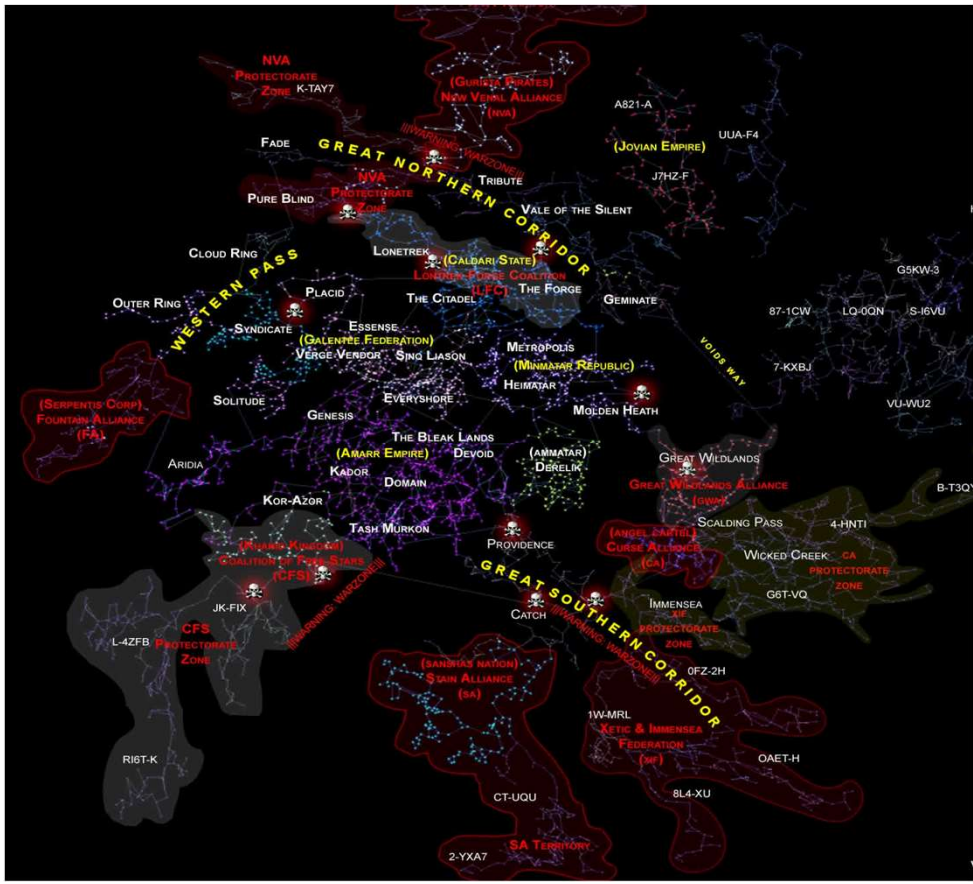
Memory analysis training

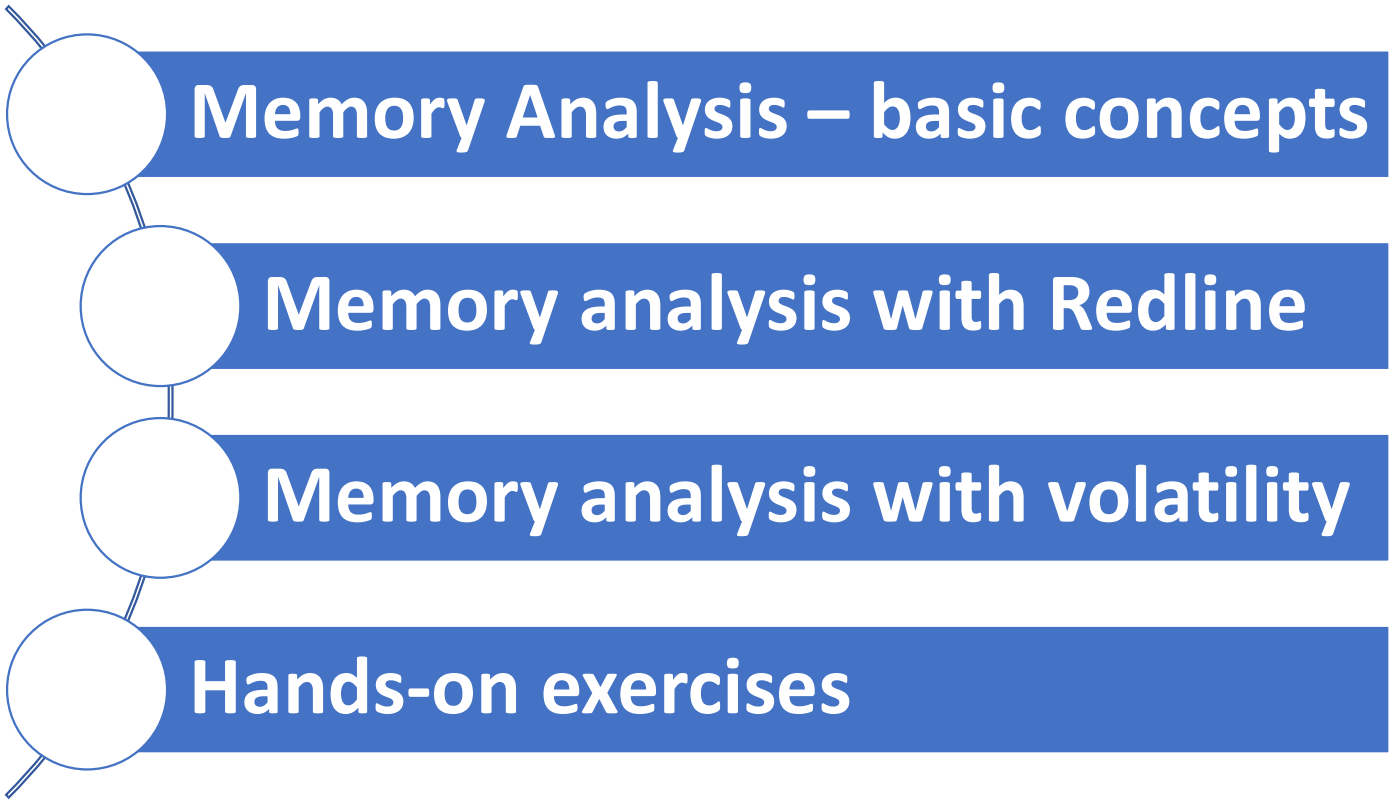
Quick Admin

- ☐ Computer checklist
  - ☐ VMWare/Virtual Box
  - ☐ Access rights
  - ☐ Images
- ☐ Wireless for personal use
- ☐ Amenities
- ☐ Breaks & cig
- ☐ Meals
- ☐ Beer?

- ☐ Evaluation – Final Exam
- ☐ Participation diploma



Agenda


Memory Analysis – basic concepts
Memory analysis with Redline
Memory analysis with volatility
Hands-on exercises

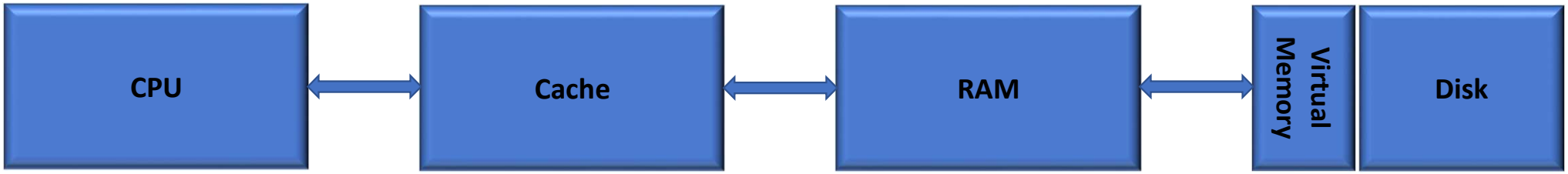
Memory Analysis – basic concepts

Memory Analysis – b. concepts

Memory analysis with Redline

Memory analysis with volatility

Memory Forensics - Why?



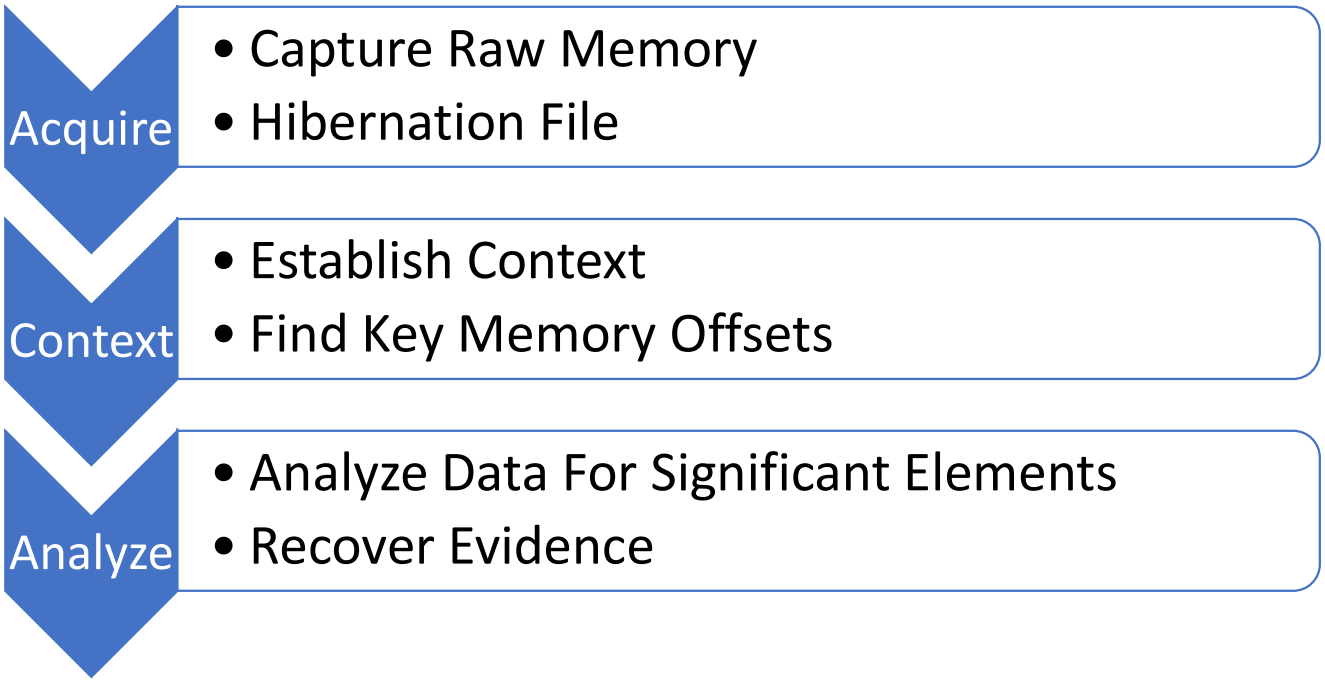
Everything in the OS traverses RAM	
Running processes and the system objects/resources with which they interact.	Portions of nonvolatile sources of evidence such as the registry, event log, and Master File Table.
Active network connections	Malware
Remnants of previously executed console commands.	Open Files
Loaded drivers	Encryption keys and clear-text data that is otherwise encrypted on disk.
User credentials (hashed, obfuscated, clear text)	Important data structures within the kernel that provide insight into process accounting, behavior, and execution.

Memory Forensics Advantages

- **Best place to identify malicious software activity**
  - Study running system
  - Identify inconsistencies in system
  - Bypass packers, binary ofuscations, rootkits.
- **Analyze recent activity on the system**
  - Identify all recent activity in context
  - Profile user or attacker activities
- **Collect evidence that cannot be found anywhere else**
  - Memory-only malware
  - Chat threads
  - Internet activities

What is Memory Forensics?

- **Study of data captured from memory of a target system**
- **Ideal analysis includes physical memory data (from RAM) as well as Page File (or SWAP space) data**



Windows Memory Analysis

1. Identify Context

- the Kernel Processor Control Region (KPCR)or Kernel Debugger Data Block (KDBG)

2. Parse Memory Structures

- Executive Process(EPROCESS)blocks
- ProcessEnvironment (PEB) blocks
  - DLLs loaded
- Virtual Address Descriptors (VAD) Tree
  - List of memory sections belonging to the process
- Kernel modules I drivers

3. Scan of Outliers

- Unlinked processes, DLLs, sockets and threads
- Unmapped memory pages with execute privileges
- Hook detection
- Known heuristics and Signatures

4. Analysis: search for anomalies

Finding the First “HIT”

1. Identify rogue processes
2. Analyze process DLLs and handles
3. Review network artifacts
4. Look for code injections
5. Search for rootkits
6. Dump suspicious processes and drivers

Analyzing Process Objects

Windows processes are composed of much more than just a binary file.

DLLs	Dynamic linked libraries (shared code)
Handles	Pointer to a resource
Files	Open files or I/O devices
Directories	lists of names used for access to kernel objects
Registry	Access to a key within the Windows Registry
Mutexes / Semaphores	Control/limit access to an object
Events	Notifications that help threads communicate and organize
Threads	Smallest unit of execution; the workhorse of a process
Memory Sections	Shared memory areas used by a process
Sockets	Network port and connection information within a process

Detecting Injection

- **DLL injection is very common with modern malware**
  - VirtualAllocEx( ) and CreateRemoteThread( )
  - SetwindowsHookEx()
- **Process hollowing is another injection technique**
  - Malware starts a new instance of legitimate process
  - Original process code de-allocated and replaced
  - Retains DLLs, handles, data, etc. from original process
- **Code injection is relatively easy to detect**
  - Review memory sections marked as Page_Execute_ReadWrite and having no memory-mapped file present
    - ✓ Scan for DLLs (PEfiles) and shellcode
- Process image not backed with file on disk = process hollowing

Rootkit Hooking

System Service Description Table (SSDT)

- Kernel instruction hooking

Interrupt Description Table (IDT)

- Kernel hooks; Not very common on modern systems

Import Address Table (IAT) and inline API

- User mode DLL function hooking
- Volatility **apihooks** module is best for identifying

I/O Request Packets (IRP)

- Driver hooking

Malware Persistence Mechanisms

Rapid Memory Search

- You can find:
  - IP Addresses/Domain Names
  - Malware file names
  - Usernames
  - Email addresses

- **Step 1:** Create ASCII and Unicode strings files

srch_strings –t d –a memory.img > memory.asc

srch_strings –t d –a –e l memory.img > memory.uni

- **Step 2:** Search for indicators

grep -i *string memory.asc*

Memory analysis with Redline

Memory Analysis – b. concepts

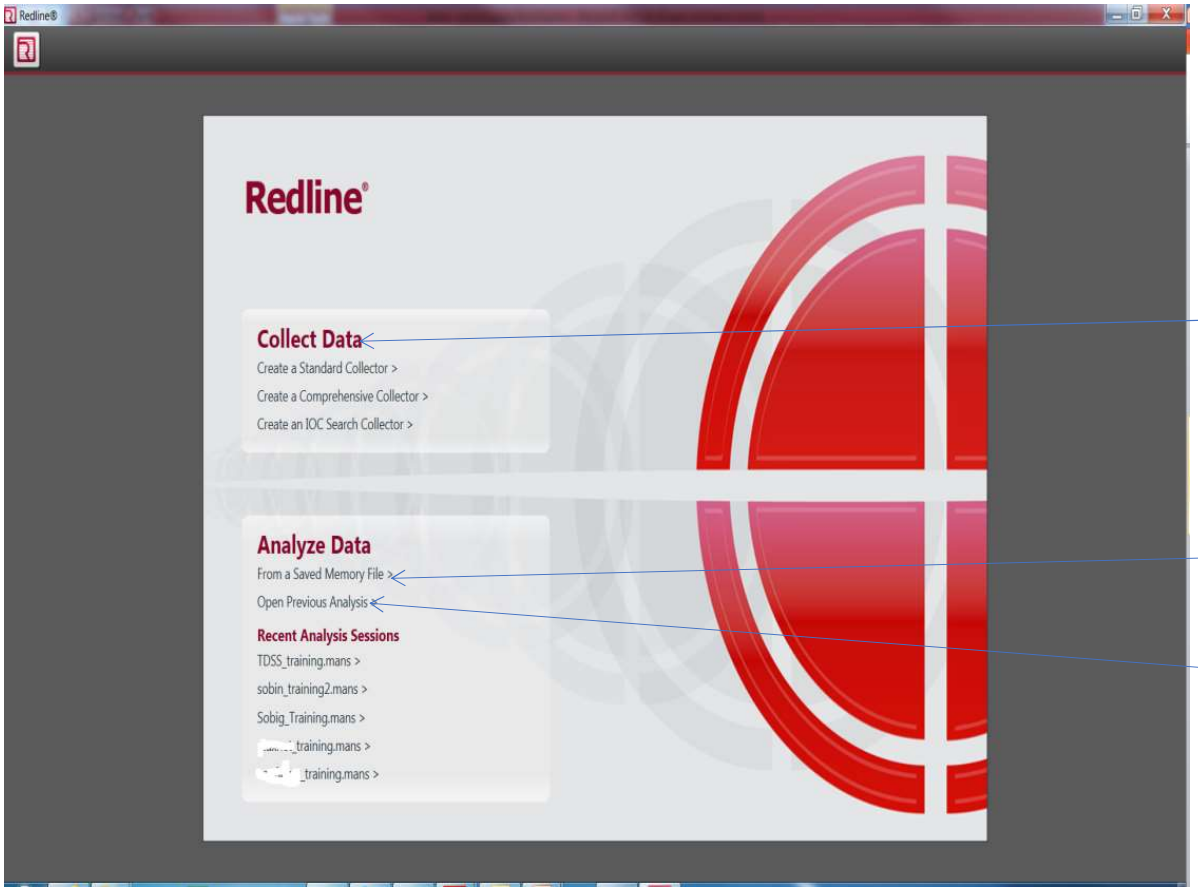
Memory analysis with Redline

Memory analysis with volatility

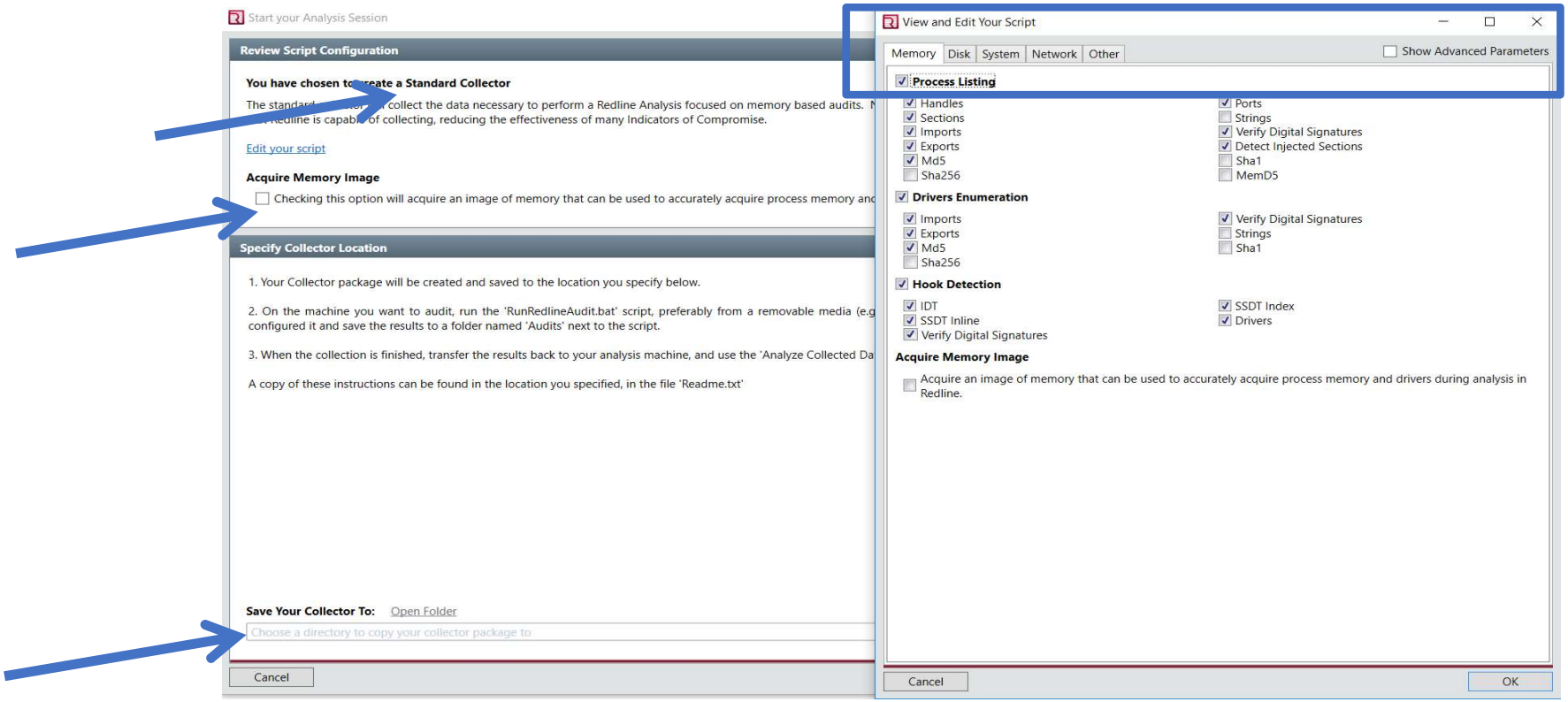
Mandiant Redline - overview

- GUI tool for memory analysis
  - Processes
  - Handles
  - Network Connections
  - Memory Sections
  - Hooks and drives
- Buil-in heuristics for suspicious processes and code
- Live memory analysis and live response capability
- IoC matching
- File whitelisting

Mandiant Redline – getting started



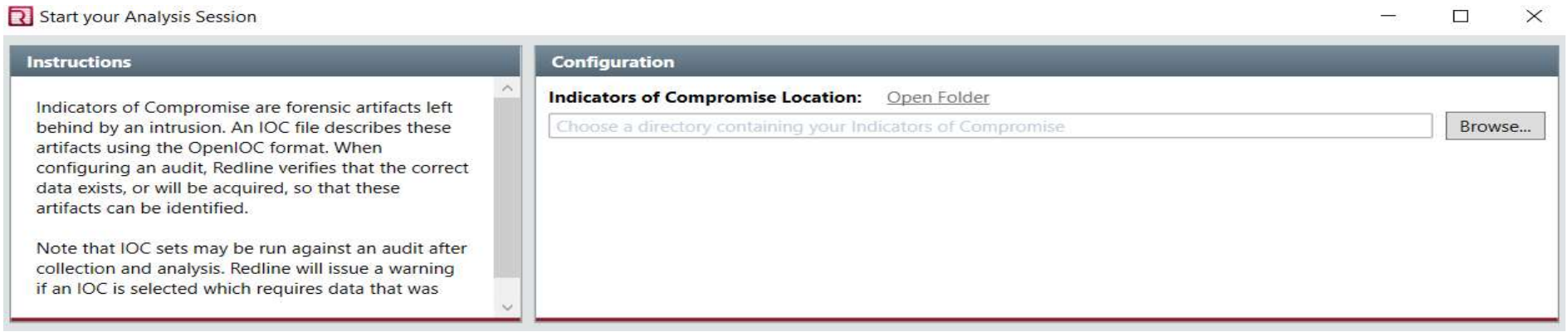
Mandiant Redline – Building a portable agent



Mandiant Redline – IoC Analysis

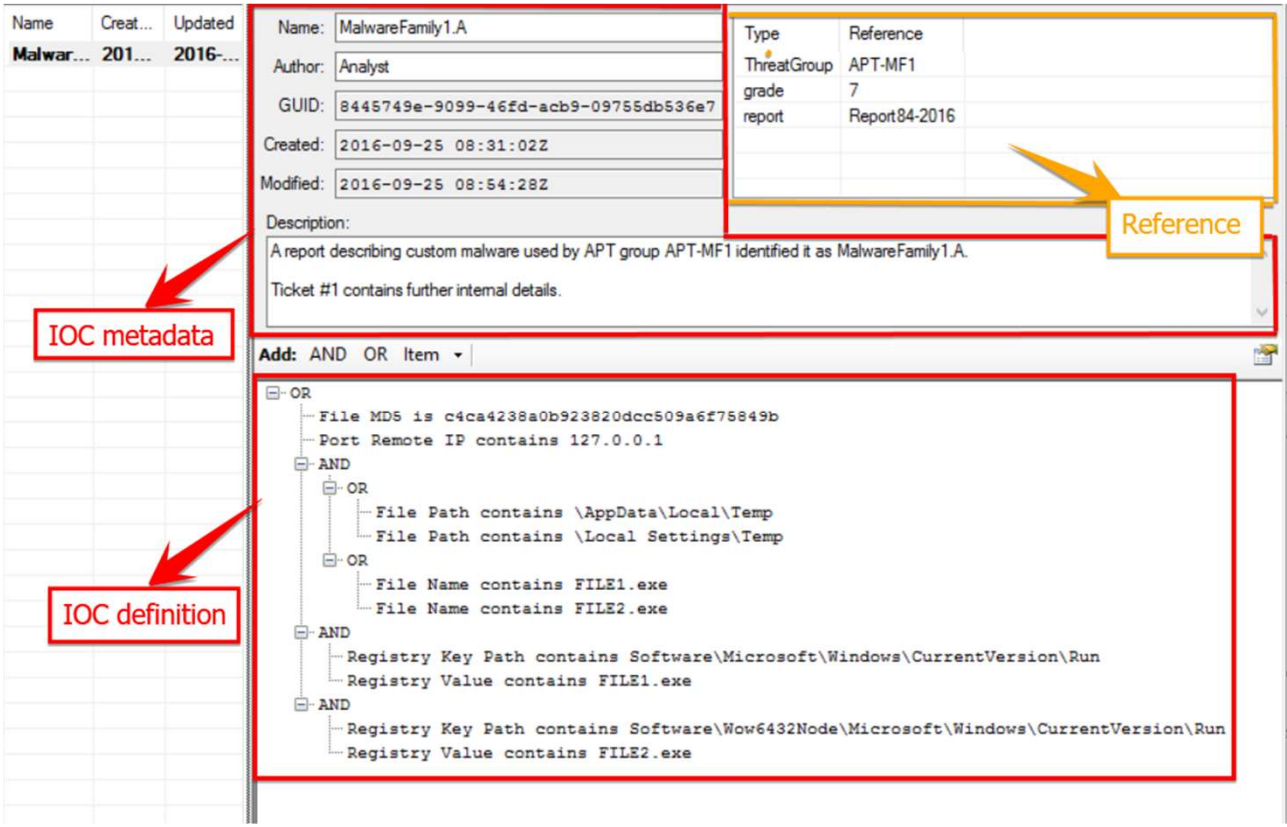
Indicators of Compromise allow a wide range of alert triggers to be set for known malware - Processes, hooks, drivers, handles, strings

IOCs can be used with any live / dead memory analysis in Redline - Scan for a single IOC or hundreds

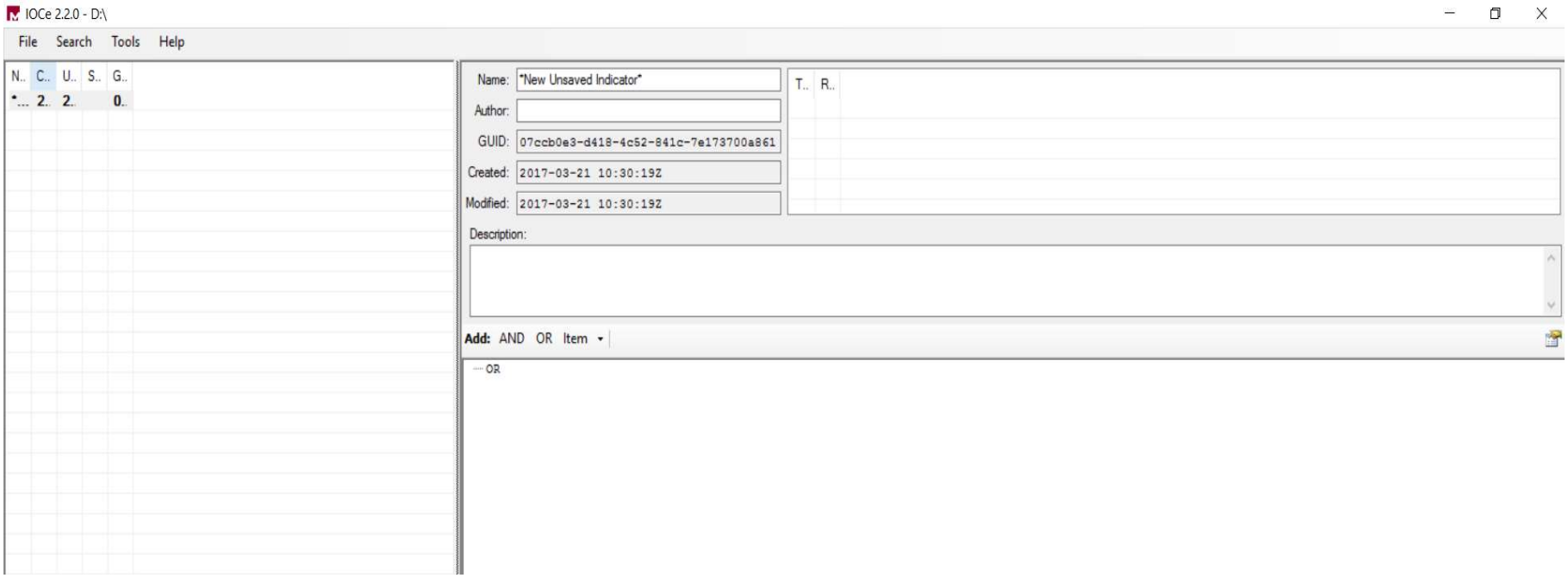


OpenIoc Format

- Open source framework developed by Mandiant
- Utilizes XML to describe threat information
- Easily transformed to a format used by IT monitoring tools (Yara and Snort)
- Free tools for managing them: IOC Editor and IOC-EDT



Mandiant IOCe overview

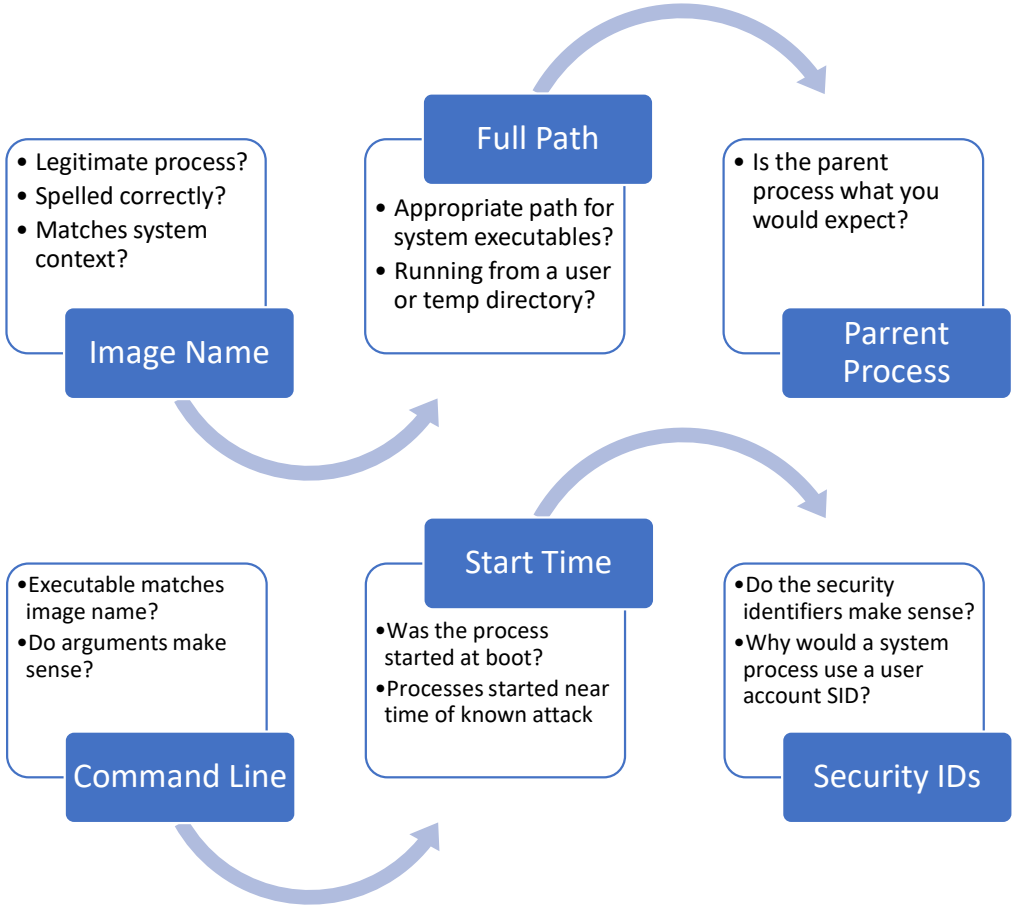


Mandiant IOCe exercise

Please create an IOC bucket for Red October campaign. You can use any public information.

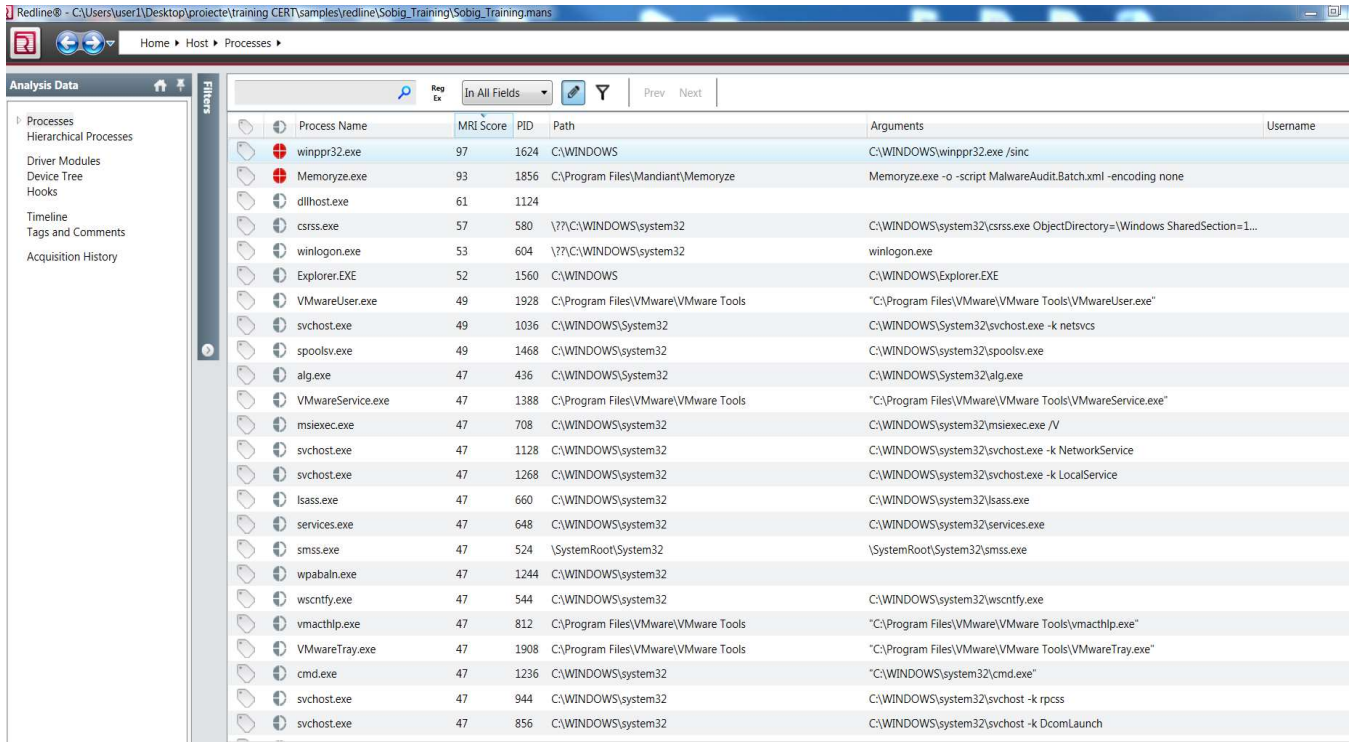
Duration: ~1h

Analyzing Processes



Identify Rogue Processes

- Redline analysis of a memory dump



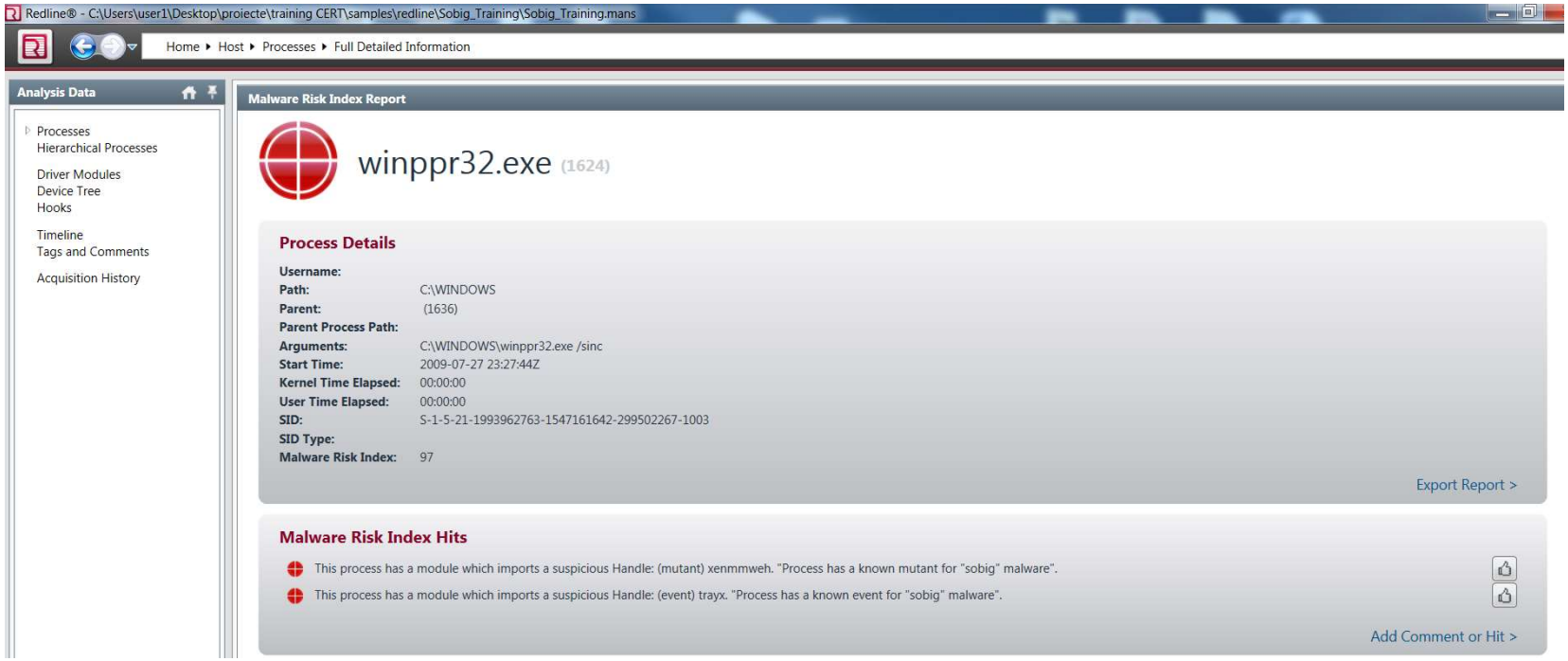
Process Name	MRI Score	PID	Path	Arguments	Username
winppr32.exe	97	1624	C:\WINDOWS	C:\WINDOWS\winppr32.exe /sinc	
Memoryze.exe	93	1856	C:\Program Files\Mandiant\Memoryze	Memoryze.exe -o -script MalwareAudit.Batch.xml -encoding none	
dllhost.exe	61	1124			
csrss.exe	57	580	\??\C:\WINDOWS\system32	C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1...	
winlogon.exe	53	604	\??\C:\WINDOWS\system32	winlogon.exe	
Explorer.EXE	52	1560	C:\WINDOWS	C:\WINDOWS\Explorer.EXE	
VMwareUser.exe	49	1928	C:\Program Files\VMware\VMware Tools	"C:\Program Files\VMware\VMware Tools\VMwareUser.exe"	
svchost.exe	49	1036	C:\WINDOWS\System32	C:\WINDOWS\System32\svchost.exe -k netsvcs	
spoolsv.exe	49	1468	C:\WINDOWS\system32	C:\WINDOWS\system32\spoolsv.exe	
alg.exe	47	436	C:\WINDOWS\System32	C:\WINDOWS\System32\alg.exe	
VMwareService.exe	47	1388	C:\Program Files\VMware\VMware Tools	"C:\Program Files\VMware\VMware Tools\VMwareService.exe"	
msiexec.exe	47	708	C:\WINDOWS\system32	C:\WINDOWS\system32\msiexec.exe /V	
svchost.exe	47	1128	C:\WINDOWS\system32	C:\WINDOWS\system32\svchost.exe -k NetworkService	
svchost.exe	47	1268	C:\WINDOWS\system32	C:\WINDOWS\system32\svchost.exe -k LocalService	
lsass.exe	47	660	C:\WINDOWS\system32	C:\WINDOWS\system32\lsass.exe	
services.exe	47	648	C:\WINDOWS\system32	C:\WINDOWS\system32\services.exe	
smss.exe	47	524	\SystemRoot\System32	\SystemRoot\System32\smss.exe	
wpabaln.exe	47	1244	C:\WINDOWS\system32		
wscntfy.exe	47	544	C:\WINDOWS\system32	C:\WINDOWS\system32\wscntfy.exe	
vmacthlp.exe	47	812	C:\Program Files\VMware\VMware Tools	"C:\Program Files\VMware\VMware Tools\vmacthlp.exe"	
VMwareTray.exe	47	1908	C:\Program Files\VMware\VMware Tools	"C:\Program Files\VMware\VMware Tools\VMwareTray.exe"	
cmd.exe	47	1236	C:\WINDOWS\system32	"C:\WINDOWS\system32\cmd.exe"	
svchost.exe	47	944	C:\WINDOWS\system32	C:\WINDOWS\system32\svchost -k rpcss	
svchost.exe	47	856	C:\WINDOWS\system32	C:\WINDOWS\system32\svchost -k DcomLaunch	

Identify Rogue Processes: MRI - Malware Risk Index

1. Behavior Ruleset
   - Code injection detection
   - Process image path verification
     - Svchost outside system32 = Bad
   - Process user verification (SIDs)
     - *dllhost* running as admin = Bad
   - Process Handle Inspection
     - iexplore. exe opening cmd.exe = Bad
     - *)! voqa. i4* = known Poison Ivy mutant
2. Verify Digital Signatures
   - Only available during live analys
   - Executable, DLL, and driver sig checks
   - Not signed?
     - Is it found in >75% of all process?

Identify Rogue Processes

- Details of the process with the biggest MRI

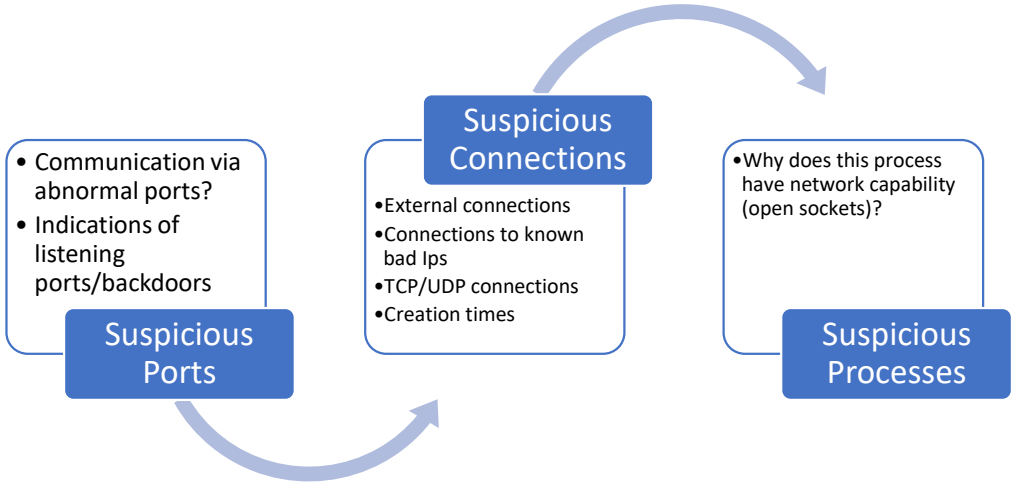


Identify Rogue Processes

1. Why this is a bad process?

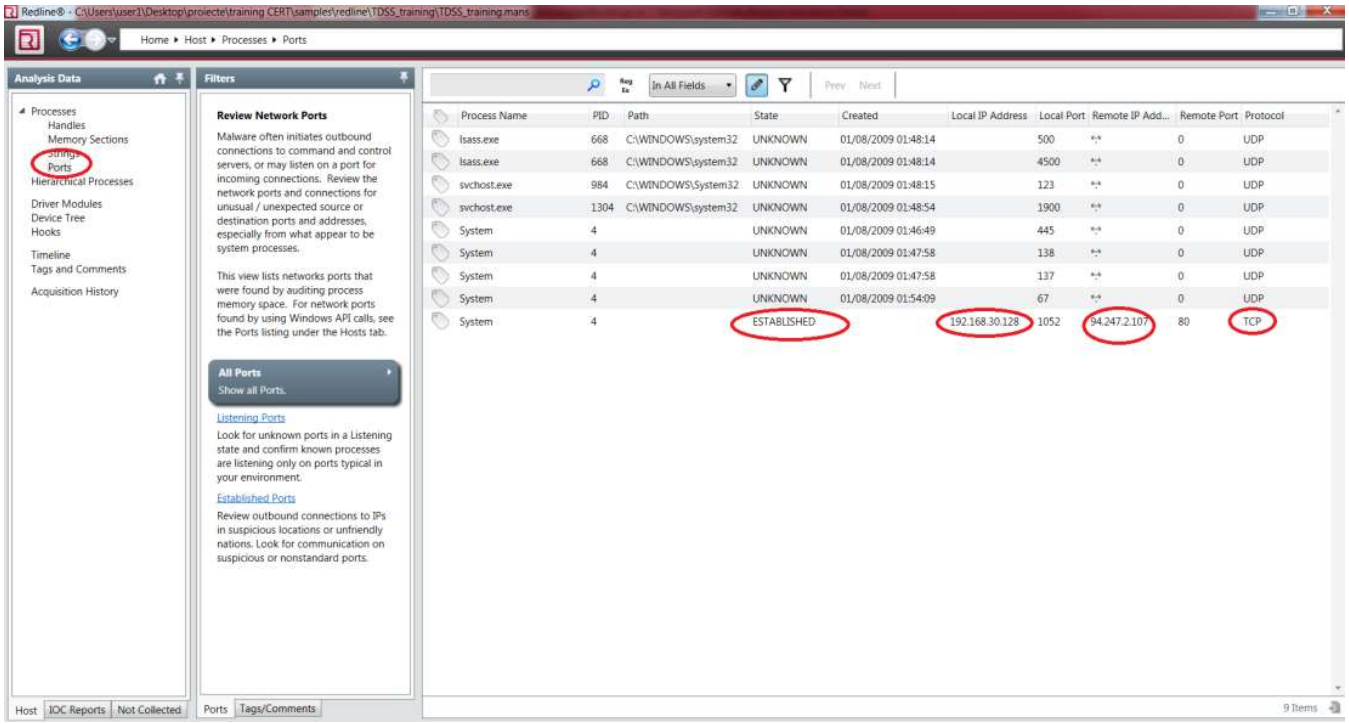
- The process was running from C:\Windows
- Security Identifier for this process appears to be user SID.
- The parent process has been terminated
- The Redline Malware Risck Index is 100/100.

Network Artifacts with Redline (1)



Network Artifacts with Redline (2)

- Processes -> Ports
- Established conections generated by a “system” processs using a non-reserved port
- Classic outbound beaconing connection



Memory analysis with volatility

Memory Analysis – b. concepts

Memory analysis with Redline

Memory analysis with volatility

Volatility framework overview

- Volatility is one of the **best** framework analysing memory images
- It is a command line based and is written completely in Python
- Has a lot of plugins: **malfind**, **apihooks**, **orphanthreads**, etc.
- Supports:

```
Profiles
--------
VistaSP0x64      - A Profile for Windows Vista SP0 x64
VistaSP0x86      - A Profile for Windows Vista SP0 x86
VistaSP1x64      - A Profile for Windows Vista SP1 x64
VistaSP1x86      - A Profile for Windows Vista SP1 x86
VistaSP2x64      - A Profile for Windows Vista SP2 x64
VistaSP2x86      - A Profile for Windows Vista SP2 x86
Win2003SP0x86    - A Profile for Windows 2003 SP0 x86
Win2003SP1x64    - A Profile for Windows 2003 SP1 x64
Win2003SP1x86    - A Profile for Windows 2003 SP1 x86
Win2003SP2x64    - A Profile for Windows 2003 SP2 x64
Win2003SP2x86    - A Profile for Windows 2003 SP2 x86
Win2008R2SP0x64  - A Profile for Windows 2008 R2 SP0 x64
Win2008R2SP1x64  - A Profile for Windows 2008 R2 SP1 x64
Win2008SP1x64    - A Profile for Windows 2008 SP1 x64
Win2008SP1x86    - A Profile for Windows 2008 SP1 x86
Win2008SP2x64    - A Profile for Windows 2008 SP2 x64
Win2008SP2x86    - A Profile for Windows 2008 SP2 x86
Win2012R2x64     - A Profile for Windows Server 2012 R2 x64
Win2012x64       - A Profile for Windows Server 2012 x64
Win7SP0x64       - A Profile for Windows 7 SP0 x64
Win7SP0x86       - A Profile for Windows 7 SP0 x86
Win7SP1x64       - A Profile for Windows 7 SP1 x64
Win7SP1x86       - A Profile for Windows 7 SP1 x86
Win8SP0x64       - A Profile for Windows 8 SP0 x64
Win8SP0x86       - A Profile for Windows 8 SP0 x86
Win8SP1x64       - A Profile for Windows 8.1 x64
Win8SP1x86       - A Profile for Windows 8 SP1 x86
WinXPSP1x64      - A Profile for Windows XP SP1 x64
WinXPSP2x64      - A Profile for Windows XP SP2 x64
WinXPSP2x86      - A Profile for Windows XP SP2 x86
WinXPSP3x86      - A Profile for Windows XP SP3 x86
```

Volatility Plugins (examples)

Volatility plugins			
apihooks	Find API hooks	procexedump	Dump a process to an executable file sample
connections	Print list of open connections	procmemdump	Dump a process to an executable memory sample
dlllist	Print list of loaded dlls for each process	pslist	print all running processes by following the EPROCESS lists
dlldump	Dump a DLL from a process address space	orphanthread	Locate hidden threads
files	Print list of open files for each process	mutantscan	Scan for mutant objects KMUTANT
getsids	Print the SIDs owning each process	pstree	Print process list as a tree
malfind	Find hidden and injected code	sockets	Print list of open sockets

Complete list: https://code.google.com/p/volatility/wiki/Plugins

How to use volatility (Help!)

- **The *-h* flag gives configuration information in Volatility**
  - Used alone it identifies the version, currently loaded plugins, and common parameters

- **Use -h with a plugin to get details and plugin-specific usage**

How to use volatility (2)

- **vol.py –f [image] [plugin] --profile=[PROFILE]**

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img pslist --profile=WinXPSP2x86
Volatility Foundation Volatility Framework 2.4
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x823c89c8 System                    4      0     50      259 ------      0
0x81e02da0 smss.exe                524      4      3       19 ------      0 2009-07-20 23:43:26 UTC+0000
0x82273020 csrss.exe               580    524     11      362      0      0 2009-07-20 23:43:31 UTC+0000
0x82147cf0 winlogon.exe            604    524     16      429      0      0 2009-07-20 23:43:33 UTC+0000
0x81dd6c70 services.exe            648    604     16      254      0      0 2009-07-20 23:43:36 UTC+0000
0x81dd53a0 lsass.exe               660    604     21      334      0      0 2009-07-20 23:43:36 UTC+0000
0x81fb5a28 vmacthlp.exe            812    648      1       25      0      0 2009-07-20 23:43:38 UTC+0000
0x821331f8 svchost.exe             856    648     18      191      0      0 2009-07-20 23:43:40 UTC+0000
```

- **you can set an environment variable to replace –f [image]**

export VOLATILITY_LOCATION=file://<file path>

vol.py pslist --profile=[PROFILE]

Image identification

- **Imageinfo**
  - Recover metadata from a memory image
  - vol.py –f memory.img imageinfo

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img imageinfo
Volatility Foundation Volatility Framework 2.4
Determining profile based on KDBG search...

          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/forensic/Desktop/samples/memory samples/sobig.img)
                      PAE type : PAE
                           DTB : 0x2f0000L
                          KDBG : 0x80545ae0
          Number of Processors : 1
     Image Type (Service Pack) : 3
                KPCR for CPU 0 : 0xffdff000
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2009-07-27 23:28:10 UTC+0000
     Image local date and time : 2009-07-27 19:28:10 -0400
```

Hibernation File Conversion
imagecopy

Purpose

- Convert crash dumps and hibernation files to raw memory images

Important Parameters

- Output file name (-O)
- Make sure to provide correct image OS via (--profile=).

Investigative Notes

- Uncompress Windows hibernation files
- Convert crashdump files to raw images
- Live firewire session data can also be converted

Identify rogue processes
pslist

Purpose

- Print all running processes by following the EPROCESS linked list

Important Parameters

- Show information for specific process IDs (-p)

Investigative Notes

- Provides the binary name (Name), parent process (PPID), and time started (Time)
- Thread (Thds) and Handle (Hnds) counts can be reviewed for anomalies
- Rootkits can unlink malicious processes from the linked list, rendering them invisible to this tool

Identify suspect processes
psscan

Scan physical memory for EPROCESS pool allocations
Hidden processes may be identified
Identify processes no longer running

pslist did not found the *dllhost.exe* process

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img pslist
Volatility Foundation Volatility Framework 2.4
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------
0x823c89c8 System                    4      0     50      259 ------      0
0x81e02da0 smss.exe                524      4      3       19 ------      0 2009-07-20 23:43:26 UTC+0000
0x82273020 csrss.exe               580    524     11      362      0      0 2009-07-20 23:43:31 UTC+0000
0x82147cf0 winlogon.exe            604    524     16      429      0      0 2009-07-20 23:43:33 UTC+0000
0x81dd6c70 services.exe            648    604     16      254      0      0 2009-07-20 23:43:36 UTC+0000
0x81dd53a0 lsass.exe               660    604     21      334      0      0 2009-07-20 23:43:36 UTC+0000
0x81fb5a28 vmacthlp.exe            812    648      1       25      0      0 2009-07-20 23:43:38 UTC+0000
0x821331f8 svchost.exe             856    648     18      191      0      0 2009-07-20 23:43:40 UTC+0000
0x8212f658 svchost.exe             944    648     11      233      0      0 2009-07-20 23:43:41 UTC+0000
0x81db9020 svchost.exe            1036    648     60     1126      0      0 2009-07-20 23:43:42 UTC+0000
0x81db5a58 svchost.exe            1128    648     12      120      0      0 2009-07-20 23:43:42 UTC+0000
0x81dba768 svchost.exe            1268    648     14      188      0      0 2009-07-20 23:43:44 UTC+0000
0x81f8d020 spoolsv.exe            1468    648     13      120      0      0 2009-07-20 23:43:47 UTC+0000
0x81f8b540 explorer.exe           1560   1544     17      535      0      0 2009-07-20 23:43:48 UTC+0000
0x820fdda0 VMwareTray.exe         1908   1560      1       29      0      0 2009-07-20 23:43:51 UTC+0000
0x81d8cab0 VMwareUser.exe         1928   1560      8      117      0      0 2009-07-20 23:43:52 UTC+0000
0x81d87870 VMwareService.e        1388    648      3      146      0      0 2009-07-20 23:44:04 UTC+0000
0x81d6e798 alg.exe                 436    648      6      106      0      0 2009-07-20 23:44:18 UTC+0000
0x81f7a020 wscntfy.exe             544   1036      1       28      0      0 2009-07-20 23:44:19 UTC+0000
0x81f3a518 wpabaln.exe            1244    604      1       58      0      0 2009-07-20 23:45:47 UTC+0000
0x81db0530 msiexec.exe             708    648      8      226      0      0 2009-07-27 20:24:43 UTC+0000
0x82128020 cmd.exe                1236   1560      1       33      0      0 2009-07-27 20:25:58 UTC+0000
0x81f56020 winppr32.exe           1624   1636      2       55      0      0 2009-07-27 23:27:44 UTC+0000
0x81fa7c30 Memoryze.exe           1856   1236      4      156      0      0 2009-07-27 23:28:06 UTC+0000
forensic@ubuntu:~/Desktop/samples/memory samples$
```

psscan found the dllhost.exe process most likely because it was terminated but lingering in unallocated memory space.

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img psscan
Volatility Foundation Volatility Framework 2.4
Offset(P)          Name                PID   PPID PDB        Time created
------------------ ---------------- ------ ------ ---------- ------------------------------
0x0000000001ef1588 dllhost.exe        1124    244 0x033801a0 2009-07-20 23:35:24 UTC+0000
0x0000000001f6e798 alg.exe             436    648 0x040001c0 2009-07-20 23:44:18 UTC+0000
0x0000000001f87870 VMwareService.e    1388    648 0x04000220 2009-07-20 23:44:04 UTC+0000
0x0000000001f8cab0 VMwareUser.exe     1928   1560 0x04000200 2009-07-20 23:43:52 UTC+0000
0x0000000001fb0530 msiexec.exe         708    648 0x040002e0 2009-07-27 20:24:43 UTC+0000
0x0000000001fb5a58 svchost.exe        1128    648 0x04000160 2009-07-20 23:43:42 UTC+0000
0x0000000001fb9020 svchost.exe        1036    648 0x04000140 2009-07-20 23:43:42 UTC+0000
```

Analyzing Process Objects
dlllist

- Display the loaded DLLs and the command line used to start each process
- Show information for specific process IDs
- The command line displayed for the process provides full path information of where the executables was located and what parameters were used to load it
- The base offset provided can be used to extract a specific DLL with dlldump.

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img dlllist -p 1624
Volatility Foundation Volatility Framework 2.4
************************************************************************
winppr32.exe pid:   1624
Command line : C:\WINDOWS\winppr32.exe /sinc
Service Pack 3

Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x00400000    0x20000     0xffff C:\WINDOWS\winppr32.exe
0x7c900000    0xaf000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x7e410000    0x91000     0xffff C:\WINDOWS\system32\user32.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
0x71b20000    0x12000        0x1 C:\WINDOWS\system32\MPR.dll
0x77dd0000    0x9b000       0x7e C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x92000       0x28 C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000       0x1f C:\WINDOWS\system32\Secur32.dll
0x7e1e0000    0xa2000        0x1 C:\WINDOWS\system32\urlmon.dll
```

During our memory analysis with Redline we identified a suspicious process named winppr32.exe. Now, we can obtain more information about that process.

Analyzing Process Objects
getsids

- Display security identifiers (SIDs) for each process
- Can be useful to determine how a process was spawned and with what permissions.

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img getsids -p 1624
Volatility Foundation Volatility Framework 2.4
winppr32.exe (1624): S-1-5-21-1993962763-1547161642-299502267-1003 (Owner)
winppr32.exe (1624): S-1-5-21-1993962763-1547161642-299502267-513 (Domain Users)
winppr32.exe (1624): S-1-1-0 (Everyone)
winppr32.exe (1624): S-1-5-32-544 (Administrators)
winppr32.exe (1624): S-1-5-32-545 (Users)
winppr32.exe (1624): S-1-5-4 (Interactive)
winppr32.exe (1624): S-1-5-11 (Authenticated Users)
winppr32.exe (1624): S-1-5-5-0-57005 (Logon Session)
winppr32.exe (1624): S-1-2-0 (Local (Users with the ability to log in locally))
```

The suspicious process has 2 user SIDs associated with it and this tell us that the process was likely spawned from a user context and hence is unlikely to be a true system process.

Analyzing Process Objects
malfind

- Scans process memory sections looking for indications of code injection and extract them for further analysis.
- You may see multiple injected sections within the same process
- Dumped sections can be reverse engineered or sent to A/V

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img  malfind --dump-dir /home/forensic/Desktop/samples/memory\ samples/output_dir/
Volatility Foundation Volatility Framework 2.4
Process: csrss.exe Pid: 580 Address: 0x7f6f0000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x7f6f0000  c8 00 00 00 60 01 00 00 ff ee ff ee 08 70 00 00   ....`........p..
0x7f6f0010  08 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00   ................
0x7f6f0020  00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f   ................
0x7f6f0030  03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x7f6f0000 c8000000          ENTER 0x0, 0x0
0x7f6f0004 60                PUSHA
0x7f6f0005 0100              ADD [EAX], EAX
0x7f6f0007 00ff              ADD BH, BH
0x7f6f0009 ee                OUT DX, AL
0x7f6f000a ff                DB 0xff
0x7f6f000b ee                OUT DX, AL
0x7f6f000c 087000            OR [EAX+0x0], DH
```

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img  malfind --dump-dir /home/forensic/Desktop/samples/memory\ samples/output_dir/ | grep Process
Volatility Foundation Volatility Framework 2.4
Process: csrss.exe Pid: 580 Address: 0x7f6f0000
Process: winlogon.exe Pid: 604 Address: 0x28a0000
Process: winlogon.exe Pid: 604 Address: 0x24e40000
Process: winlogon.exe Pid: 604 Address: 0x54550000
Process: winlogon.exe Pid: 604 Address: 0x5c840000
Process: winlogon.exe Pid: 604 Address: 0x67fa0000
```

Six injected sections in this image memory

Rootkit Detection
psxview

- Performs a cross-view analysis using six different process listing plugins to visually identify hidden processes.
- It is important to know the output differences between each source:
  - An entry not found by pslist is often a hidden process
  - Processes terminated may only show in psscan column

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img  psxview
Volatility Foundation Volatility Framework 2.4
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x01fd53a0 lsass.exe               660 True   True   True     True   True  True    True
0x01fd6c70 services.exe            648 True   True   True     True   True  True    True
0x021a7c30 Memoryze.exe           1856 True   True   True     True   True  True    True
0x02328020 cmd.exe                1236 True   True   True     True   True  True    True
0x0232f658 svchost.exe             944 True   True   True     True   True  True    True
0x02156020 winppr32.exe           1624 True   True   True     True   True  True    True
0x021b5a28 vmacthlp.exe            812 True   True   True     True   True  True    True
0x01f87870 VMwareService.e        1388 True   True   True     True   True  True    True
0x0218d020 spoolsv.exe            1468 True   True   True     True   True  True    True
0x0217a020 wscntfy.exe             544 True   True   True     True   True  True    True
0x01fb0530 msiexec.exe             708 True   True   True     True   True  True    True
0x02347cf0 winlogon.exe            604 True   True   True     True   True  True    True
0x01fba768 svchost.exe            1268 True   True   True     True   True  True    True
0x01fb5a58 svchost.exe            1128 True   True   True     True   True  True    True
0x022fdda0 VMwareTray.exe         1908 True   True   True     True   True  True    True
0x0218b540 explorer.exe           1560 True   True   True     True   True  True    True
0x023331f8 svchost.exe             856 True   True   True     True   True  True    True
0x01f6e798 alg.exe                 436 True   True   True     True   True  True    True
0x01f8cab0 VMwareUser.exe         1928 True   True   True     True   True  True    True
0x0213a518 wpabaln.exe            1244 True   True   True     True   True  True    True
0x01fb9020 svchost.exe            1036 True   True   True     True   True  True    True
0x02473020 csrss.exe               580 True   True   True     True   False True    True
0x02002da0 smss.exe                524 True   True   True     True   False False   False
0x025c89c8 System                    4 True   True   True     True   False False   False
0x03be2518 wpabaln.exe            1244 False  True   False    False  False False   False
0x1cf5d020 winppr32.exe           1624 False  True   False    False  False False   False
0x18e941f8 svchost.exe             856 False  True   False    False  False False   False
```

Analyzing Process Objects: ***handles*** (1)

Purpose

- Print list of handles opened by the process

Important Parameters

- Operate only on these process IDs (-p PID)
- Show only handles of a certain type (-t type)

Investigative Notes

- Each process can have hundreds or even thousands of handles; reviewing them can be like searching for a needle in a haystack
- Limit your search by looking at specific types (-t) of handles
- Least Frequency of Occurrence counts in Redline make analysis more feasible

More commonly they are reviewed for specific processes that are already suspected of being malicious.

Analyzing Process Objects: ***handles*** (2)

The available handle types are:

- Process
- Thread
- Key
- Event
- File
- Mutant
- Semaphore
- Token
- WmiGuid
- Port
- Thread
- Directory
- WindowStation
- IOCompletion
- Timer

Analyzing Process Objects: *svscan*

Purpose

- Scan memory for Windows service records, giving information on associated processes and drivers

Important Parameters

- None

Investigative Notes

- A vast amount of malware uses a Window Service as a persistence mechanism
- Drivers can be loaded via a service, hence evidence of malicious drivers can also be found using this plug in
- Can identify processes stopped by malware (i.e. Wuauserv)
- Redline does not have the capability to enumerate Services

Analyzing Process Objects: ***cmdscan & consoles***

Purpose

- Scan csrss.exe (XP) or conhost.exe (Win 7) for Command_History and Console_Information residue

Important Parameters

- None

Investigative Notes

- Gathering command history and console output can give insight into user I attacker activities
- **Cmdscan** provides information from the command history buffer
- **consoles** prints commands (inputs) + screen buffer (outputs)
- Plugins can identify info from active *and* closed sessions

Analyzing Process Objects: ***sockets*** & ***sockscan***

Purpose

- Walk linked list of sockets (sockets plugin)
- Scan memory image to find closed or unlinked socket structures (**sockscan pluqin**)

Important Parameters

- None

Investigative Notes

- Socket structures maintain a creation time
- Run both plugins and compare results to separate active and closed socket information
- Pay close attention to the PID attached to the connection. Should that process be listening on that port/protocol?

Analyzing Process Objects: ***driverscan*** (1)

Purpose

- Scans the memory image for both currently and previously loaded driver modules in the kernel

Important Parameters

- None

Investigative Notes

- Provides a list of loaded drivers, their size and location
- Drivers are a common means for malware to take control; loading a driver gives complete access to kernel objects
- Identifying a bad driver amongst hundreds of others can be hard. Other information like hooks may help

Analyzing Process Objects: ***apihooks***

Purpose

- Detect inline and Import Address Table function hooks used by rootkits to modify and control information returned.

Important Parameters

- Operate only on these process IDs (-p PID)
- Skip kernel mode checks (-R)
- Only scan critical processes and dlls (-q)

Investigative Notes

- A large number of legitimate hooks can exist, weeding them out takes practice and an eye for looking for anomalies
- This plug-in can take a long time to run due to the sheer number of locations it must query - be patient!

Analyzing Process Objects: *ssdt*

Purpose

- Display hooked functions within the System Service Description Table (Windows kernel hooking)

Important Parameters

- None

Investigative Notes

- A large number of legitimate hooks exist in the kernel
  - Eliminate those from ntoskrnl.exe and win32k.sys using **l egrep -v '(ntoskrnl l win32k)'**
- The plugin **ssdt_ex** ignores ntoskrnl and win32k hooks, dumps hooking drivers, and readies files for disassembly

Acquiring Processes and Drivers Plugins

dlldump

- Dump DLLs from a process

moddump

- Dump a kernel driver to an executable file sample

procmemdump

- Dump a process to an executable file sample

memdump

- Dump all addressable memory for a process into one file

Registry Analysis Plugins

hivelist

- Find and display the list of available registry hives

printkey

- Print a registry key, its subkeys and values

userassist

- Parse userassist registry keys

hivedump

- Recursively print all keys in a registry hive

hashdump

- Dumps passwords hashes (LM/NTLM) from memory

Registry Analysis
userassist

Purpose

- Find parse and display the userassist key for all memory mapped registry keys

Important Parameters

- Specific the full path of the userassist key (- K “userassist”)
  - Not necessary by default
- Search only in the hive at *offset* (- o virtual address offset)

Investigative Notes

- The Userassist registry key gives evidence of application execution via Windows Explorer - including the last time and execution count
- Parses both versions of userassist (XPand Windows 7)
- Auto-decodes the ROT-13cipher and replaces Win7 GUIDs with friendly folder names

Memory Timelining
timeliner

Purpose

- Timeliner collect timestamps from memory artifacts and outputs then in a timeline format

Important Parameters

- Send output to a delimited file (--output-file=*file_name)*

Investigative Notes

- Compatible with XPand Win7: automatically adjusts helper plugins
- Output can voluminous; best practice is to use"--output-file"
- The output is not currently compatible with other timeline formats
- Timeliner cantake hours to run - be patient!
- The "-h" help information currently lists many incorrect options

Acquiring DLLs
dlldump

- Extract DLL files belonging to a specific process or group of processes
- Use – p (PID), -r (DLLs matching a REGEX name pattern) or –b (specific offset) to limit the number of DLLs extracted.
- Since many processes point to the same DLLs you may encounter multiple copies of the same DLL extracted.

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img dlldump -p 1624 --dump-dir=/home/forensic/Desktop/samples/memory\ samples/output_dir
Volatility Foundation Volatility Framework 2.4
Process(V) Name                Module Base Module Name          Result
---------- ------------------- ----------- -------------------- ------
0x81f56020 winppr32.exe        0x000400000 winppr32.exe         Error: VirtualAddress f6000000 is past the end of image. Try -u/--unsafe
0x81f56020 winppr32.exe        0x07c900000 ntdll.dll            OK: module.1624.2156020.7c900000.dll
0x81f56020 winppr32.exe        0x0773d0000 comctl32.dll         OK: module.1624.2156020.773d0000.dll
0x81f56020 winppr32.exe        0x076f60000 WLDAP32.dll          OK: module.1624.2156020.76f60000.dll
0x81f56020 winppr32.exe        0x05ad70000 uxtheme.dll          OK: module.1624.2156020.5ad70000.dll
0x81f56020 winppr32.exe        0x076fc0000 rasadhlp.dll         OK: module.1624.2156020.76fc0000.dll
0x81f56020 winppr32.exe        0x077dd0000 ADVAPI32.dll         OK: module.1624.2156020.77dd0000.dll
0x81f56020 winppr32.exe        0x077fe0000 Secur32.dll          OK: module.1624.2156020.77fe0000.dll
0x81f56020 winppr32.exe        0x077c00000 VERSION.dll          OK: module.1624.2156020.77c00000.dll
0x81f56020 winppr32.exe        0x077f60000 SHLWAPI.dll          OK: module.1624.2156020.77f60000.dll
0x81f56020 winppr32.exe        0x071a50000 mswsock.dll          OK: module.1624.2156020.71a50000.dll
0x81f56020 winppr32.exe        0x077e70000 RPCRT4.dll           OK: module.1624.2156020.77e70000.dll
0x81f56020 winppr32.exe        0x071ab0000 WS2_32.dll           OK: module.1624.2156020.71ab0000.dll
0x81f56020 winppr32.exe        0x071ad0000 WSOCK32.dll          OK: module.1624.2156020.71ad0000.dll
0x81f56020 winppr32.exe        0x0774e0000 ole32.dll            OK: module.1624.2156020.774e0000.dll
0x81f56020 winppr32.exe        0x07e410000 user32.dll           OK: module.1624.2156020.7e410000.dll
0x81f56020 winppr32.exe        0x077120000 OLEAUT32.dll         OK: module.1624.2156020.77120000.dll
0x81f56020 winppr32.exe        0x071b20000 MPR.dll              OK: module.1624.2156020.71b20000.dll
0x81f56020 winppr32.exe        0x071aa0000 WS2HELP.dll          OK: module.1624.2156020.71aa0000.dll
0x81f56020 winppr32.exe        0x076fb0000 winrnr.dll           OK: module.1624.2156020.76fb0000.dll
0x81f56020 winppr32.exe        0x07c800000 kernel32.dll         OK: module.1624.2156020.7c800000.dll
0x81f56020 winppr32.exe        0x07e1e0000 urlmon.dll           OK: module.1624.2156020.7e1e0000.dll
0x81f56020 winppr32.exe        0x077c10000 msvcrt.dll           OK: module.1624.2156020.77c10000.dll
0x81f56020 winppr32.exe        0x077f10000 GDI32.dll            OK: module.1624.2156020.77f10000.dll
0x81f56020 winppr32.exe        0x076f20000 DNSAPI.dll           OK: module.1624.2156020.76f20000.dll
```

Acquiring Processes and Drivers
procdump

- Dump a process to an executable memory sample
- Why?
  - Anti-virus scanning engines
  - Malware analysis sandboxes
  - Dynamic malware analysis
  - Static malware debugging and disassembly

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f sobig.img procdump -p 524 --dump-dir=/home/forensic/Desktop/samples/memory\ samples/output_dir
Volatility Foundation Volatility Framework 2.4
Process(V) ImageBase  Name             Result
---------- ---------- ---------------- ------
0x81e02da0 0x48580000 smss.exe         OK: executable.524.exe
```

Network Artifacts
connections & connscan

- Walk linked list of TCP connections (connections plugin)
- Scan memory image to find closed or unlinked TCP connection structures (connscan plugin)
- Run both plugins and compare results to identify active and closed connections
- Pay attention to the PID attached to the connection.

```
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f zeus.img connections
Volatility Foundation Volatility Framework 2.4
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
forensic@ubuntu:~/Desktop/samples/memory samples$ vol.py -f zeus.img connscan
Volatility Foundation Volatility Framework 2.4
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------ ------------------------ ---
0x02214988 172.16.176.143:1054       193.104.41.75:80          856
0x06015ab0 0.0.0.0:1056              193.104.41.75:80          856
forensic@ubuntu:~/Desktop/samples/memory samples$
```

Quick Recap

- ☑ Redline basics
- ☑ Memory Analysis basics
- ☑ Volatility usage