

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



Memory Forensics POSTER

Malware Can Hide, But It Must Run

digital-forensics.sans.org
SPRING 2015 – 361st 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
REKALL:
Malware Analysis
GREM

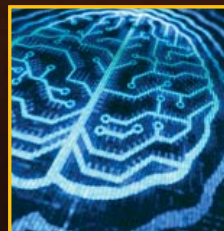
SPECIALIZATION



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-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\pme0B1F.tmp
Driver Unloaded.
Loaded Driver C:\Users\SANSFO-2\AppData\Local\Temp\pme0B1F.tmp
Selecting C:\Users\SANSFO-2\AppData\Local\Temp\pme0B1F.tmp
R3: 0x00001A000
5 memory ranges:
Start: 0x00001000 - Length: 0x00090000
Start: 0x00100000 - Length: 0x00020000
Start: 0x00103000 - Length: 0x00000000
Start: 0xBFF00000 - Length: 0x00100000
Start: 0x100000000 - Length: 0x40000000
```

Load the winpmem driver from an administrative cmdshell

Launch Rekall with access to \\.\pmem

```
C:\Program Files\Rekall>rekall -f \\.\pmem
-----
The Rekall Memory Forensic Framework 1.2.1 (coi 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.
```

Process	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start
System	System	4	0	85	-	-	False	2014-12-30 16:33:20
smss.exe	smss.exe	308	4	2	-	-	False	2014-12-30 16:33:20
svchost.exe	svchost.exe	376	532	24	-	-	False	2014-12-30 16:33:23
csrss.exe	csrss.exe	380	372	8	-	-	False	2014-12-30 16:33:21
tab1p32.exe	tab1p32.exe	396	1780	1	-	0	True	2014-12-30 16:36:12
wininit.exe	wininit.exe	444	372	1	-	0	False	2014-12-30 16:33:21
csrss.exe	csrss.exe	452	436	9	-	0	False	2014-12-30 16:33:21
winlogon.exe	winlogon.exe	492	436	2	-	0	False	2014-12-30 16:33:21
services.exe	services.exe	532	444	4	-	0	False	2014-12-30 16:33:21
services.exe	services.exe	610	711	6	-	0	False	2014-12-30 16:33:21

Conduct Analysis with Rekall Plugins

How to Parse a Memory Image with the Volatility® Framework

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	
0x86223440	smss.exe	268	4	2	29	-----	0	2012-02-16 12:04:46 UTC+0000	
0x86a81440	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	
0x86a55330	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	

Identify rogue processes

Six-Step Investigative Methodology Walkthrough

Base	Size	LoadCount	Path
0x00400000	0x1c2000	0xffff	C:\Program Files\TrueCrypt\TrueCrypt.exe
0x776b0000	0x13c000	0xffff	C:\Windows\SYSTEM32\ntdll.dll
0x76980000	0x4d000	0xffff	C:\Windows\system32\kernel32.dll
0x759f0000	0x4a000	0xffff	C:\Windows\system32\KERNELBASE.dll
0x75a70000	0x84000	0xffff	C:\Windows\WinSxS\x86_microsoft.windows.common-internet_659506414-ccf1d5-5.02.7600.16385_x-ww_eb126c75b8d5f3d5.CONCT132.dll
0x76610000	0xa0000	0xffff	C:\Windows\system32\ADVAPI32.dll
0x76560000	0xa0000	0xffff	C:\Windows\system32\msvcrt.dll
0x75b80000	0x19000	0xffff	C:\Windows\SYSTEM32\sechost.dll

Analyze process DLLs and handles

Review network artifacts

Foreign Address	State	Pid	Owner	Created
0x3d7338a0	UDPv4	0.0.0.0:58732	svchost.exe	2012-02-16 12:43:06
0x3d7338a0	UDPv6	:::58732	svchost.exe	2012-02-16 12:43:06
0x3dabaaf8	UDPv4	0.0.0.0:51319	svchost.exe	2012-02-16 12:43:26
0x3dabaaf8	UDPv6	:::51319	svchost.exe	2012-02-16 12:43:26
0x3db1f720	UDPv6	:::11900	svchost.exe	2012-02-16 12:43:05
0x3dc23b08	TCPv4	0.0.0.0:445	System	2012-02-16 12:43:05
0x3dc23b08	TCPv6	:::445	System	2012-02-16 12:43:05
0x3dece50	UDPv4	127.0.0.1:65096	svchost.exe	2012-02-16 12:43:05
0x3d22a70	UDPv4	172.16.246.144:138	System	2012-02-16 12:43:05
0x3d223570	UDPv4	0.0.0.0:5355	svchost.exe	2012-02-16 12:43:26
0x3dfc190	UDPv6	:::165095	svchost.exe	2012-02-16 12:43:05
0x3e21ae18	UDPv4	0.0.0.0:0	svchost.exe	2012-02-16 12:43:26
0x3e21ae18	UDPv6	:::0	svchost.exe	2012-02-16 12:43:26
0x3e22ef50	UDPv4	0.0.0.0:63429	svchost.exe	2012-02-16 12:05:02
0x3e22ef50	UDPv6	:::63429	svchost.exe	2012-02-16 12:05:02
0x3e2592a0	UDPv6	fe80::2186:5739:c69:4546:1900	svchost.exe	2012-02-16 12:43:05

Check for signs of a rootkit

```
sansforensics/cases $ vol.py -f be2.vmem ssdt [egrep -v 'ntoskrnl32k'
[x86] Gathering all referenced SSDTs from KTHREADS...
Finding appropriate address space for tables...
SSDT[0] at f3a890 with 284 entries
Entry 0x0041: 0xffffd2487 (NtDeleteValueKey) owned by 00004A2A
Entry 0x0047: 0xffffd216b (NtEnumerateValueKey) owned by 00004A2A
Entry 0x0049: 0xffffd2267 (NtEnumerateValueKey) owned by 00004A2A
Entry 0x0077: 0xffffd2d3c (NtOpenKey) owned by 00004A2A
Entry 0x007a: 0xffffd1e93 (NtOpenProcess) owned by 00004A2A
Entry 0x0080: 0xffffd1f0b (NtOpenThread) owned by 00004A2A
Entry 0x0089: 0xffffd2617 (NtProtectVirtualMemory) owned by 00004A2A
Entry 0x00ad: 0xffffd1d1a (NtQuerySystemInformation) owned by 00004A2A
Entry 0x00ba: 0xffffd256b (NtReadVirtualMemory) owned by 00004A2A
Entry 0x00d5: 0xffffd2070 (NtSetContextThread) owned by 00004A2A
Entry 0x00f7: 0xffffd2397 (NtSetThreadExecutionState) owned by 00004A2A
Entry 0x00fe: 0xffffd201d (NtSuspendThread) owned by 00004A2A
Entry 0x0102: 0xffffd1f1c (NtTerminateThread) owned by 00004A2A
Entry 0x0115: 0xffffd25c1 (NtWriteVirtualMemory) owned by 00004A2A
```

Dump suspicious processes and drivers

```
sansforensics/cases $ vol.py -f be2.vmem moddump -b 0xffffd1000 -D
Module Base Module Name Result
0xffffd1000 00004A2A OK: driver:f0d1000.sys
```

In-Depth Memory Analysis

Timestamp

Filename

Reason

Attributes

2012-02-16 12:06:00.332	mysecrdata.tc	CREATE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.332	mysecrdata.tc	CREATE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.348	mysecrdata.tc	EXTEND & CREATE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.348	mysecrdata.tc	EXTEND & CREATE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	BASIC INFO CHANGE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	CLOSE & BASIC INFO CHANGE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	EXTEND & CREATE & CLOSE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	BASIC INFO CHANGE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	CLOSE & BASIC INFO CHANGE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.426	mysecrdata.tc	EXTEND & CREATE & CLOSE	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.582	mysecrdata.tc	RENAME_OLD_NAME	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.582	mysecrdata.tc	RENAME_NEW_NAME	ARCHIVE & NOT CONTENT INDEXED
2012-02-16 12:06:00.582	mysecrdata.tc	RENAME_NEW_NAME & CLOSURE	ARCHIVE & NOT CONTENT INDEXED

sanforensics/cases \$ vol.py -f cridx.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.exe" (PIDs: 1640)

Time

Source

Destination

Protocol

Length

Info

0.000000	216.239.115.141	192.168.2.7	HTTP	1514	Continuation of 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
0.0.024518	216.239.115.141	192.168.2.7	HTTP	1514	Continuation of non-HTTP traffic
4.0.041405	192.168.2.7	216.239.115.141	TCP	54	1125 > 80 [ACK] Seq=1 Ack=1461 Win=64512 Len=0
0.0.050469	216.239.115.141	192.168.2.7	HTTP	1514	Continuation of non-HTTP traffic
6.0.060020	216.239.115.141	192.168.2.7	HTTP	1514	Continuation of non-HTTP traffic
7.0.068842	216.239.115.141	192.168.2.7	HTTP	1514	Continuation of non-HTTP traffic
8.0.084990	192.168.2.7	216.239.115.141	TCP	54	1125 > 80 [ACK] Seq=1 Ack=1461 Win=64512 Len=0

Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0

Ethernet II, Src: 00:11:50:56:1c:1c:5c (00:11:50:56:1c: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

Module

User

Domain

Password

wdlgst	Marc Lucas	Budweiser	P@ssw0rd
wdlgst	BUDWEISERS	WORKGROUP	

index url

title

visits typed

last visit time

101	http://www.kickstarter.com/projects/baz-the-capital-of-maine-is-augusta-also-paz-is-makin	1	0	2012-02-21 00:21:56
100	http://allthingsd.com/20120220/ask-a-qualcomm-samsung-nokia-qualcomm-samsung-gear-up-for-the-next-round-of-patent-fights-its	1	0	2012-02-21 01:31:31
99	http://www.wired.com/	1	0	2012-02-20 23:47:56
98	http://www.wired.com/	1	0	2012-02-20 23:47:56
97	http://venturebeat.com/	1	0	2012-02-20 23:47:56
96	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
95	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
94	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
93	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
92	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
91	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
90	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
89	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
88	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
87	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
86	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
85	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
84	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
83	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
82	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
81	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
80	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
79	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
78	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
77	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
76	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
75	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
74	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
73	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
72	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
71	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
70	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
69	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
68	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
67	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
66	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
65	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
64	http://www.businessinsider.com/	1	0	2012-02-20 23:47:56
63	http://			

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

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).

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!

_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 Rekal. 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!

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

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

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

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. Rekal 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.

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

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.