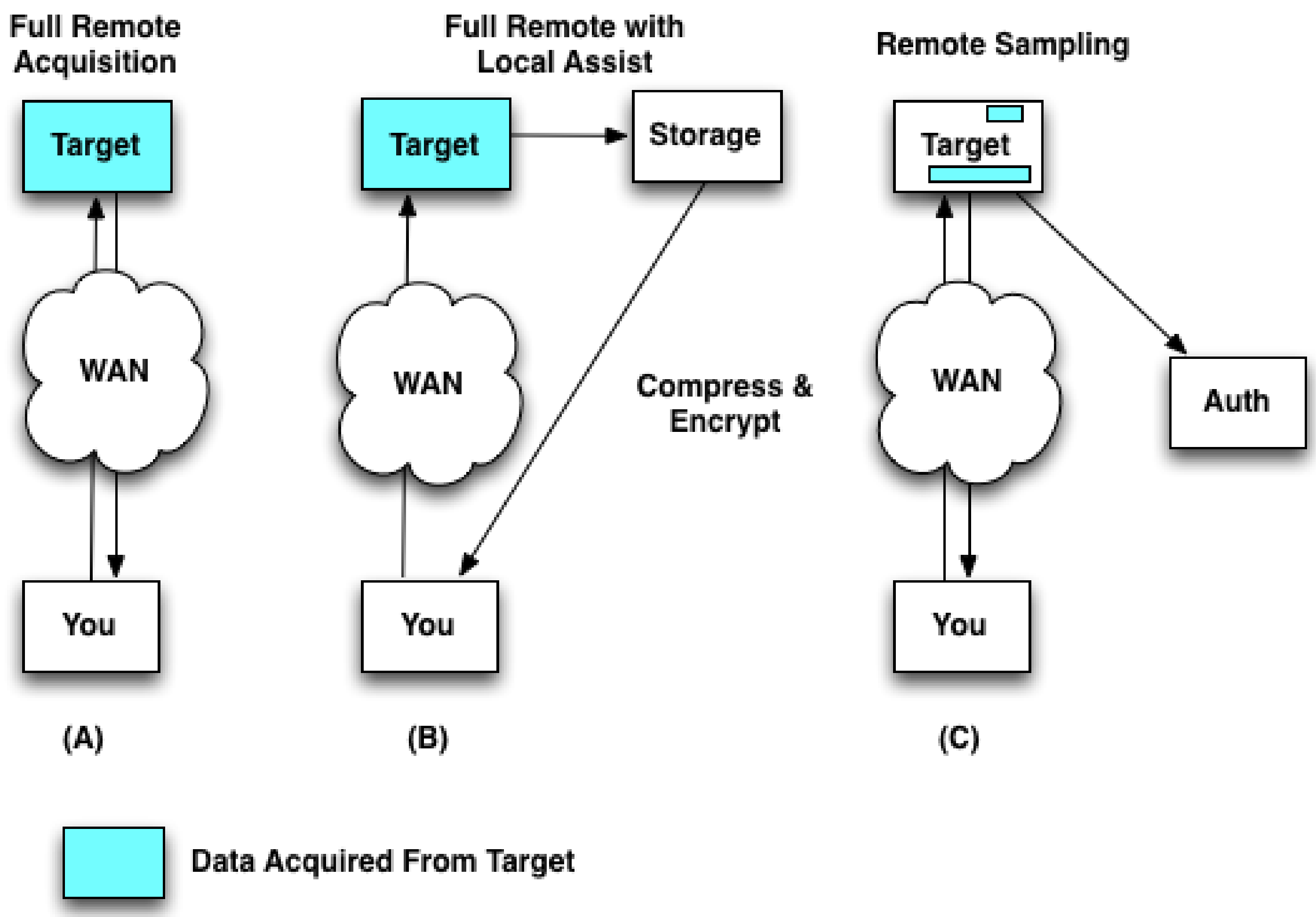
Pid 1164 is the process in which malfind found the injected DLL

A Device’s Symlink

```
$ diff xpbefore/symlink.txt xpafter/symlink.txt
> 0x0000000008b728e0  1 0 2014-10-09 19:16:53 UTC+0000
{E07DB02C...934B7D6} \Device\{E07DB02C-387E-43b2-A6F2-
C59B4934B7D6}
```

Scaling Memory Forensics

Methods of Network Acquisition



Notes on Sampling

- Can query small amounts of memory (<10MB) to deeply explore system state
- IOCs can quickly be checked for and then full acquisition initiated if found
- This allows for large scale sweeps of data

Sampling with Local Agents

- In certain environments the network was not designed to support the extra data from remote sampling
- Agents on the local machine can also perform sweeping in these cases
- This reduces bandwidth requirements greatly as only alerts are sent
- Downside is that more code must run on each end point

Effects of Sampling

- Machines are swept at a reasonable interval (hours or days) looking for suspicious behavior
- Organizations can add their own threat intelligence and IOCs to sweeping tools
- Meaningful alerts are then triggered

Sampling for Active Hunting

- Why wait until a compromise is suspected?
- Why not actively look for indicators?
- Attackers persist on systems for months or years because no one is looking or they are looking in the wrong places
- Proactive threat hunting is essential to deal with modern threats

Conclusions

- Memory forensics is now a required part of IR processes
- Limiting yourself to disk and network forensics misses critical data
- Don't wait for attackers to slip up and trigger an alert – go hunting for them!

Questions?

- Contact
  - andrew@dfir.org
  - @attrc
- Volatility
  - www.volatilityfoundation.org
  - @volatility

References

[1] http://www.kaspersky.com/about/news/virus/2014/Kaspersky-Lab-Uncovers-The-Mask-One-of-the-Most-Advanced-Global-Cyber-espionage-Operations-to-Date-Due-to-the-Complexity-of-the-Toolset-Used-by-the-Attackers

[2] https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/25000/PD25037/en_US/McAfee_Labs_Threat_Advisory_Careto_Attack_The%20Mask_3.pdf