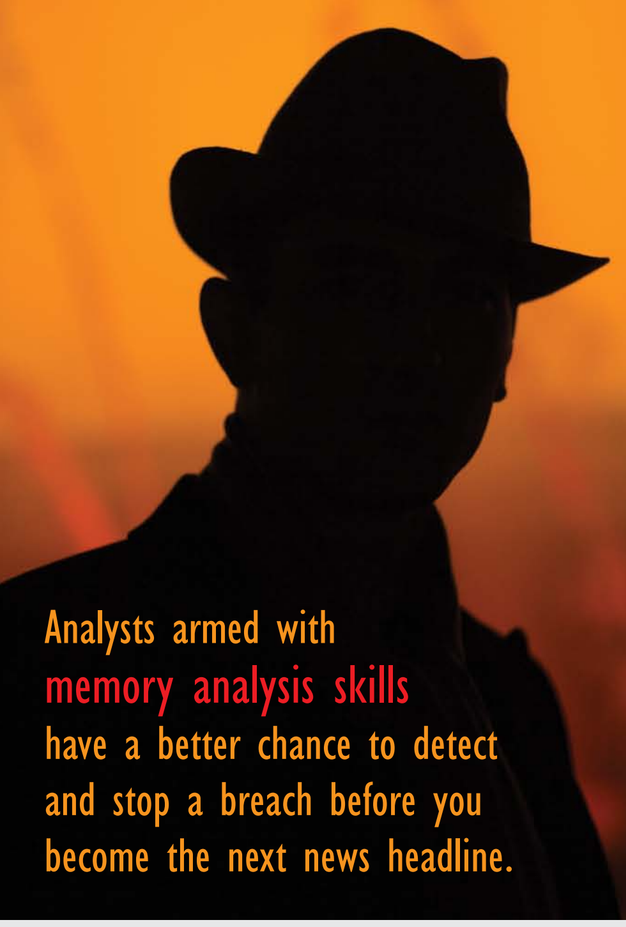


Analysts armed with memory analysis skills have a better chance to detect and stop a breach before you become the next news headline.

Stop being blind to the adversary – Arm your incident response team today!

Memory Analysis will prepare your team to:

- Discover zero-day malware
- Detect compromises
- Uncover evidence that others miss

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPONSE

Memory Forensics POSTER

Malware Can Hide, But It Must Run

digital-forensics.sans.org

SPRING 2015 – 36TH EDITION – $25.00

SANS DFIR CURRICULUM

CORE

FOR408 Windows Forensics **GCFE**

SEC504 Hacker Techniques, Exploits, and Incident Handling **GCIH**

IN-DEPTH INCIDENT RESPONSE

FOR508 Advanced Incident Response **GCFA**

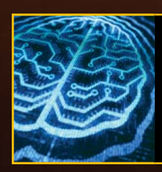
FOR572 Advanced Network Forensics and Analysis

FOR610 REM: Malware Analysis **GREM**

SPECIALIZATION

MGT535 Incident Response Team Management

FOR518 Mac Forensics



FOR526 Memory Forensics In-Depth

FOR585 Advanced Smartphone Forensics

DFIR Blog digital-forensics.sans.org/blog

DFIR Twitter @sansforensics

DFIR Facebook sansforensics

DFIR Google+ gplus.to/sansforensics

DFIR Mailing List dfir.to/MAIL-LIST

FIND EVIL WHERE IT LIES

Memory Forensics

Essential in Effective Incident Response Today

sans.org/FOR526

Live Memory Analysis with Rekall

Rekall

rekall-forensic.com

The Rekall Memory Forensic Framework is a collection of memory acquisition and analysis tools implemented in Python under the GNU General Public License. It originated in 2011 as the "Technology Preview" branch of the Volatility Framework, with goals of streamlining code and improving efficiency, performance and usability. Code differences over years of development made it difficult to remerge the Volatility Framework with this rapidly developing branch, so the developers deemed it necessary to fork the project in Dec 2013 The Rekall Framework has been included in the development of Google Rapid Response, a live enterprise IR/forensics triage tool.

Some of the key differences that most analysts notice with Rekall is its ease of use, as it does not require the specification of a Windows target system profile when invoking a plugin. Rekall uses an alternative means of deciphering the profile of the Windows system other than reading the KDBG (Kernel Debugging Data Block). Rekall also uses interactive analysis sessions that cache information in memory, allowing data to remain available for increased speed in subsequent module analysis. Rekall's most exciting feature is its ability to work with winpmem for LIVE system memory analysis - further reducing the time responders must take in triaging a possibly compromised system.

rekall-forensic.com

```
C:\Program Files\Rekall>winpmem_1.6.2.exe -l
Extracting driver to C:\Users\SANSFO~2\AppData\Local\Temp\pmeDB1F.tmp
Driver Unloaded.
Loaded Driver C:\Users\SANSFO~2\AppData\Local\Temp\pmeDB1F.tmp.
Deleting C:\Users\SANSFO~2\AppData\Local\Temp\pmeDB1F.tmp
CR3: 0x00001AA000
 5 memory ranges:
Start 0x00001000 - Length 0x0009E000
Start 0x00100000 - Length 0x00002000
Start 0x00103000 - Length 0xBFDDD000
Start 0xBFF00000 - Length 0x00100000
Start 0x100000000 - Length 0x40000000
```

1 Load the winpmem driver from an administrative cmdshell

```
C:\Program Files\Rekall>rekal -f \\.\pmem
------------------------------------------------------------------------------
The Rekall Memory Forensic framework 1.2.1 (Col de la Croix).

"We can remember it for you wholesale!"

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License.

See http://www.rekall-forensic.com/docs/Manual/tutorial.html to get started.
------------------------------------------------------------------------------
```

2 Launch Rekall with access to \\.\pmem

```
[1] pmem 17:02:19> pslist
---------------> pslist()
_EPROCESS          Name              PID  PPID  Thds  Hnds  Sess  Wow64  Start
------------------ ----------------- ---- ----- ----- ----- ----- ------ ----------
0xe00102edd900 System              4    0     85    -     -     False  2014-12-30 16:33:20
0xe00104f56040 smss.exe            308  4     2     -     -     False  2014-12-30 16:33:20
0xe00105be8900 svchost.exe         376  532   24    -     0     False  2014-12-30 16:33:23
0xe00104f3d580 csrss.exe           380  372   8     -     0     False  2014-12-30 16:33:21
0xe00105a51680 TabTip32.exe        396  1780  1     -     0     True   2014-12-30 16:36:17
0xe001050bd300 wininit.exe         444  372   1     -     0     False  2014-12-30 16:33:21
0xe00104ead080 csrss.exe           452  436   9     -     0     False  2014-12-30 16:33:21
0xe00105094900 winlogon.exe        492  436   2     -     0     False  2014-12-30 16:33:21
0xe00105ifb080 services.exe        532  444   4     -     0     False  2014-12-30 16:33:22
```

3 Conduct Analysis with Rekall Plugins

How to Parse a Memory Image with the Volatility Framework

```
sansforensics:/cases $ vol.py -f win7crypto.vmem --profile=Win7SP0x86 pslist
Offset(V)  Name            PID  PPID  Thds  Hnds  Sess  Wow64  Start                          Exit
---------- --------------- ---- ----- ----- ----- ----- ------ ------------------------------ ----------
0x84f48bb0 System          4    0     94    425   ------ 0      2012-02-16 12:04:46 UTC+0000
0x86223d40 smss.exe        268  4     2     29    ------ 0      2012-02-16 12:04:46 UTC+0000
0x86a81d40 csrss.exe       360  352   9     489   0      0      2012-02-16 12:04:49 UTC+0000
0x86a19530 wininit.exe     416  352   3     75    0      0      2012-02-16 12:04:50 UTC+0000
0x86a23478 csrss.exe       424  408   9     313   1      0      2012-02-16 12:04:50 UTC+0000
0x86a65530 winlogon.exe    472  408   3     111   1      0      2012-02-16 12:04:50 UTC+0000
0x85960448 services.exe    520  416   14    216   0      0      2012-02-16 12:04:52 UTC+0000
```

1 Identify rogue processes

```
sansforensics:/cases $ vol.py -f win7crypto.vmem --profile=Win7SP0x86 dlllist -p 2748
************************************************************************
TrueCrypt.exe pid:  2748
Command line : "C:\Program Files\TrueCrypt\TrueCrypt.exe"

Base       Size      LoadCount Path
---------- ---------- ---------- ----
0x00400000 0x1a2000  0xffff C:\Program Files\TrueCrypt\TrueCrypt.exe
0x776b0000 0x13c000  0xffff C:\Windows\SYSTEM32\ntdll.dll
0x76980000 0xd4000   0xffff C:\Windows\system32\kernel32.dll
0x759f0000 0x4a000   0xffff C:\Windows\system32\KERNELBASE.dll
0x75a70000 0x84000   0xffff C:\Windows\WinSxS\x86_microsoft.windows.common-
 controls_6595b64144ccf1df_5.82.7600.16385_none_ebf82fc36c758ad5\COMCTL32.dll
0x76610000 0xa0000   0xffff C:\Windows\system32\ADVAPI32.dll
0x76560000 0xac000   0xffff C:\Windows\system32\msvcrt.dll
0x75b80000 0x19000   0xffff C:\Windows\SYSTEM32\sechost.dll
```

2 Analyze process DLLs and handles

3 Review network artifacts

```
sansforensics:/cases $ vol.py -f win7crypto.vmem --profile=Win7SP0x86 netscan
Offset(P)  Proto  Local Address          Foreign Address  State      Pid   Owner          Created
0x3d7338a0 UDPv4  0.0.0.0:58732         *:*                         1100  svchost.exe    2012-02-16 12:43:06
0x3d7338a0 UDPv6  :::58732              *:*                         1100  svchost.exe    2012-02-16 12:43:06
0x3dabaaf8 UDPv4  0.0.0.0:51319         *:*                         1100  svchost.exe    2012-02-16 12:43:26
0x3dabaaf8 UDPv6  :::51319              *:*                         1100  svchost.exe    2012-02-16 12:43:26
0x3db1f720 UDPv6  ::1:1900              *:*                         3084  svchost.exe    2012-02-16 12:43:05
0x3dc23b08 TCPv4  0.0.0.0:445           0.0.0.0:0        LISTENING  4     System
0x3dc23b08 TCPv6  :::445                :::0             LISTENING  4     System
0x3dcec50  UDPv4  127.0.0.1:65096       *:*                         3084  svchost.exe    2012-02-16 12:43:05
0x3df22d70 UDPv4  172.16.246.144:138    *:*                         4     System         2012-02-16 12:43:05
0x3df23570 UDPv4  0.0.0.0:5355          *:*                         1100  svchost.exe    2012-02-16 12:43:05
0x3dffc190 UDPv6  ::1:65095             *:*                         3084  svchost.exe    2012-02-16 12:43:05
0x3e21ae18 UDPv4  0.0.0.0:0             *:*                         1100  svchost.exe    2012-02-16 12:43:26
0x3e21ae18 UDPv6  :::0                  *:*                         1100  svchost.exe    2012-02-16 12:43:26
0x3e22ef50 UDPv4  0.0.0.0:63429         *:*                         1100  svchost.exe    2012-02-16 12:05:02
0x3e22ef50 UDPv6  :::63429              *:*                         1100  svchost.exe    2012-02-16 12:05:02
0x3e2592a0 UDPv6  fe80::2186:5739:c69:4546:1900 *:*                 3084  svchost.exe    2012-02-16 12:43:05
```

4 Look for evidence of code injection

```
sansforensics:/cases $ vol.py -f win7crypto.vmem --profile=Win7SP0x86 malfind -p 3196
Process: iexplore.exe Pid: 3196 Address: 0x5fff0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 16, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x5fff0000  64 74 72 52 00 00 00 00 60 04 ff 5f 00 00 00 00   dtrR....`.._....
0x5fff0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x5fff0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x5fff0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x5fff0000 647472    JZ 0x5fff0075
0x5fff0003 52        PUSH EDX
0x5fff0004 0000      ADD [EAX], AL
0x5fff0006 0000      ADD [EAX], AL
0x5fff0008 60        PUSHA
0x5fff0009 04ff      ADD AL, 0xff
0x5fff000b 5f        POP EDI
0x5fff000c 0000      ADD [EAX], AL
```

5 Check for signs of a rootkit

```
sansforensics:/cases $ vol.py -f be2.vmem ssdt |egrep -v 'ntoskrnl|win32k'
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at ff3aab90 with 284 entries
  Entry 0x0041: 0xff0d2487 (NtDeleteValueKey) owned by 00004A2A
  Entry 0x0047: 0xff0d216b (NtEnumerateKey) owned by 00004A2A
  Entry 0x0049: 0xff0d2267 (NtEnumerateValueKey) owned by 00004A2A
  Entry 0x0077: 0xff0d20c3 (NtOpenKey) owned by 00004A2A
  Entry 0x007a: 0xff0d1e93 (NtOpenProcess) owned by 00004A2A
  Entry 0x0080: 0xff0d1f0b (NtOpenThread) owned by 00004A2A
  Entry 0x0089: 0xff0d2617 (NtProtectVirtualMemory) owned by 00004A2A
  Entry 0x00ad: 0xff0d1da0 (NtQuerySystemInformation) owned by 00004A2A
  Entry 0x00ba: 0xff0d256b (NtReadVirtualMemory) owned by 00004A2A
  Entry 0x00d5: 0xff0d2070 (NtSetContextThread) owned by 00004A2A
  Entry 0x00f7: 0xff0d2397 (NtSetValueKey) owned by 00004A2A
  Entry 0x00fe: 0xff0d201d (NtSuspendThread) owned by 00004A2A
  Entry 0x0102: 0xff0d1fca (NtTerminateThread) owned by 00004A2A
  Entry 0x0115: 0xff0d25c1 (NtWriteVirtualMemory) owned by 00004A2A
```

```
sansforensics:/cases $ vol.py -f be2.vmem moddump -b 0xff0d1000 -D .
Module Base  Module Name        Result
-----------  -------------------  ------
0x0ff0d1000  00004A2A             OK: driver.ff0d1000.sys
```

6 Dump suspicious processes and drivers

Six-Step Investigative Methodology Walkthrough

In-Depth Memory Analysis

Timestamp	Filename	Reason	Attributes
2012-02-16 12:06:00.332	mysecretdata.tc	CREATE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.332	mysecretdata.tc	CREATE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.348	mysecretdata.tc	EXTEND & CREATE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.348	mysecretdata.tc	EXTEND & CREATE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretdata.tc	BASIC_INFO_CHANGE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretdata.tc	CLOSE & BASIC_INFO_CHANGE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretdata.tc	EXTEND & CREATE & CLOSE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretdata.tc	BASIC_INFO_CHANGE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretdata.tc	CLOSE & BASIC_INFO_CHANGE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.426	mysecretda????	EXTEND & CREATE & CLOSE	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.582	mysecretdata.tc	RENAME_OLD_NAME	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.582	mysecretdata.tc	RENAME_NEW_NAME	ARCHIVE & NOT_CONTENT_INDEXED
2012-02-16 12:06:00.582	mysecretdata.tc	RENAME_NEW_NAME & CLOSE	ARCHIVE & NOT_CONTENT_INDEXED

Retrieve USN Journal Entries `USNParser` *by Tom Spencer*

The $USNJournal can hold trace artifacts for files/directories that USED to be present on a volume. Spencer's plugin carves entries from a memory image – awesome!

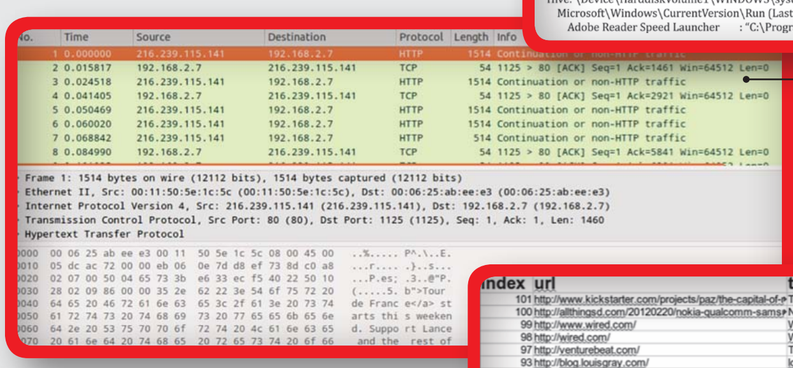
```
$ vol.py -f system_image.vmem --profile=Win7SP1x64 usnparser --output=csv --output-file=usn.csv -CS
```

```
sansforensics:/cases $ vol.py -f cridex.vmem --profile=WinXPSP2x86 autoruns -t autoruns

Autoruns ==========================================

Hive: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
  Software\Microsoft\Windows\CurrentVersion\Run (Last modified: 2012-07-22 02:31:51 UTC+0000)
    KB00207877.exe                : "C:\Documents and Settings\Robert\Application Data\KB00207877.exe" (PIDs: -)

Hive: \Device\HarddiskVolume1\WINDOWS\system32\config\software
  Microsoft\Windows\CurrentVersion\Run (Last modified: 2012-02-18 20:09:37 UTC+0000)
    Adobe Reader Speed Launcher   : "C:\Program Files\Adobe\Reader 9.0\Reader\Reader_sl.exe" (PIDs: 1640)
```

Identify Persistence Mechanisms `AutoRuns` *by Thomas Chopitea*

Identifying persistence mechanisms implemented on a compromised system to reinstantiate a malicious process after a reboot or process termination can be performed in a variety of ways. Using only a memory image of that system, we can extract some of the most common persistence mechanisms used by malware today by invoking Chopitea's autoruns plugin.

```
$ vol.py -f system_image.vmem --profile=Win7SP1x64 autoruns -t autoruns
```



```
No.  Time      Source           Destination      Protocol  Length  Info
  1  0.000000  216.239.115.141  192.168.2.7      HTTP      1514    Continuation or non-HTTP traffic
  2  0.015817  192.168.2.7      216.239.115.141  TCP         54    1125 > 80 [ACK] Seq=1 Ack=1461 Win=64512 Len=0
  3  0.024518  216.239.115.141  192.168.2.7      HTTP      1514    Continuation or non-HTTP traffic
  4  0.041405  192.168.2.7      216.239.115.141  TCP         54    1125 > 80 [ACK] Seq=1 Ack=2921 Win=64512 Len=0
  5  0.050469  216.239.115.141  192.168.2.7      HTTP      1514    Continuation or non-HTTP traffic
  6  0.060020  216.239.115.141  192.168.2.7      HTTP      1514    Continuation or non-HTTP traffic
  7  0.068842  216.239.115.141  192.168.2.7      HTTP       514    Continuation or non-HTTP traffic
  8  0.084990  192.168.2.7      216.239.115.141  TCP         54    1125 > 80 [ACK] Seq=1 Ack=5841 Win=64512 Len=0

Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
Ethernet II, Src: 00:11:50:5e:1c:5c (00:11:50:5e:1c:5c), Dst: 00:06:25:ab:ee:e3 (00:06:25:ab:ee:e3)
Internet Protocol Version 4, Src: 216.239.115.141 (216.239.115.141), Dst: 192.168.2.7 (192.168.2.7)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 1125 (1125), Seq: 1, Ack: 1, Len: 1460
Hypertext Transfer Protocol

0000  00 06 25 ab ee e3 00 11  50 5e 1c 5c 08 00 45 00   ..%..... P^.\..E.
0010  05 dc ac 72 00 00 eb 06  0e 7d d8 ef 73 8d c0 a8   ...r.... .}..s...
0020  02 07 00 50 04 65 73 3b  e6 33 ec f5 40 22 50 10   ...P.es; .3..@"P.
0030  28 02 09 86 00 00 35 2e  62 22 3e 54 6f 75 72 20   (.....5. b">Tour
0040  64 65 20 46 72 61 6e 63  65 3c 2f 61 3e 20 73 74   de Franc e</a> st
0050  61 72 74 73 20 74 68 69  73 20 77 65 65 6b 65 6e   arts thi s weeken
0060  64 2e 20 53 75 70 70 6f  72 74 20 4c 61 6e 63 65   d. Suppo rt Lance
0070  20 61 6e 64 20 74 68 65  20 72 65 73 74 20 6f 66    and the  rest of
```

Extract Network Packets `ethscan` *by Jamaal Speights*

In some investigations, the sole source of network traffic must be carved out of the system memory image. Using Speights' plugin, we are able to extract network packets from memory, with an output option ("-C") of creating a pcap file.

```
$ vol.py -f system_image.vmem --profile=Win7SP1x64 ethscan --save-pcap=out.pcap
```

```
Module   User            Domain           Password
-------- --------------- ---------------- ----------------
-----------------------
wdigest  Marc Lucas      Budweiser        P@ssw0rd

wdigest  BUDWEISER$      WORKGROUP
```

Extract Plaintext Passwords `mimikatz` *by Francesco Picasso*

A huge thanks to Benjamin Delpy for creating the mimikatz plaintext credential harvester that acts on the lsass process to extract usernames and passwords from interactive sessions of the target system. Picasso has given us easy access to this functionality on memory images through his Volatility plugin.

```
$ vol.py -f system_image.vmem --profile=Win7SP1x64 mimikatz
```

index	url	title	visits	typed	last_visit_ti
101	http://www.kickstarter.com/projects/paz/the-capital-of-m	The Capital Of Maine Is Augusta. Also PAZ Is Makin	1		0 2012-02-21 00:21:56
100	http://allthingsd.com/20120220/nokia-qualcomm-samsung	Nokia, Qualcomm, Samsung Gear Up for the Next Round of Patent Fights - In	4		0 2012-02-21 01:31:31
99	http://www.wired.com/	Wired.com	1		0 2012-02-20 23:47:56
98	http://wired.com/	Wired.com	1		0 2012-02-20 23:47:56
97	http://venturebeat.com/	Tech News | Innovation News | Money News | VentureBeat	1		0 2012-02-20 23:54:57
93	http://blog.louisgray.com/	louisgray.com	1		0 2012-02-20 23:55:46
95	http://www.alleyinsider.com/	SAI	1		0 2012-02-20 23:47:47
96	http://www.businessinsider.com/sai	SAI	1		0 2012-02-20 23:47:47
94	http://valleywag.com/	Gawker — Today's gossip is tomorrow's news	2		0 2012-02-20 23:47:47
92	http://scobleizer.com/	Scobleizer — Searching for world-changing technology	1		0 2012-02-20 23:47:43
91	http://lifehacker.com/	Lifehacker, tips and downloads for getting things done	1		0 2012-02-20 23:52:50
89	http://www.cnet.com/	Product reviews and prices, software downloads, and tech news - CNET	1		0 2012-02-20 23:47:37
88	http://cnet.com/	Product reviews and prices, software downloads, and tech news - CNET	1		0 2012-02-20 23:47:37
90	http://louisgray.com/		1		0 2012-02-20 23:47:38
87	http://gizmodo.com/	Gizmodo, the Gadget Guide	1		0 2012-02-20 23:47:35
86	http://news.cnet.com/crave/	Crave, the gadget blog - CNET	1		0 2012-02-20 23:47:32
84	http://www.boygeniusreport.com/	BGR: The Three Biggest Letters In Tech	1		0 2012-02-20 23:47:30

Reconstruct Browser History `Chrome/Mozilla` *by John Lassalle*

Browser forensics may reveal suspicious web activity, and there are multiple tools that examiners can use to reconstruct browser artifacts from disk. What is the gain in reconstructing browser history from memory? One case is when the browser history is trapped in a hibernation file but has since been deleted by the user (this may even indicate intent). Lassalle's Chrome and Mozilla plugins grant easy access to these artifacts.

```
$ vol.py -f system_image.vmem --profile=Win7SP1x64 firefoxhistory --output=csv --output-file=firefox.csv
```

SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPONSE

Memory Forensics

MALWARE CAN HIDE, BUT IT MUST RUN

This poster shows some of the structures analyzed during memory forensic investigations. Just as those practicing disk forensics benefit from an understanding of filesystems, memory forensic practitioners also benefit from an understanding of OS internal structures. The internal structures detailed in the poster are the most important in most investigations, but by no means are they complete. Similarly, each structure has far more members than are shown on the poster. Some structures have hundreds of members. We have again chosen to show those that are most useful to our investigations.

FOR526
Memory Forensics In-Depth
AUTHORS:

Alissa Torres *@sibertor*
Jake Williams *@malwarejake*

Memory analysis is now a crucial skill for any incident responder who is analyzing intrusions. The malware paradox is key to understanding that while intruders are becoming more advanced with anti-forensic tactics and techniques, it is impossible to hide their footprints completely from a skilled incident responder performing memory analysis.

Learn more about **FOR526: Memory Forensics In-Depth** at **sans.org/FOR526**

_EPROCESS

The _EPROCESS is perhaps the most important structure in memory forensics. As opposed to the KDBG (used only by Volatility), it is also used by Rekall. The _EPROCESS structure has more than 100 members, many of them pointers to other structures.

The _EPROCESS gives us the PID and parent PID of a given process. Analyzing PID relationships between processes can reveal malware. For more information, see the SANS DFIR poster "Know Normal, Find Evil."

The _EPROCESS block also contains the creation and exit time of a process. Why would the OS keep track of exited processes? The answer is that when a process exits, it may have open handles which must be closed by the OS. The OS also needs time to gracefully deallocate other structures used by the process. The ExitTime field allows us to see that a process has exited but has not yet been completely removed by the OS. Note that the task manager and other live response tools will not show exited processes at all, but they are easy to see with the use of memory forensics!

VAD

VADs (Virtual Address Descriptors) are used by the memory manager to track ALL memory allocated on the system. Malware and rootkits can hide from a lot of different OS components, but hiding from the memory manager is unwise. If it can't see your memory, it will give it away!

Unloaded Modules

The Windows OS keeps track of recently unloaded kernel modules (device drivers). This is useful for finding rootkits (and misbehaving legitimate device drivers).

Process Environment Block

The PEB contains pointers to the _PEB_LDR_DATA structure (discussed below). It also contains a flag that tells whether a debugger is attached to a process. Some malware will debug a child process as an anti-reversing measure. Finally, the PEB also contains a pointer to the command line arguments that were supplied to the process on creation.

PLUGINS: `modules`, `ldrmodules`, `dlllist`, `pstree -v`

_MMVAD

- **LeftChild** – Pointer to the left VAD child
- **RightChild** – Pointer to the right VAD child
- **StartingVpn** – Starting address described by VAD
- **EndingVpn** – Ending address described by VAD
- **VadsProcess** – Pointer to the _EPROCESS block that owns this VAD

Process Struct (_EPROCESS)

- **Pcb** – Process control block
- **CreateTime** – Time when the process was started.
- **ExitTime** – Exit time of the process – process is still stored in the process list for some time after it exits. It allows for graceful deallocation of other process structures.
- **UniqueProcessId** – PID of the process
- **ActiveProcessLinks** – Doubly linked list to other process' EPROCESS structures (process list)
- **ObjectTable** – Pointer to the process' handle table
- **Peb** – Pointer to the process environment block
- **InheritedFromUniqueProcessId** – The parent PID
- **ThreadListHead** – List of active threads (_ETHREAD)
- **VadRoot** – Pointer to the root of the VAD tree

System Process DTB (directory table base)

The directory table base of a process points to the base of the page directory table (sometimes called the page directory base, or PDB). The CR3 register points to this location, which is unique per process. From the DTB, the complete list of the processes' page tables can be discovered. Rekall locates the DTB for the Idle process (the Idle process is really just an accounting structure) and then uses this to find the image base of the kernel. Then, the KDBG (if needed at all) can be found deterministically, rather than using the scanning approach to find the KDBG used by Volatility. From the Idle process DTB, all other required structure offsets can be determined.

Unloaded Drivers

- **Name** – Driver name
- **StartAddress** –Start address where driver was loaded
- **EndAddress** – End address where driver was loaded
- **CurrentTime** – Time when driver was unloaded

Process Environment Block (_PEB)

- **BeingDebugged** – Is a debugger attached to the process
- **ImageBaseAddress** – Virtual address where the executable is loaded
- **Ldr** – Pointer to _PEB_LDR_DATA structure
- **ProcessParameters** – Full path name and command-line arguments

Kernel Debugger Data Block (_KDDEBUGGER_DATA64)

- **PsLoadedModuleList** – Pointer to the list of loaded kernel modules
- **PsActiveProcessHead** – Pointer to the list head of active processes
- **PspCidTable** – Table of processes used by the scheduler
- **MmUnloadedDrivers** – List of recently unloaded drivers

PEB Loader Data (_PEB_LDR_DATA)

- **InLoadOrderModuleList** – List of loaded DLLs
- **InMemoryOrderModuleList** – List of loaded DLLs
- **InInitializationOrderModuleList** – List of loaded DLLs

_LDR_DATA_TABLE_ENTRY

- **DllBase** – The base address of the DLL
- **EntryPoint** – Entry point of the DLL.
- **SizeOfImage** – Size of the DLL in memory
- **FullDllName** – Full path name of the DLL
- **TimeDateStamp** – The compile time stamp for the DLL

PsLoadedModuleList

The PsLoadedModuleList structure of the KDBG points to the list of loaded kernel modules (device drivers) in memory. Many malware variants use kernel modules because they require low level access to the system. Rootkits, packet sniffers, and many keyloggers use may be found in the loaded modules list. The members of the list are _LDR_DATA_TABLE_ENTRY structures. Stuxnet, Duqu, Regin, R2D2, Flame, etc. have all used some kernel mode module component – so this is a great place to look for advanced (supposed) nation-state malware. However, note that some malware has the ability to unlink itself from this list, so scanning for structures may also be necessary.

ThreadListHead

Where are the thread list structures on the poster? Sorry, we just don't have room to do them justice. But most investigations don't require us to dive into thread structures directly. Threads are still important though. In Windows, a process is best thought of as an accounting structure. The Windows scheduler never deals with processes directly, rather it schedules individual threads (inside a process) for execution. Still, you'll find yourself using process structures more in your investigations.

ObjectTable

For a process in Windows to use any resource (registry key, file, directory, process, etc.) it must have a handle to that object. We can tell a lot about a process just by looking at its open handles. For instance, you could potentially infer the log file a keylogger is using or persistence keys used by the malware, all by examining handles.

_LDR_DATA_TABLE_ENTRY

This structure is used to describe a loaded module. Loaded modules come in two forms. The first is the kernel module (aka device driver). The second type of loaded module are dynamic link libraries (DLLs), which are loaded into user mode processes.

PLUGINS: `modules`, `ldrmodules`, `dlllist`

PEB Loader Data

This structure contains pointers to three linked lists of loaded modules in a given process. Each is ordered differently (order of loading, order of initialization, and order of memory addresses). Sometimes malware will inject a DLL into a legitimate Windows service and then try to hide. But they'd better hide from all three lists or you'll detect it with no trouble.

PLUGINS: `ldrmodules`

Note that many internal OS structures are doubly linked lists. The pointers in the lists actually point to the pointer in the next structure. However, for clarity of illustration, we have chosen to show the type of structure they point to. Also, note that the PsActiveProcessHead member of the KDBG structure points to ActiveProcessLinks member of the _EPROCESS structure. However, for clarity we depict the pointer pointing to the base of the _EPROCESS structure. We feel that this depiction more clearly illustrates the relationship between the various structures.

DFIR

DFIR Resources digital-forensics@sans.org
DFIR Blog digital-forensics.sans.org/blog
DFIR Twitter @sansforensics
DFIR Facebook sansforensics
DFIR Google+ gplus.to/sansforensics
DFIR Mailing List dfir.to/MAIL-LIST