

Volatility command 2.1

(번역 문서)

해당 문서는 연구목적으로 진행된 번역 프로젝트입니다.

상업적으로 사용을 하거나, 악의적인 목적에 의한 사용을 할 시 발생하는
법적인 책임은 사용자 자신에게 있음을 경고합니다.

원본 : <http://code.google.com/p/volatility/wiki/CommandReference21#hivescan>

번역자

이승준 (에디)

전창배 (13lack4t)

조정원 (니키)

- 보안프로젝트 (www.boanproject.com) -

목 차

1. 시작하기전에	1
2. 이미지 식별	3
2.1. Imageinfo.....	3
2.2. kdbgscan.....	4
2.3. kpcrscan	6
3. 프로세스와 DLL	7
3.1. pllist	7
3.2. pstree.....	8
3.3. psscan.....	9
3.4. psdispscan	9
3.5. dlllist	9
3.6. dlldump	10
3.7. handles	12
3.8. getsids	13
3.9. cmdscan	13
3.10. consoles	15
3.11. envvars	17
3.12. verinfo	18
3.13. enumfunc.....	18
4. 프로세스 메모리	20
4.1. memmap.....	20
4.2. memmap.....	20
4.3. procmemdump	21
4.4. procexedump	21
4.5. vadinfo	21
4.6. vadwalk	22
4.7. vadtrees	23
4.8. vaddump.....	23
5. 커널 메모리와 오브젝트	25
5.1. modules	25
5.2. modules	26
5.3. moddump	26
5.4. ssdt	27

5.5. driverscan	28
5.6. filescan	29
5.7. mutantscan.....	30
5.8. symlinkscan	30
5.9. thrdsan	31
6. 네트워크	32
6.1. connections	32
6.2. conscan	32
6.3. sockets	33
6.4. sockscan.....	33
6.5. netscan	34
7. 레지스트리	36
7.1. hivescan	36
7.2. hivelist	36
7.3. printkey	37
7.4. hivedump.....	38
7.5. hashdump.....	39
7.6. lsadump	40
7.7. userassist	41
7.8. shimcache	42
8. 충돌, 절전, 전환	43
8.1. crashinfo	43
8.2. hidinfo	43
8.3. imagecopy	44
8.4. raw2dmp.....	44
9. 악성 코드, 루트킷	45
9.1. malfind	45
9.2. yarascan	46
9.3. svcscan	47
9.4. ldrmodules.....	48
9.5. impscan.....	49
9.6. apihooks.....	52
9.7. idt.....	55
9.8. gdt.....	56
9.9. threads	58
9.10. callbacks.....	59

9.11. devicetree	60
9.12. psxview	62
9.13. timers	63
9.14. driverirp	64
10. 그외	66
10.1. strings	66
10.2. volshell	69
10.3. bioskbd	72
10.4. pacher	73
10.5. pagecheck.....	74
11. 끝맺음	75

그림 목차

그림 1. volatility 설치 화면.....	1
그림 2. 윈도우 환경 volatility 실행 화면.....	2

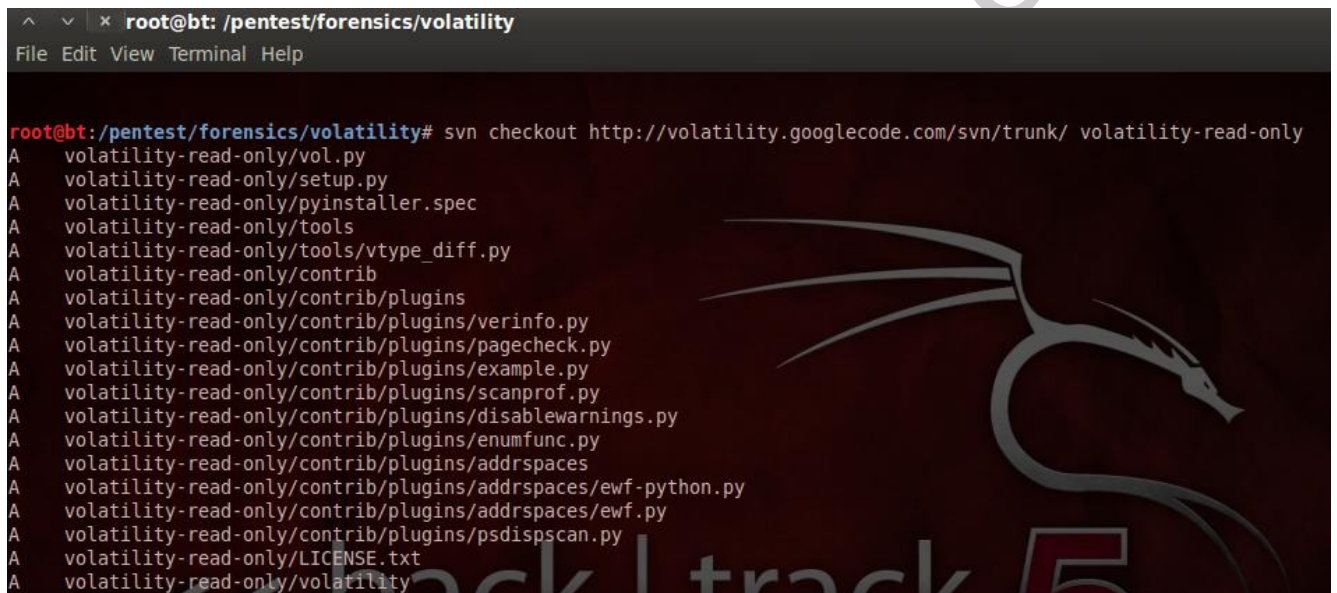
www.boanproject.com

1. 시작하기전에

본 문서는 포렌식 메모리 분석으로 많이 사용되고 있는 Volatility 의 Tutorial 을 번역하고 이에 대한 연구를 목적으로 만들어졌다.

번역자가 선택한 환경은 BackTrack V5 R2 에서 최신 버전을 다운로드 받아(현재 v2.1) 실습을 수행하였으며, vmware 이미지를 대상으로 진행하였다. volatility 의 신규 버전을 설치 하기 위해서는 svn 을 이용해서 설치를 간단하게 할 수 있다.

```
svn checkout http://volatility.googlecode.com/svn/trunk/ volatility-read-only
```



```

root@bt: /pentest/forensics/volatility
File Edit View Terminal Help

root@bt:/pentest/forensics/volatility# svn checkout http://volatility.googlecode.com/svn/trunk/ volatility-read-only
A      volatility-read-only/vol.py
A      volatility-read-only/setup.py
A      volatility-read-only/pyinstaller.spec
A      volatility-read-only/tools
A      volatility-read-only/tools/vtype_diff.py
A      volatility-read-only/contrib
A      volatility-read-only/contrib/plugins
A      volatility-read-only/contrib/plugins/verinfo.py
A      volatility-read-only/contrib/plugins/pagecheck.py
A      volatility-read-only/contrib/plugins/example.py
A      volatility-read-only/contrib/plugins/scanprof.py
A      volatility-read-only/contrib/plugins/disablewarnings.py
A      volatility-read-only/contrib/plugins/enumfunc.py
A      volatility-read-only/contrib/plugins/addrspaces
A      volatility-read-only/contrib/plugins/addrspaces/ewf-python.py
A      volatility-read-only/contrib/plugins/addrspaces/ewf.py
A      volatility-read-only/contrib/plugins/psdispcan.py
A      volatility-read-only/LICENSE.txt
A      volatility-read-only/volatility
  
```

그림 1. volatility 설치 화면

테스트 이미지는 개인적으로 사용하고 있던 테스트 환경의 이미지 파일(vmem)이며, 아래 항목의 이미지도 활용 하였다.

[Images from [The Malware Analyst's Cookbook](#)]

Description	url	OS
be2.vmem.zip	be2.vmem.zip	XP SP2
coreflood.vmem.zip	coreflood.vmem.zip	XP SP2
laqma.vmem.zip	laqma.vmem.zip	XP SP2
prolaco.vmem.zip	prolaco.vmem.zip	XP SP2
salinity.vmem.zip	salinity.vmem.zip	XP SP2
silentbanker.vmem.zip	silentbanker.vmem.zip	XP SP2

Volatility Command 2.1 번역 문서

tigger.vmem.zip	tigger.vmem.zip	XP SP2
zeus.vmem.zip	zeus.vmem.zip	XP SP2
spyeye.vmem.zip	spyeye.vmem.zip	XP SP2

[Other Images]

Description	url	OS
Stuxnet image	stuxnet.vmem.zip	XP SP3
NIST	http://www.cfreds.nist.gov/mem/memory-images.rar	XP SP2
Hogfly's malware memory samples	http://cid-5694a755c9c6a175.skydrive.live.com/browse.aspx/Public	?
Moyix's Fuzzy Hidden Process Sample	http://amnesia.gtisc.gatech.edu/~moyix/ds_fuzz_hidden_proc.img.bz2	XP SP3
Honeynet Banking Troubles Image	https://www.honeynet.org/challenges/2010_3_banking_troubles	XP SP2
NPS 2009-M57	https://domex.nps.edu/corp/nps/scenarios/2009-m57-patents/ram/	Various XP / Vista
Dougee's comparison samples	before and after infection	XP
Shylock Sample	Shylock vmem	XP
R2D2 Sample	0zapftis.rar (pw: infected)	XP SP2
Honeynet Compromised Server Challenge	http://www.honeynet.org/challenges/2011_7_compromised_server	Linux - Debian 2.6.26-26
Pikeworks Linux Samples	http://secondlookforensics.com/images.html	Misc Linux
Cridex Sample	cridex memdump.zip	XP SP2

윈도우 환경은 후에 "윈도우 환경 설치"가 나오겠지만, 이것을 설치하기 위해서는 많은 단계를 거치게 되며, 실제 설치 단계에서 많은 오류가 발생한다. 최적으로 사용할 수 있는 것은 Python 2.7 만 설치된 상태에서 volatility-2.1.standalone.exe 를 사용할 수 있다는 것이다.

```

관리자: C:\Windows\system32\cmd.exe

D:\vmware\Windows XP Professional>volatility-2.1.exe -f "Windows XP Professional-440e3175.vmem" imageinfo
Volatile Systems Volatility Framework 2.1
Determining profile based on KDBG search...

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (D:\vmware\Windows XP Professional\Windows XP Professional-440e3175.vmem)
PAE type : PAE
DTB : 0xc1f000L
KDBG : 0x80547ae0L
Number of Processors : 1
Image Type (Service Pack) : 3
KPCR for CPU 0 : 0xffdf000L
KUSER_SHARED_DATA : 0xffdf000L
Image date and time : 2012-06-07 02:27:59 UTC+0000
Image local date and time : 2012-06-07 11:27:59 +0900

D:\vmware\Windows XP Professional>
  
```

그림 2. 윈도우 환경 volatility 실행 화면

2. 이미지 식별

2.1. Imageinfo

당신이 분석하는 메모리 샘플의 고난이도 요약을 위해서는 imageinfo 명령어를 사용해야 한다. 대부분의 경우 이 명령어는 운영시스템이나 서비스팩, 그리고 하드웨어 구조(32 또는 64 비트)를 식별하기 위해 쓰였지만 DTB 주소나 수집된 샘플의 시간과 같은 유용한 정보도 담고 있다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw imageinfo
Volatile Systems Volatility Framework 2.1_alpha
Determining profile based on KDBG search...

Suggested Profile(s) : Win7SP0x64, Win7SP1x64, Win2008R2SP0x64, Win2008R2SP1x64
AS Layer1 : AMD64PagedMemory (Kernel AS)
AS Layer2 : FileAddressSpace (/Users/Michael/Desktop/win7_trial_64bit.raw)
PAE type : PAE
DTB : 0x187000L
KDBG : 0xf80002803070
Number of Processors : 1
Image Type (Service Pack) : 0
KPCR for CPU 0 : 0xfffff80002804d00L
KUSER_SHARED_DATA : 0xfffff78000000000L
Image date and time : 2012-02-22 11:29:02 UTC+0000
Image local date and time : 2012-02-22 03:29:02 -0800
```

Imageinfo 출력값은 다른 플러그인을 사용할 때 --profile=PROFILE----을 파라미터로 해야만 하는 제안된 프로파일을 알려 준다. 만약 프로파일들이 근접하게 관련되어 있다면 아마 한 가지 프로파일 제안보다는 더 많을 것이다. 또한 그것은 프로세스와 모듈 리스트 헤드들을 각각 찾기 위해 CommandReference21#pslist 와 CommandReference21#modlist 와 같은 플러그인들에 의해 사용되는 KDBG(KDDEBUGGER_DATA64 의 약자) 구조의 주소를 출력한다. 특히 큰 메모리 샘플들같은 몇몇의 경우에는, 여러 개의 KDBG 구조들이 있을 것이다. 유사하게, 만약 여러 개의 프로세서들이 있다면, 당신은 각각에 해당하는 KPCR 주소와 CPU 번호를 볼 수 있을 것이다.

플러그인들은 필요로 할 때 KPCR 과 KDBG 값들을 자동으로 스캔 한다. 그러나, 당신은 --kpcr=ADDRESS 또는 --kdbg=ADDRESS 를 제공함으로써 어떠한 플러그인이든 직접적으로 값들을 명시할 수 있다. 프로파일이나 KDBG(또는 KPCR 값으로 놓고)를 다른 Volatility 명령어들 제공함으로써, 당신은 가장 가능한 정확하고 빠른 결과를 얻을 수 있을 것이다.

2.2. kdbgscan

단순히 프로파일만 제안하는 CommandReference21#imageinfo 와는 반대로, kdbgscan 은 정확한 프로파일과 정확한 KDBG 주소(만약 여러 개가 있다면)를 식별하도록 디자인 되어 있다. KDBGHeader 특징들을 위한 플러그인 스캔들은 Volatility 프로파일들과 연결되어 있고 오탐을 줄이기 위한 분별체크를 적용한다. 많은 출력과 분별체크의 숫자들은 Volatility 가 DTB 를 찾을 수 있는지에 따라 실행 될 수 있다. 만약 이미 정확한 프로파일(또는 CommandReference21#imageinfo 에서 얻은 프로파일 제안을 가지고 있다면)을 알고 있다면, 반드시 그것을 사용하도록 해라.

이 플러그인이 유용할때의 예제 시나리오가 있다. 당신이 Windows 2003 SP2 x64 라고 믿고있는 메모리 샘플이 있다. pslist 플러그인은 KDBG 에 의해 지목된 프로세스 리스트 헤드를 찾는데 의존한다. 하지만 플러그인이 메모리 샘플에서 얻은 첫번째 KDBG 는 항상 썬 나온 것이 될 수는 없다. 만약 유효하지 않은 PsActiveProcessHead 포인터가 있는 KDBG 가 유효한 KDBG 보다 먼저 샘플(즉, 낮은 물리적 오프셋에서)에서 발견된다면 문제가 될 수 있다.

아래에서 어떻게 kdbgscan 이 두개의 KDBG 구조들을 뽑았는지 알아보자. 유효하지 않은 것(0 프로세스, 0 모듈)은 첫번째로 0xf80001172cb0 에서 발견되었고, 유효한 것(37 프로세스, 116 모듈)은 그 다음에 0xf80001175cf0 에서 발견되었다. 이 샘플에서 CommandReference21#pslist 를 고치기 위해서는, 단순히 pslist 플러그인에 --kdbg=0xf80001175cf0 를 제공하기만 하면 된다.

```
$ python vol.py -f Win2K3SP2x64-6f1bedec.vmem --profile=Win2003SP2x64 kdbgscan
Volatile Systems Volatility Framework 2.1_alpha
*****
Instantiating KDBG using: Kernel AS Win2003SP2x64 (5.2.3791 64bit)
Offset (V) : 0xf80001172cb0
Offset (P) : 0x1172cb0
KDBG owner tag check : True
Profile suggestion (KDBGHeader): Win2003SP2x64
Version64 : 0xf80001172c70 (Major: 15, Minor: 3790)
Service Pack (CmNtCSDVersion) : 0
Build string (NtBuildLab) : T?
PsActiveProcessHead : 0xfffff800011947f0 (0 processes)
PsLoadedModuleList : 0xfffff80001197ac0 (0 modules)
KernelBase : 0xfffff80001000000 (Matches MZ: True)
Major (OptionalHeader) : 5
Minor (OptionalHeader) : 2
*****
Instantiating KDBG using: Kernel AS Win2003SP2x64 (5.2.3791 64bit)
Offset (V) : 0xf80001175cf0
Offset (P) : 0x1175cf0
KDBG owner tag check : True
Profile suggestion (KDBGHeader): Win2003SP2x64
Version64 : 0xf80001175cb0 (Major: 15, Minor: 3790)
Service Pack (CmNtCSDVersion) : 2
Build string (NtBuildLab) : 3790.srv03_sp2_rtm.070216-1710
PsActiveProcessHead : 0xfffff800011977f0 (37 processes)
PsLoadedModuleList : 0xfffff8000119aae0 (116 modules)
KernelBase : 0xfffff80001000000 (Matches MZ: True)
```

Volatility Command 2.1 번역 문서

Major (OptionalHeader)	: 5
Minor (OptionalHeader)	: 2
KPCR	: 0xfffff80001177000 (CPU 0)

어떻게 KDBG 구조들을 식별하는지에 대한 더 자세한 정보는 FindingKernel Global Variable in Windows 와 Identifying MemoryImages 를 읽으면 된다.

2.3. kpcrscan

이 명령어는 Finding Object Roots in Vista 에 묘사된 것처럼 자체참조 멤버들을 체크함으로써 잠재적 KPCR 구조들을 스캔 하기 위한 것이다. 멀티코어 시스템에서는 각각의 프로세서마다 KPCR 을 가지고 있다. 따라서, IDT 와 GDT 주소, current, idle, 그리고 다음 스레드들, CPU 숫자, 벤더&속도, 그리고 CR3 값들을 포함하는 각각의 프로세서에 대한 자세한 세부 정보들을 볼 수 있을 것이다.

```
$ python vol.py -f dang_win7_x64.raw --profile=Win7SP1x64 kpcrscan
Volatile Systems Volatility Framework 2.1_alpha
*****
Offset (V)           : 0xf800029ead00
Offset (P)           : 0x29ead00
KdVersionBlock       : 0x0
IDT                  : 0xfffff80000b95080
GDT                  : 0xfffff80000b95000
CurrentThread        : 0xfffffa800cf694d0 TID 2148 (kd.exe:2964)
IdleThread           : 0xfffff800029f8c40 TID 0 (Idle:0)
Details              : CPU 0 (GenuineIntel @ 2128 MHz)
CR3/DTB              : 0x1dcec000
*****
Offset (V)           : 0xf880009e7000
Offset (P)           : 0x4d9e000
KdVersionBlock       : 0x0
IDT                  : 0xfffff880009f2540
GDT                  : 0xfffff880009f24c0
CurrentThread        : 0xfffffa800cf694d0 TID 2148 (kd.exe:2964)
IdleThread           : 0xfffff880009f1f40 TID 0 (Idle:0)
Details              : CPU 1 (GenuineIntel @ 2220 MHz)
CR3/DTB              : 0x1dcec000
```

만약 KdVersionBlock 이 null 이 아니라면, KPCR 을 통해 컴퓨터의 KDBG 주소를 찾는 것이 가능할 수도 있다.

3. 프로세스와 DLL

3.1. pllist

시스템의 프로세스들을 열거하기 위해 pslist 명령어를 사용한다. 이는 PsActiveProcessHead 를 가리키는 이중연결리스트를 지나가며 오프셋, 프로세스 이름, 프로세스 ID, 부모 프로세스 ID, 스레드의 수, 핸들의 수, 프로세스 시작 시간과 종료 시간을 보여 준다. 2.1 에서는 세션 ID 와 Wow64 프로세스(64bit 커널에서 32bit 주소를 사용한다.) 여부를 보여 준다.

이 플러그인은 숨겨 지거나 연결이 끊어진 프로세스들을 보여주지 않는다.(하지만 psscan 은 가능하다.)

만약 스레드 수가 0 이고, 핸들 수가 0 이고, 종료 시간이 비어 있지 않은 프로세스들을 본다면, 그 프로세스들은 실제로 여전히 활성화된 상태는 아닐 수도 있다. 더 많은 정보를 위해, PsActiveProcessHead 의 Missing Active 를 보라. 아래를 보면, regsvr32.exe 는 종료되었지만, 여전히 "활성" 목록에 존재하는 것을 볼 수 있다.

또한 System 과 smss.exe 는 세션 ID 를 가지지 않는다. 왜냐하면 System 은 세션들이 자리 잡기 이전에 시작되고 smss.exe 는 자기 스스로가 세션 관리자이기 때문이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 pslist
```

Volatile Systems Volatility Framework 2.1_alpha

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64
Start	Exit						
0xfffffa80004b09e0	System	4	0	78	489	-----	0 2012-02-22
19:58:20							
0xfffffa8000ce97f0	smss.exe	208	4	2	29	-----	0 2012-02-22
19:58:20							
0xfffffa8000c006c0	csrss.exe	296	288	9	385	0	0 2012-02-22
19:58:24							
0xfffffa8000c92300	wininit.exe	332	288	3	74	0	0 2012-02-22
19:58:30							
0xfffffa8000c06b30	csrss.exe	344	324	7	252	1	0 2012-02-22
19:58:30							
0xfffffa8000c80b30	winlogon.exe	372	324	5	136	1	0 2012-02-22
19:58:31							
0xfffffa8000c5eb30	services.exe	428	332	6	193	0	0 2012-02-22
19:58:32							
0xfffffa80011c5700	lsass.exe	444	332	6	557	0	0 2012-02-22
19:58:32							
0xfffffa8000ea31b0	lsim.exe	452	332	10	133	0	0 2012-02-22
19:58:32							
0xfffffa8001296b30	svchost.exe	568	428	10	352	0	0 2012-02-22
19:58:34							
0xfffffa80012c3620	svchost.exe	628	428	6	247	0	0 2012-02-22
19:58:34							
0xfffffa8001325950	sppsvc.exe	816	428	5	154	0	0 2012-02-22
19:58:41							
0xfffffa80007b7960	svchost.exe	856	428	16	404	0	0 2012-02-22
19:58:43							
0xfffffa80007bb750	svchost.exe	880	428	34	1118	0	0 2012-02-22
19:58:43							
0xfffffa80007d09e0	svchost.exe	916	428	19	443	0	0 2012-02-22

Volatility Command 2.1 번역 문서

19:58:43							
0xfffffa8000c64840	svchost.exe	348	428	14	338	0	0 2012-02-22
20:02:07							
0xfffffa8000c09630	svchost.exe	504	428	16	496	0	0 2012-02-22
20:02:07							
0xfffffa8000e86690	spoolsv.exe	1076	428	12	271	0	0 2012-02-22
20:02:10							
0xfffffa8000518b30	svchost.exe	1104	428	18	307	0	0 2012-02-22
20:02:10							
0xfffffa800094d960	wlms.exe	1264	428	4	43	0	0 2012-02-22
20:02:11							
0xfffffa8000995b30	svchost.exe	1736	428	12	200	0	0 2012-02-22
20:02:25							
0xfffffa8000aa0b30	SearchIndexer.	1800	428	12	757	0	0 2012-02-22
20:02:26							
0xfffffa8000aea630	taskhost.exe	1144	428	7	189	1	0 2012-02-22
20:02:41							
0xfffffa8000eafb30	dwm.exe	1476	856	3	71	1	0 2012-02-22
20:02:41							
0xfffffa80008f3420	explorer.exe	1652	840	21	760	1	0 2012-02-22
20:02:42							
0xfffffa8000c9a630	regsvr32.exe	1180	1652	0	-----	1	0 2012-02-22
20:03:05 2012-02-22 20:03:08							
0xfffffa8000a03b30	rundll32.exe	2016	568	3	67	1	0 2012-02-22
20:03:16							
0xfffffa8000a4f630	svchost.exe	1432	428	12	350	0	0 2012-02-22
20:04:14							
0xfffffa8000999780	iexplore.exe	1892	1652	19	688	1	1 2012-02-22
11:26:12							
0xfffffa80010c9060	iexplore.exe	2820	1892	23	733	1	1 2012-02-22
11:26:15							
0xfffffa8001016060	DumpIt.exe	2860	1652	2	42	1	1 2012-02-22
11:28:59							
0xfffffa8000acab30	conhost.exe	2236	344	2	51	1	0 2012-02-22 11:28:59

기본적으로, pslist 는 EPROCESS 를 위한 가상 오프셋을 보여주지만, -P 스위치를 이용하면 물리 오프셋으로 보여준다.

3.2. pstree

트리 형태로 프로세스를 목록화해서 보기위해 pstree 명령어를 사용한다. 이는 pslist 와 같은 방식을 사용하여 프로세스들을 열거하기에, pslist 와 마찬가지로 숨겨지거나 연결이 끊어진 프로세스들을 보여주지 않는다. 자식 프로세스들은 공백과 마침표를 이용하여 표기한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 pstree
Volatile Systems Volatility Framework 2.1_alpha
Name                               Pid  PPid  Thds  Hnds Time
-----
0xfffffa80004b09e0:System            4    0    78   489 2012-02-22 19:58:20
. 0xfffffa8000ce97f0:smss.exe        208   4     2    29 2012-02-22 19:58:20
. 0xfffffa8000c006c0:csrss.exe       296  288     9   385 2012-02-22 19:58:24
. 0xfffffa8000c92300:wininit.exe     332  288     3    74 2012-02-22 19:58:30
. 0xfffffa8000c5eb30:services.exe    428  332     6   193 2012-02-22 19:58:32
.. 0xfffffa8000aa0b30:SearchIndexer. 1800  428    12   757 2012-02-22 20:02:26
.. 0xfffffa80007d09e0:svchost.exe     916  428    19   443 2012-02-22 19:58:43
.. 0xfffffa8000a4f630:svchost.exe    1432  428    12   350 2012-02-22 20:04:14
.. 0xfffffa800094d960:wlms.exe       1264  428     4    43 2012-02-22 20:02:11
.. 0xfffffa8001325950:sppsvc.exe     816  428     5   154 2012-02-22 19:58:41
.. 0xfffffa8000e86690:spoolsv.exe    1076  428    12   271 2012-02-22 20:02:10
.. 0xfffffa8001296b30:svchost.exe     568  428    10   352 2012-02-22 19:58:34
... 0xfffffa8000a03b30:rundll32.exe  2016  568     3    67 2012-02-22 20:03:16
...
```

3.3. psscan

풀 태그 스캐닝을 사용하여 프로세스들을 열거하기 위해 psscan 명령어를 사용한다. 이는 이전에 종료된(비활성) 프로세스들과 루트킷에 의해 숨겨지거나 연결이 끊어진 프로세스들을 찾을 수 있다.

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp psscan
```

Volatile Systems Volatility Framework 2.0

Offset	Name	PID	PPID	PDB	Time created	Time exited
0x3e025ba8	svchost.exe	1116	508	0x3ecf1220	2010-06-16 15:25:25	
0x3e04f070	svchost.exe	1152	508	0x3ecf1340	2010-06-16 15:27:40	
0x3e144c08	dwm.exe	1540	832	0x3ecf12e0	2010-06-16 15:26:58	
0x3e145c18	TPAutoConnSvc.	1900	508	0x3ecf1360	2010-06-16 15:25:41	
0x3e3393f8	lsass.exe	516	392	0x3ecf10e0	2010-06-16 15:25:18	
0x3e35b8f8	svchost.exe	628	508	0x3ecf1120	2010-06-16 15:25:19	
0x3e383770	svchost.exe	832	508	0x3ecf11a0	2010-06-16 15:25:20	
0x3e3949d0	svchost.exe	740	508	0x3ecf1160	2010-06-16 15:25:20	
0x3e3a5100	svchost.exe	872	508	0x3ecf11c0	2010-06-16 15:25:20	
0x3e3f64e8	svchost.exe	992	508	0x3ecf1200	2010-06-16 15:25:24	
0x3e45a530	wininit.exe	392	316	0x3ecf10a0	2010-06-16 15:25:15	
0x3e45d928	svchost.exe	1304	508	0x3ecf1260	2010-06-16 15:25:28	
0x3e45f530	csrss.exe	400	384	0x3ecf1040	2010-06-16 15:25:15	
0x3e4d89c8	vmtoolsd.exe	1436	508	0x3ecf1280	2010-06-16 15:25:30	
0x3e4db030	spoolsv.exe	1268	508	0x3ecf1240	2010-06-16 15:25:28	
0x3e50b318	services.exe	508	392	0x3ecf1080	2010-06-16 15:25:18	
0x3e7f3d40	csrss.exe	352	316	0x3ecf1060	2010-06-16 15:25:12	
0x3e7f5bc0	winlogon.exe	464	384	0x3ecf10c0	2010-06-16 15:25:18	
0x3eac6030	SearchProtocol	2448	1168	0x3ecf15c0	2010-06-16 23:30:52	2010-06-16 23:33:14
0x3eb10030	SearchFilterHo	1812	1168	0x3ecf1480	2010-06-16 23:31:02	2010-06-16 23:33:14

[snip]

만약 프로세스가 이미 종료되었다면, 종료시간 항목에서 종료시간을 볼수가 있다. 만약 숨겨진 프로세스에 대해서 조사하고 싶다면(또한 그 프로세스의 DLL 들을 보려면), EPROCESS 오브젝트의 물리 오프셋을 필요할 것이고, 이는 왼쪽 멀리 떨어진 행에서 보일 것이다.

3.4. psdispscan

이 플러그인은 psscan 하고 비슷하지만 pool tags 대신에 DISPATCHER_HEADER 를 조사하여 프로세스들을 열거한다. 이는 공격자가 pool tags 를 수정하여 숨기려 한 이벤트의 EPROCESS 오브젝트들을 추출하기 위한 대안을 제공한다. 이 플러그인은 잘 관리되지 않았고, 오직 XP x86 만 지원한다. 이를 쓰기 위해서는 명령어라인에 --plugins=contrib/plugins 을 꼭 입력해야한다.

3.5. dlllist

한 프로세스에서 로드된 DLL 들을 보기 위해 dlllist 명령어를 사용한다. 이는 PEB 의 InLoadOrderModuleList 에서 가리키는 LDR_DATA_TABLE_ENTRY 구조체들의 이중연결리스트를 지난다. DLL 들은 한 프로세스에서 LoadLibrary(또는 LdrLoadDll 과 같은 몇몇 파생함수)를

Volatility Command 2.1 번역 문서

호출할 때 이 리스트에 자동으로 추가되며 FreeLibrary 가 호출되거나 레퍼런스 카운트가 0 이 될 때까지 제거되지 않는다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 dlllist
*****
wininit.exe pid: 332
Command line : wininit.exe

Base                               Size Path
-----
0x00000000ff530000 0x23000 C:\Windows\system32\wininit.exe
0x0000000076d40000 0x1ab000 C:\Windows\SYSTEM32\ntdll.dll
0x0000000076b20000 0x11f000 C:\Windows\system32\kernel32.dll
0x000007fefcd50000 0x6b000 C:\Windows\system32\KERNELBASE.dll
0x0000000076c40000 0xfa000 C:\Windows\system32\USER32.dll
0x000007fefd7c0000 0x67000 C:\Windows\system32\GDI32.dll
0x000007fefe190000 0xe000 C:\Windows\system32\LPK.dll
0x000007fefef80000 0xca000 C:\Windows\system32\USP10.dll
0x000007fefd860000 0x9f000 C:\Windows\system32\msvcrt.dll
[snip]
```

모든 프로세스들 대신에 특정 프로세스의 DLL 들을 보기 위해 아래와 같이 -p 또는 -pid 필터를 사용한다. 또한 아래 결과와 같이 Wow64 프로세스를 분석중인 것을 알려준다. Wow64 프로세스들은 PEB 리스트들 안에 한정된 DLL 들의 목록을 가지고 있지만, 그 목록들이 프로세스의 주소 공간에 로드된 DLL 의 전부는 아니다. 따라서 이런 프로세스들을 위해 Volatility 는 ldrmodules 명령어를 제공한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 dlllist -p 1892
Volatile Systems Volatility Framework 2.1_alpha
*****
iexplore.exe pid: 1892
Command line : "C:\Program Files (x86)\Internet Explorer\iexplore.exe"
Note: use ldrmodules for listing DLLs in Wow64 processes

Base                               Size Path
-----
0x0000000000800000 0xa6000 C:\Program Files (x86)\Internet Explorer\iexplore.exe
0x0000000076d40000 0x1ab000 C:\Windows\SYSTEM32\ntdll.dll
0x00000000748d0000 0x3f000 C:\Windows\SYSTEM32\wow64.dll
0x0000000074870000 0x5c000 C:\Windows\SYSTEM32\wow64win.dll
0x0000000074940000 0x8000 C:\Windows\SYSTEM32\wow64cpu.dll
```

루트킷에 의해 숨겨지거나 연결이 끊어진 프로세스들의 DLL 들을 보기위해 psscan 명령어를 이용하여 EPROCESS 오브젝트의 물리 주소를 획득하고 --offset=OFFSET 으로 사용한다. 이 플러그인은 돌아오고 EPROCESS 의 가상주소를 결정하고 PEB 에 접근하기 위해 주소 공간을 획득한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 dlllist --offset=0x04a291a8
```

3.6. dlldump

프로세스의 메모리 공간에서 DLL 을 추출하거나 분석을 위해 디스크로 덤프를 쓰기위해 dlldump 명령어를 사용한다. 문법은 dlllist 에서 본것과 거의 동일하다. 아래와 같은 일을 할수 있다:

Volatility Command 2.1 번역 문서

- 모든 프로세스로부터 모든 DLL 을 덤프 뜬다.
- 특정 프로세스로부터 모든 DLL 을 덤프 뜬다.(--pid=PID 사용)
- 숨겨지거나 연결이 끊어진 프로세스로부터 모든 DLL 을 덤프 뜬다.(--offset=OFFSET 사용)
- 프로세스 메모리의 아무위치로부터 PE 를 덤프 뜬다.(--base=BASEADDR 사용) 이 옵션은 숨겨진 DLL 들을 추출하는데 유용하다.
- 대소문자구분(--ignore-case 사용)과 정규표현식(--regex=REGEX 사용)에 맞는 하나 또는 그 이상의 DLL 들을 덤프뜬다.

특정 출력 디렉토리를 설정하려면, --dump-dir=DIR 또는 -d DIR 을 사용한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 dlldump -D dlls/
...
Dumping sechost.dll, Process: lsass.exe, Base: 7fef830000 output: module.444.173c5700.7fef830000.dll
Dumping cryptbase.dll, Process: lsass.exe, Base: 7fefcb80000 output: module.444.173c5700.7fefcb80000.dll
Cannot dump lsass.exe@pstersvc.dll at 7fef71e0000
Dumping USP10.dll, Process: lsass.exe, Base: 7fefef80000 output: module.444.173c5700.7fefef80000.dll
Dumping LPK.dll, Process: lsass.exe, Base: 7fefe190000 output: module.444.173c5700.7fefe190000.dll
Cannot dump lsass.exe@WINSTA.dll at 7fefcc40000
Dumping GDI32.dll, Process: lsass.exe, Base: 7fef7c0000 output: module.444.173c5700.7fef7c0000.dll
Dumping DNSAPI.dll, Process: lsass.exe, Base: 7fefc270000 output: module.444.173c5700.7fefc270000.dll
Dumping Secur32.dll, Process: lsass.exe, Base: 7fefc5d0000 output: module.444.173c5700.7fefc5d0000.dll
Dumping SAMSRV.dll, Process: lsass.exe, Base: 7fefc7e0000 output: module.444.173c5700.7fefc7e0000.dll
Dumping KERNELBASE.dll, Process: lsass.exe, Base: 7fefcd50000 output:
module.444.173c5700.7fefcd50000.dll
...
```

만약 몇몇 DLL 들만 추출되고 나머지는 추출이 안된다면, 이는 아마도 해당 DLL 의 몇몇 메모리 페이지들이 페이징 때문에 메모리에 존재하지 않는다는 것을 의미한다. 특히, PE 헤더나 PE 섹션을 가진 첫번째 페이지가 매핑이 안될때 문제가 된다. 이런 경우에는 vaddump 명령어를 이용하여 메모리 파편을 추출할 수 있지만, 따로 손수 PE 헤더를 다시 빌드하거나 섹션들을 [Volatility 와 함께 CoreFlood Binary 들을 복구하는 것](#) 에서 묘사한 것과 같이 수정해야 한다.

DLL 목록에 존재하지 않는 PE 파일을 덤프 뜨기 위해 프로세스 메모리의 PE 의 base 주소를 특정합니다.

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp dlldump --pid=492 -D out --base=0x00680000
```

또한 EPROCESS 오프셋을 특정지으므로해서 숨겨진 프로세스에서 원하는 DLL 을 찾을수도 있다.

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp dlldump -o 0x3e3f64e8 -D out --base=0x00680000
```


3.7. handles

한 프로세스에서 열고있는 핸들들을 보기위해, handles 명령어를 사용한다. 이는 일반 파일, 레지스트리 키들, 뮤텍스들, 이름있는 파이프들, 이벤트들, 윈도우 스테이션들, 데스크톱들, 쓰레드들, 그리고 안전하게 실행가능한 모든 다른 형태의 오브젝트들에 해당된다. 이 명령어는 Volatility 1.3 framework 의 명령어 "files" 와 "regobjkeys" 를 대체한다. 현재 2.1 에서는 결과에 핸들 값과 각 오브젝트의 granted access 를 포함한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 handles
```

Offset(V)	Pid	Handle	Access Type	Details
0xfffffa80004b09e0	4	0x4	0x1ffff Process	System(4)
0xfffffa80000821a0	4	0x10	0x2001f	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS		
0xfffffa800007e040	4	0x14	0xf003f	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT_PREFETCHPARAMETERS		
0xfffffa8000081fa0	4	0x18	0x2001f Key	MACHINE\SYSTEM\SETUP
0xfffffa8000546990	4	0x1c	0x1f0001 ALPC Port	PowerMonitorPort
0xfffffa800054d070	4	0x20	0x1f0001 ALPC Port	PowerPort
0xfffffa80000676a0	4	0x24	0x20019	
Key		MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER		
0xfffffa8000625460	4	0x28	0x1ffff Thread	TID 160 PID 4
0xfffffa800007f400	4	0x2c	0xf003f	
Key		MACHINE\SYSTEM\CONTROLSET001		
0xfffffa800007f200	4	0x30	0xf003f	
Key		MACHINE\SYSTEM\CONTROLSET001\ENUM		
0xfffffa8000080d10	4	0x34	0xf003f	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\CLASS		
0xfffffa800007f500	4	0x38	0xf003f	
Key		MACHINE\SYSTEM\CONTROLSET001\SERVICES		
0xfffffa80001cd990	4	0x3c	0xe Token	
0xfffffa800007bfa0	4	0x40	0x20019	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\WMI\SECURITY		
0xfffffa8000cd52b0	4	0x44	0x120116 File	\Device\Mup
0xfffffa8000ce97f0	4	0x48	0x2a Process	smss.exe(208)
0xfffffa8000df16f0	4	0x4c	0x120089	
File		\Device\HarddiskVolume2\Windows\System32\en-US\win32k.sys.mui		
0xfffffa8000de37f0	4	0x50	0x12019f File	\Device\clsTxLog
0xfffffa8000952fa0	4	0x54	0x2001f	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\VIDEO\{6A8FC9DC-A76B-47FC-A703-17800182E1CE}\0000\VOLATILESETTINGS		
0xfffffa800078da20	4	0x58	0x12019f File	\Device\Tcp
0xfffffa8002e17610	4	0x5c	0x9	
Key		MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\IMAGE FILE EXECUTION OPTIONS		
0xfffffa80008f7b00	4	0x60	0x10	
Key		MACHINE\SYSTEM\CONTROLSET001\CONTROL\LSA		
0xfffffa8000da2870	4	0x64	0x100001 File	\Device\KsecDD
0xfffffa8000da3040	4	0x68	0x0 Thread	TID 228 PID 4
...				

특정 프로세스의 핸들들을 보기위해 --pid=PID 나 EPROCESS 구조체의 물리 오프셋(--physical-offset=OFFSET)을 특정한다. 또한 -t 나 --object-type=OBJECTTYPE 을 이용하여 오브젝트 형태를 필터링 할 수 있다. 예시로 pid 296 의 프로세스 오브젝트의 핸들들을 보여준다. 그 결과는 아래와 같다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 handles -p 296 -t Process
```

Offset(V)	Pid	Handle	Access Type	Details
-----------	-----	--------	-------------	---------

Volatility Command 2.1 번역 문서

0xfffffa8000c92300	296	0x54	0x1fffff Process	wininit.exe(332)
0xfffffa8000c5eb30	296	0xc4	0x1fffff Process	services.exe(428)
0xfffffa80011c5700	296	0xd4	0x1fffff Process	lsass.exe(444)
0xfffffa8000ea31b0	296	0xe4	0x1fffff Process	lsmd.exe(452)
0xfffffa8000c64840	296	0x140	0x1fffff Process	svchost.exe(348)
0xfffffa8001296b30	296	0x150	0x1fffff Process	svchost.exe(568)
0xfffffa80012c3620	296	0x18c	0x1fffff Process	svchost.exe(628)
0xfffffa8001325950	296	0x1dc	0x1fffff Process	sppsvc.exe(816)
...				

몇몇의 경우, Details 열이 공란이다(예를 들면, 오브젝트들이 이름을 가지지 않을 때). 기본적으로, 이름이 있는 오브젝트들과 이름이 없는 오브젝트 둘다 보여 준다. 하지만, 의미가 적은 결과를 숨기거나 이름있는 오브젝트들만 보고 싶다면 --silent 를 사용하면 된다.

3.8. getsids

한 프로세스와 관련된 SID(보안식별자)들을 보기 위해 getsids 라는 명령어를 사용한다. 다른 것들 가운데, 이는 악의적으로 권한이 상승된 프로세스를 식별하는데 도움을 준다.

더 많은 정보를 위해 BDG의 유저들을 위한 연결된 프로세스들을 참고하라.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 getsids
Volatile Systems Volatility Framework 2.1_alpha
System (4): S-1-5-18 (Local System)
System (4): S-1-5-32-544 (Administrators)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
System (4): S-1-16-16384 (System Mandatory Level)
smss.exe (208): S-1-5-18 (Local System)
smss.exe (208): S-1-5-32-544 (Administrators)
smss.exe (208): S-1-1-0 (Everyone)
smss.exe (208): S-1-5-11 (Authenticated Users)
smss.exe (208): S-1-16-16384 (System Mandatory Level)
[snip]
```

3.9. cmdscan

cmdscan 플러그인은 공격자가 콘솔 셸(cmd.exe)을 통해 입력한 명령어들을 찾기 위해 XP/2003/Vista/2008 에서 csrss.exe 의 메모리와 Windows 7 에서 conhost.exe 를 조사한다. 이는 공격자들이 RDP 세션을 통해 cmd.exe 를 열었던지 네트워크 백도어로부터 명령셸로 입력/출력을 변경하든 상관없이 사고 시스템에서 공격자의 행동들중 눈에 보이는 정보를 얻을 수 있는 가장 강력한 명령어들 중에 하나이다.

이 플러그인은 알려진 고정값(MaxHistory)을 찾고, 정상적인지 확인하는 것으로 COMMAND_HISTORY 라고 알려진 구조체들을 찾는다. MaxHistory 값을 cmd.exe 창의 좌상단에서 우클릭 후 속성에서 변경할 수 있는 건 중요한 점이다. 또한 HKCU\Console\HistoryBufferSize 레지스트리 키를 변경한 주어진 사용자에게 의해 열린 모든 콘솔들도 이값이 변경된다. 윈도우 시스템의 기본값은 50 이며, 이는 거의 대부분은 50 개의

Volatility Command 2.1 번역 문서

명령어들을 저장하고 있다는 것을 의미한다. 이 값을 수정할 필요가 있다면, --max_history=NUMBER 를 사용하면 된다.

이 플러그인에서 사용하는 구조체들은 공식적인 것은 아니다.(즉, Microsoft 는 이들에 대한 PDB 들을 제공하지 않는다.) 그러므로 WinDBG 나 다른 포렌식 프레임워크에는 사용이 불가능하다. 이것들은 conhost.exe 와 winsrv.dll 바이너리들을 Michael Ligh 가 리버싱 하였다.

뿐만 아니라 이 명령어가 쉘에 입력되면, 이 플러그인은 아래와 같은 정보를 보여준다:

- 콘솔 호스트 프로세스(csrss.exe 또는 conhost.exe)의 이름
- 콘솔로 사용되고 있는 어플리케이션의 이름(어떤 프로세스든 cmd.exe 를 사용함)
- 명령어 히스토리 버퍼들의 위치, 현재 버퍼 카운트, 마지막으로 추가된 명령어, 그리고 마지막으로 보여진 명령어
- 어플리케이션 프로세스 핸들

이 플러그인이 사용하는 스캐닝 기술로 인해, 활성화된 콘솔과 종료된 콘솔에 상관없이 명령어를 찾을 수 있다.

```
$ python vol.py -f VistaSP2x64.vmem --profile=VistaSP2x64 cmdscan
Volatile Systems Volatility Framework 2.1_alpha

*****
CommandProcess: csrss.exe Pid: 528
CommandHistory: 0x135ec00 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 18 LastAdded: 17 LastDisplayed: 17
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x330
Cmd #0 @ 0x135ef10: cd \
Cmd #1 @ 0x135ef50: cd de
Cmd #2 @ 0x135ef70: cd PerfLogs
Cmd #3 @ 0x135ef90: cd ..
Cmd #4 @ 0x5c78b90: cd "Program Files"
Cmd #5 @ 0x135fae0: cd "Debugging Tools for Windows (x64)"
Cmd #6 @ 0x135efb0: livekd -w
Cmd #7 @ 0x135f010: windbg
Cmd #8 @ 0x135efd0: cd \
Cmd #9 @ 0x135fd20: rundll32 c:\apphelp.dll,ExportFunc
Cmd #10 @ 0x5c8bdb0: rundll32 c:\windows_apphelp.dll,ExportFunc
Cmd #11 @ 0x5c8be10: rundll32 c:\windows_apphelp.dll
Cmd #12 @ 0x135ee30: rundll32 c:\windows_apphelp.dll,Test
Cmd #13 @ 0x135fd70: cd "Program Files"
Cmd #14 @ 0x5c8b9e0: dir
Cmd #15 @ 0x5c8be60: cd "Debugging Tools for Windows (x64)"
Cmd #16 @ 0x5c8ba00: dir
Cmd #17 @ 0x135eff0: livekd -w
```

[snip]

3.10. consoles

cmdscan 과 비슷하게, consoles 플러그인은 공격자가 cmd.exe 에서 입력하거나 백도어를 경유하여 실행되는 명령어들을 찾는다. 하지만 이 플러그인은 COMMAND_HISTORY 를 스캐닝하는 대신에 CONSOLE_INFORMATION 을 스캐닝한다. 이 플러그인의 주요 장점은 공격자가 입력한 명령을 출력해줄뿐만 아니라, 스크린 버퍼(입력과 출력)를 모두 수집한다. 예를 들어, 단지 "dir"만 보는 것이 아닌, 공격자가 보는 것과 똑같이 "dir" 명령어를 통한 모든 파일과 디렉토리의 목록까지 볼수 있다.

추가로, 이 플러그인이 출력하는 내용은 아래와 같다:

- 원본 콘솔 창 이름과 현재 콘솔 창 이름
- 첨부된 프로세스들의 pid 와 이름(하나가 아닐 경우 LIST_ENTRY 를 순회하면서 열거한다)
- 특정 명령어 실행과 연관된 앨리어스. 예를 들면, 공격자는 "hello" 입력시 실제로는 "cd system"이 실행되는 앨리어스를 등록 할 수 있다.
- cmd.exe 콘솔의 화면 위치

여기에 consoles 명령어의 예시가 있다. 더 많은 정보를 위해 한 파일에 일반적인 이미지로부터 여러가지 출력 예시를 담아놓았다. [issue #147 에 첨부된 cmd_history.txt](#) 를 보시오. 아래에 꽤 재미있는 것들을 볼수가 있다. 포렌식 조서관은 메모리를 덤프뜨기 위해 dd.exe 를 사용할 마음이 사라질 것이다. 20 번째 줄 뒤로 사용자가 쓴 툴들을 찾고 사용할 수 있다.

```
$ python vol.py -f xp-laptop-2005-07-04-1430.img consoles
Volatile Systems Volatility Framework 2.1_alpha

[csrss.exe @ 0x821c11a8 pid 456 console @ 0x4e23b0]
  OriginalTitle: '%SystemRoot%\system32\cmd.exe'
  Title: 'C:\WINDOWS\system32\cmd.exe - dd if=\\\\.\\PhysicalMemory of=c:\\xp-2005-07-04-1430.img
conv=noerror'
  HistoryBufferCount: 2
  HistoryBufferMax: 4
  CommandHistorySize: 50
[history @ 0x4e4008]
  CommandCount: 0
  CommandCountMax: 50
  Application: 'dd.exe'
[history @ 0x4e4d88]
  CommandCount: 20
  CommandCountMax: 50
  Application: 'cmd.exe'
  Cmd #0 @ 0x4e1f90: 'dd'
  Cmd #1 @ 0x4e2cb8: 'cd\\'
  Cmd #2 @ 0x4e2d18: 'dr'
  Cmd #3 @ 0x4e2d28: 'ee:'
  Cmd #4 @ 0x4e2d38: 'e;'
  Cmd #5 @ 0x4e2d48: 'e:'
  Cmd #6 @ 0x4e2d58: 'dr'
  Cmd #7 @ 0x4e2d68: 'd;'
  Cmd #8 @ 0x4e2d78: 'd:'
  Cmd #9 @ 0x4e2d88: 'dr'
  Cmd #10 @ 0x4e2d98: 'ls'
  Cmd #11 @ 0x4e2da8: 'cd Docu'
  Cmd #12 @ 0x4e2dc0: 'cd Documents and'
```

Volatility Command 2.1 번역 문서

```
Cmd #13 @ 0x4e2e58: 'dr'
Cmd #14 @ 0x4e2e68: 'd:'
Cmd #15 @ 0x4e2e78: 'cd dd\\'
Cmd #16 @ 0x4e2e90: 'cd UnicodeRelease'
Cmd #17 @ 0x4e2ec0: 'dr'
Cmd #18 @ 0x4e2ed0: 'dd '
Cmd #19 @ 0x4e4100: 'dd if=\\\\.\\PhysicalMemory of=c:\\xp-2005-07-04-1430.img conv=noerror'
[screen @ 0x4e2460 X:80 Y:300]
Output: Microsoft Windows XP [Version 5.1.2600]
Output: (C) Copyright 1985-2001 Microsoft Corp.
Output:
Output: C:\\Documents and Settings\\Sarah>dd
Output: 'dd' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\Documents and Settings\\Sarah>cd\\
Output:
Output: C:\\>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\>ee:
Output: 'ee:' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\>e;
Output: 'e' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\>e:
Output: The system cannot find the drive specified.
Output:
Output: C:\\>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\>d;
Output: 'd' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: C:\\>d:
Output:
Output: D:\\>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: D:\\>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: D:\\>ls
Output: 'ls' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: D:\\>cd Docu
Output: The system cannot find the path specified.
Output:
Output: D:\\>cd Documents and
Output: The system cannot find the path specified.
Output:
Output: D:\\>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
Output:
Output: D:\\>d:
Output:
Output: D:\\>cd dd\\
Output:
Output: D:\\dd>
Output: D:\\dd>cd UnicodeRelease
Output:
Output: D:\\dd\\UnicodeRelease>dr
Output: 'dr' is not recognized as an internal or external command,
Output: operable program or batch file.
```

Volatility Command 2.1 번역 문서

```
Output:
Output: D:\dd\UnicodeRelease>dd
Output:
Output: 0+0 records in
Output: 0+0 records out
Output: ^C
Output: D:\dd\UnicodeRelease>dd if=\\.\PhysicalMemory of=c:\xp-2005-07-04-1430.img conv=
Output: noerror
Output: Forensic Acquisition Utilities, 1, 0, 0, 1035
Output: dd, 3, 16, 2, 1035
Output: Copyright (C) 2002-2004 George M. Garner Jr.
Output:
Output: Command Line: dd if=\\.\PhysicalMemory of=c:\xp-2005-07-04-1430.img conv=noerror
Output:
Output: Based on original version developed by Paul Rubin, David MacKenzie, and Stuart K
Output: emp
Output: Microsoft Windows: Version 5.1 (Build 2600.Professional Service Pack 2)
Output:
Output: 04/07/2005 18:30:32 (UTC)
Output: 04/07/2005 14:30:32 (local time)
Output:
Output: Current User: SPLATITUDE\Sarah
Output:
Output: Total physical memory reported: 523676 KB
Output: Copying physical memory...
Output: Physical memory in the range 0x00004000-0x00004000 could not be read.
```

3.11. envvars

프로세스의 환경변수들을 보기위해 envvars 플러그인을 사용한다. 전형적으로 이는 설치된 CPU 들의 수와 하드웨어 아키텍처(하지만 kdbgscan 플러그인의 결과가 좀 더 믿을만 하다.), 프로세스의 현재 디렉토리, 임시 디렉토리, 세션 이름, 컴퓨터 이름, 유저 이름, 그리고 다른 흥미로운 아티팩트들을 보여준다.

```
$ /usr/bin/python2.6 vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 envvars
Volatile Systems Volatility Framework 2.1_alpha
```

Pid	Process	Block	Variable	Value
296	csrss.exe	0x00000000003d1320	ComSpec	C:\Windows\system32\cmd.exe
296	csrss.exe	0x00000000003d1320	FP_NO_HOST_CHECK	NO
296	csrss.exe	0x00000000003d1320	NUMBER_OF_PROCESSORS	1
296	csrss.exe	0x00000000003d1320	OS	Windows_NT
296	csrss.exe	0x00000000003d1320		
	Path			C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
296	csrss.exe	0x00000000003d1320		
	PATHEXT			.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
296	csrss.exe	0x00000000003d1320	PROCESSOR_ARCHITECTURE	AMD64
296	csrss.exe	0x00000000003d1320	PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 2
	Stepping 3, GenuineIntel			
296	csrss.exe	0x00000000003d1320	PROCESSOR_LEVEL	6
296	csrss.exe	0x00000000003d1320	PROCESSOR_REVISION	0203
296	csrss.exe	0x00000000003d1320		
	PSModulePath			C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
296	csrss.exe	0x00000000003d1320	SystemDrive	C:
296	csrss.exe	0x00000000003d1320	SystemRoot	C:\Windows
296	csrss.exe	0x00000000003d1320	TEMP	C:\Windows\TEMP
296	csrss.exe	0x00000000003d1320	TMP	C:\Windows\TEMP
296	csrss.exe	0x00000000003d1320	USERNAME	SYSTEM
296	csrss.exe	0x00000000003d1320	windir	C:\Windows

3.12. verinfo

PE 파일들에 내장된 버전 정보를 보기위해 verinfo 명령어를 사용한다. 반드시 모든 PE 파일이 버전 정보를 가지고 있는 것은 아니지만, 많은 악성코드 제작자들은 잘못된 데이터를 포함하기 위해 버전 정보를 잊어버린다. 따라서 이 명령어는 이진파일을 식별하고 다른 파일들과의 연관성을 찾는 데 도움을 준다.

이 플러그인은 contrib 디렉토리에 있고, 필요하다면 Volatility 에서 --plugins 옵션을 사용하면 볼 수 있다. 현재는 실행가능한 프로세스와 DLL 들의 버전정보의 출력만 지원하지만, 차후에 커널 모듈들까지 포함하여 확장될 것이다. 모듈 이름으로 필터링하기를 원한다면, --regex=REGEX 또는 --ignore-case 옵션들을 사용하면 된다.

```
$ python vol.py --plugins=contrib/plugins/ -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 verinfo
Volatile Systems Volatility Framework 2.1_alpha
\SystemRoot\System32\smss.exe
C:\Windows\SYSTEM32\ntdll.dll

C:\Windows\system32\csrss.exe
File version      : 6.1.7600.16385
Product version   : 6.1.7600.16385
Flags             :
OS                : Windows NT
File Type         : Application
File Date         :
CompanyName       : Microsoft Corporation
FileDescription   : Client Server Runtime Process
FileVersion       : 6.1.7600.16385 (win7_rtm.090713-1255)
InternalName      : CSRSS.Exe
LegalCopyright    : \xa9 Microsoft Corporation. All rights reserved.
OriginalFilename  : CSRSS.Exe
ProductName       : Microsoft\xae Windows\xae Operating System
ProductVersion    : 6.1.7600.16385

[snip]
```

3.13. enumfunc

이 플러그인은 프로세스들, DLL 들, 그리고 커널 드라이브들에서 불러오고 내보내는 함수들을 열거한다. 특별히, 이는 불러오고 내보내는 함수들을 내보내는 함수들부터 이름 또는 서수순으로 다룬다. 결과는 대개 매우 장황하다.(ntdll, msvcrt, 그리고 kernel32 에 의해 내보내지는 함수들만 1000 개 이상이다.)

아래보이는 것처럼 명령줄 옵션들을 통하여 기준들을 필터링하거나 현재 플러그인에서 enumfunc.py 를 찾아서 IAT 와 EAT 파싱 API 함수들을 예시처럼 사용하여 장황함을 줄일 수 있다. 예시로, apihooks 플러그인은 혹들을 확인할 때 메모리에서 함수들을 찾기위해 불러오고 내보내는 API 들에게 영향을 미칩니다.

또한 이 플러그인은 contrib 디렉토리에 있고, 경로는 --plugins 옵션으로 아래와 같이 지정할 수 있다.

Volatility Command 2.1 번역 문서

```
$ python vol.py --plugins=contrib/plugins/ -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 enumfunc -h
....
-s, --scan           Scan for objects
-P, --process-only   Process only
-K, --kernel-only    Kernel only
-I, --import-only     Imports only
-E, --export-only     Exports only
```

프로세스들과 커널 드라이버들을 찾기위해 연결 리스트들을 순회하는 것 대신에 풀 스캐너를 이용하려면 -s 옵션을 사용한다. 이는 숨겨진 드라이버들 또는 숨겨진 프로세스들의 함수들을 열거하는데 유용하다. 명령줄 옵션들을 포함한 예시는 아래와 같다.

프로세스 메모리의 내보낸 함수들을 보기위해 -P 와 -E 를 아래와 같이 쓴다:

```
$ python vol.py --plugins=contrib/plugins/ -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 enumfunc -P -E
Process      Type      Module      Ordinal      Address      Name
lsass.exe     Export    ADVAPI32.dll 1133          0x000007fe11dd34 CreateWellKnownSid
lsass.exe     Export    ADVAPI32.dll 1134          0x000007fe17a460 CredBackupCredentials
lsass.exe     Export    ADVAPI32.dll 1135          0x000007fe170590 CredDeleteA
lsass.exe     Export    ADVAPI32.dll 1136          0x000007fe1704d0 CredDeleteW
lsass.exe     Export    ADVAPI32.dll 1137          0x000007fe17a310 CredEncryptAndMarshalBinaryBlob
lsass.exe     Export    ADVAPI32.dll 1138          0x000007fe17d080 CredEnumerateA
lsass.exe     Export    ADVAPI32.dll 1139          0x000007fe17cf50 CredEnumerateW
lsass.exe     Export    ADVAPI32.dll 1140          0x000007fe17ca00 CredFindBestCredentialA
lsass.exe     Export    ADVAPI32.dll 1141          0x000007fe17c8f0 CredFindBestCredentialW
lsass.exe     Export    ADVAPI32.dll 1142          0x000007fe1730c10 CredFree
lsass.exe     Export    ADVAPI32.dll 1143          0x000007fe1630f0 CredGetSessionTypes
lsass.exe     Export    ADVAPI32.dll 1144          0x000007fe1703d0 CredGetTargetInfoA
[snip]
```

커널 메모리의 불러온 함수들을 보기위해 -K 와 -I 를 아래와 같이 쓴다:

```
$ python vol.py --plugins=contrib/plugins/ -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 enumfunc -K -I
Volatile Systems Volatility Framework 2.1_alpha
Process      Type      Module      Ordinal      Address      Name
<KERNEL>     Import    VIDEOPRT.SYS 583          0xfffff80002acc320
ntoskrnl.exe!IoRegisterPlugPlayNotification
<KERNEL>     Import    VIDEOPRT.SYS 1325         0xfffff800029f9f30
ntoskrnl.exe!RtlAppendStringToString
<KERNEL>     Import    VIDEOPRT.SYS 509          0xfffff800026d06e0
ntoskrnl.exe!IoGetAttachedDevice
<KERNEL>     Import    VIDEOPRT.SYS 443          0xfffff800028f7ec0
ntoskrnl.exe!IoBuildSynchronousFsdRequest
<KERNEL>     Import    VIDEOPRT.SYS 1466         0xfffff80002699300
ntoskrnl.exe!RtlInitUnicodeString
<KERNEL>     Import    VIDEOPRT.SYS 759          0xfffff80002697be0 ntoskrnl.exe!KeInitializeEvent
<KERNEL>     Import    VIDEOPRT.SYS 1461         0xfffff8000265e8a0 ntoskrnl.exe!RtlInitAnsiString
<KERNEL>     Import    VIDEOPRT.SYS 1966         0xfffff80002685060 ntoskrnl.exe!ZwSetValueKey
<KERNEL>     Import    VIDEOPRT.SYS 840          0xfffff80002699440
ntoskrnl.exe!KeReleaseSpinLock
<KERNEL>     Import    VIDEOPRT.SYS 1190         0xfffff800027a98b0
ntoskrnl.exe!PoRequestPowerIrp
<KERNEL>     Import    VIDEOPRT.SYS 158          0xfffff800026840f0
ntoskrnl.exe!ExInterlockedInsertTailList
<KERNEL>     Import    VIDEOPRT.SYS 1810         0xfffff80002684640 ntoskrnl.exe!ZwClose
[snip]
```


4. 프로세스 메모리

4.1. memmap

memmap 은 정확히 어떤 페이지가 메모리 상주인지 상세한 프로세스 DTB(또는 대기상태나 시스템 프로세스에서 이 플러그인을 쓴다면 커널 DTB)를 주어주고 보여준다. 페이지의 가상 주소, 페이지의 일치하는 물리적 오프셋, 그리고 페이지 사이즈를 보여준다. 맵 정보는 근본적인 주소 공간의 get_available_addresses 방법을 사용해 플러그인으로써 발생된다.

현재 2.1 에서는 새로운 DumpFileOffset 열은 CommandReference21#memdump 플러그인에 의해 생성된 덤프 파일과 memmap 의 출력이 서로 연관성 있도록 도와준다. 예를 들어 아래의 출력에 따르면, 시스템 프로세스 메모리안의 가상 주소 0x0000000000058000 의 페이지는 win7_trial_64bit.raw 파일의 오프셋 0x00000000162ed000 에서 찾을 수 있다. 시스템 프로세스의 주소 메모리를 각각의 파일로 추출하기 위해 CommandReference21#memdump 를 사용한 후에는 이 페이지를 오프셋 0x8000 에서 찾을 수 있다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 memmap -p 4
Volatile Systems Volatility Framework 2.1_alpha
System pid: 4
Virtual Physical Size DumpFileOffset
-----
0x0000000000050000 0x0000000000cbc000 0x1000 0x0
0x0000000000051000 0x000000000015ec6000 0x1000 0x1000
0x0000000000052000 0x0000000000f5e7000 0x1000 0x2000
0x0000000000053000 0x00000000005e28000 0x1000 0x3000
0x0000000000054000 0x00000000008b29000 0x1000 0x4000
0x0000000000055000 0x0000000000155b8000 0x1000 0x5000
0x0000000000056000 0x0000000000926e000 0x1000 0x6000
0x0000000000057000 0x00000000002dac000 0x1000 0x7000
0x0000000000058000 0x0000000000162ed000 0x1000 0x8000
[snip]
```

4.2. memmap

프로세스(자세한건 CommandReference21#memmap 을 보면 된다)에서 모든 메모리에 거주하고 있는 페이지들을 각각의 파일로 추출하기 위해서는, memdump 명령어를 써야한다. 출력 디렉토리는 -D 또는 --dump-dir=DIR 을 사용하면 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 memdump -p 4 -D dump/
Volatile Systems Volatility Framework 2.1_alpha
*****
Writing System [ 4 ] to 4.dmp
$ ls -alh dump/4.dmp
-rw-r--r-- 1 Michael staff 111M Jun 24 15:47 dump/4.dmp
```

CommandReference21#memmap 논의에서 시작한 입증을 결론짓기 위해서는, 매핑되고 추출된 페이지들의 관계와 관해서 이제 단언 할 수 있어야 한다.

Volatility Command 2.1 번역 문서

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 volshell
Volatile Systems Volatility Framework 2.1_alpha
Current context: process System, pid=4, ppid=0 DTB=0x187000
Welcome to volshell! Current memory image is:
file:///Users/Michael/Desktop/win7_trial_64bit.raw
To get help, type 'hh()'

>>> PAGE_SIZE = 0x1000

>>> assert self.addrspac.read(0x0000000000058000, PAGE_SIZE) == \
...     self.addrspac.base.read(0x00000000162ed000, PAGE_SIZE) == \
...     open("dump/4.dump", "rb").read()[0x8000:0x8000 + PAGE_SIZE]
>>>
```

4.3. procmemdump

실행가능한 프로세스들을 덤프하기 위해서는 procmemdump 명령어를 사용해야 한다. PE 헤더를 파싱할때 사용된 특정 정상 체크들을 우회하기 위해 --unsafe 또는 -u 플래그를 사용할 수 있다. 몇몇 악성코드는 PE 헤더에서 의도적으로 사이즈 필드를 위조해서 메모리 덤프 툴이 실패하게 만든다.

더 많은 정보는, Andreas Schuster 가 쓴 Reconstructin a Binary 의 4 파트 시리즈를 보면 된다. 또한, impscan 은 바이너리의 주소 테이블을 불러오는 것을 재구축 하는데 도움이 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 procmemdump -D dump/ -p 296
Volatile Systems Volatility Framework 2.1_alpha
*****
Dumping csrss.exe, pid: 296 output: executable.296.exe

$ file dump/executable.296.exe
dump/executable.296.exe: PE32+ executable for MS Windows (native) Mono/.Net assembly
```

4.4. procexedump

프로세스의 실행(슬랙 공간은 포함하지 않는다)을 덤프하기 위해서는, procexedump 명령어를 사용하면 된다. 문법은 procmemdump 와 동일하다.

4.5. vadinfo

vadinfo 는 프로세스의 VAD 노드들에 대한 확장된 정보를 보여주기 위한 명령어이다. 특히 다음과 같은 항목을 보여준다.

- 커널 메모리에서 MMVAD 구조의 주소
- MMVAD 구조가 적용되는 프로세스 메모리에서의 시작과 끝 지점의 가상주소
- VAD 태그
- VAD 플래그와 컨트롤 플래그 등
- 매핑된 메모리의 이름(존재할 경우)

Volatility Command 2.1 번역 문서

- 메모리 보호 상수(허가). 오리지널 보호와 현재 보호는 차이가 있다는 것을 알아야 한다. 오리지널 보호는

flProtect 패러미터에서 VirtualAlloc 으로 파생된 것이다. 예를 들어, 보호 PAGE_NOACCESS(오리지널 보호)와 함께 메모리를 보유할 수 있다. 나중에 다시 VirtualAlloc 을 실행하도록(MEM_COMMIT) 호출 할 수 있고 PAGE_READWRITE(오리지널 보호)를 명시 할 수 있다. vadinfo 명령어는 오직 오리지널 보호만을 보여준다. 따라서, 여기서는 단지 PAGE_NOACCESS 만을 볼 수 있고 이것은 이 영역의 코드를 읽고 쓰고 실행할 수 없다는 뜻은 아니다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 vadinfo -p 296
Volatile Systems Volatility Framework 2.1_alpha
*****
Pid: 296
VAD node @ 0xfffffa8000c00620 Start 0x000000007f0e0000 End 0x000000007ffdffff Tag VadS
Flags: PrivateMemory: 1, Protection: 1
Protection: PAGE_READONLY
Vad Type: VadNone

[snip]

VAD node @ 0xfffffa8000c04ce0 Start 0x000007fefcd00000 End 0x000007fefcd10fff Tag Vad
Flags: CommitCharge: 2, Protection: 7, VadType: 2
Protection: PAGE_EXECUTE_WRITECOPY
Vad Type: VadImageMap
ControlArea @fffffa8000c04d70 Segment fffff8a000c45c10
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences: 0 NumberOfPfnReferences: 13
NumberOfMappedViews: 2 NumberOfUserReferences: 2
WaitingForDeletion Event: 00000000
Control Flags: Accessed: 1, File: 1, Image: 1
FileObject @fffffa8000c074d0, Name: \Windows\System32\basesrv.dll
First prototype PTE: fffff8a000c45c58 Last contiguous PTE: ffffffffffffffff
Flags2: Inherit: 1

[snip]
```

VAD 의 더 많은 정보는 BDG 의 The VAD Tree: A Process-Eye View of Physical Memory 에서 볼 수 있다.

4.6. vadwalk

테이블 폼에서 프로세스의 VAD 를 검사하기 위해서는 vadwalk 명령어를 써야 한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 vadwalk -p 296
Volatile Systems Volatility Framework 2.1_alpha
*****
Pid: 296
Address      Parent      Left      Right      Start      End      Tag
-----
0xfffffa8000c00620 0x0000000000000000 0xfffffa8000deaa40 0xfffffa8000c043d0 0x000000007f0e0000
0x000000007ffdffff VadS
0xfffffa8000deaa40 0xfffffa8000c00620 0xfffffa8000bc4660 0xfffffa80011b8d80 0x000000000ae0000
0x000000000b1ffff VadS
0xfffffa8000bc4660 0xfffffa8000deaa40 0xfffffa8000c04260 0xfffffa8000c91010 0x00000000004d0000
0x0000000000650fff Vadm
0xfffffa8000c04260 0xfffffa8000bc4660 0xfffffa8000c82010 0xfffffa80012acce0 0x00000000002a0000
0x000000000039ffff VadS
0xfffffa8000c82010 0xfffffa8000c04260 0xfffffa8000cbce80 0xfffffa8000c00330 0x00000000001f0000
```

Volatility Command 2.1 번역 문서

```
0x00000000001f0fff Vadm
0xfffffa8000cbce80 0xfffffa8000c82010 0xfffffa8000bc4790 0xfffffa8000d9bb80 0x0000000000180000
0x0000000000181fff Vad
0xfffffa8000bc4790 0xfffffa8000cbce80 0xfffffa8000c00380 0xfffffa8000e673a0 0x0000000000100000
0x0000000000166fff Vad
0xfffffa8000c00380 0xfffffa8000bc4790 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x00000000000fffff VadS
[snip]
```

4.7. vadtrees

비주얼 트리 폼에서 VAD 노드들을 보여주기 위해서는 vadtrees 명령어를 쓰면 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 vadtrees -p 296
Volatile Systems Volatility Framework 2.1_alpha
*****
Pid: 296
0x0000000007f0e0000 - 0x0000000007ffdefff
0x0000000000ae0000 - 0x0000000000b1ffff
0x00000000004d0000 - 0x0000000000650fff
0x00000000002a0000 - 0x000000000039ffff
0x00000000001f0000 - 0x00000000001f0fff
0x0000000000180000 - 0x0000000000181fff
0x0000000000100000 - 0x0000000000166fff
0x00000000000000000 - 0x0000000000000ffff
0x00000000000170000 - 0x00000000000170fff
0x000000000001a0000 - 0x000000000001a1fff
0x00000000000190000 - 0x00000000000190fff
0x000000000001b0000 - 0x000000000001effff
0x00000000000240000 - 0x0000000000024ffff
0x00000000000210000 - 0x00000000000216fff
0x00000000000200000 - 0x0000000000020ffff
[snip]
```

만약 안정된 바이너리 트리를 Graphviz 포맷으로 보고 싶다면 --output=dot --output-file=graph.dot 을 명령어에 추가 하기만 하면 된다. 그러면 어떠한 Graphviz-compatible 뷰어로 graph.dot 을 열 수 있다.

4.8. vaddump

VAD 노드로 묘사된 페이지들의 범위를 추출하고 싶다면 vaddump 명령어를 사용하면 된다. 이것은 CommandReference21#memdump 와 비슷하고, 각각의 VAD 노드에 속하는 페이지들이 하나의 큰 복합파일 대신에 각각의 파일들(시작과 끝 지점 주소에 따라 이름이 만들어짐)에 저장되었다는 점만 다르다. 범위 안의 어떤 페이지도 메모리에 상주하지 않는다면, 그것들은 주소 공간의 zread() 방법을 이용하여 숫자 0 으로 차 있을 것이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 vaddump -D vads
Volatile Systems Volatility Framework 2.1_alpha
Pid: 4
*****
Pid: 208
*****
Pid: 296
*****
Pid: 332
```

Volatility Command 2.1 번역 문서

```
*****
Pid: 344
*****
[snip]

$ ls -alh vads
total 229536
drwxr-xr-x 107 Michael staff 3.6K Jun 24 16:15 .
drwxr-xr-x 25 Michael staff 850B Jun 24 16:14 ..
-rw-r--r-- 1 Michael staff 140K Jun 24 16:15 System.17fef9e0.00010000-00032fff.dmp
-rw-r--r-- 1 Michael staff 4.0K Jun 24 16:15 System.17fef9e0.00040000-00040fff.dmp
-rw-r--r-- 1 Michael staff 1.7M Jun 24 16:15 System.17fef9e0.76d40000-76eeafff.dmp
-rw-r--r-- 1 Michael staff 1.5M Jun 24 16:15 System.17fef9e0.76f20000-7709ffff.dmp
-rw-r--r-- 1 Michael staff 64K Jun 24 16:15 System.17fef9e0.7ffe0000-7ffeffff.dmp
-rw-r--r-- 1 Michael staff 1.0M Jun 24 16:15 csrss.exe.176006c0.00000000-000fffff.dmp
-rw-r--r-- 1 Michael staff 412K Jun 24 16:15 csrss.exe.176006c0.00100000-00166fff.dmp
-rw-r--r-- 1 Michael staff 4.0K Jun 24 16:15 csrss.exe.176006c0.00170000-00170fff.dmp
-rw-r--r-- 1 Michael staff 8.0K Jun 24 16:15 csrss.exe.176006c0.00180000-00181fff.dmp
-rw-r--r-- 1 Michael staff 4.0K Jun 24 16:15 csrss.exe.176006c0.00190000-00190fff.dmp
[snip]
```

파일들은 다음과 같이 이름이 되어있다.

ProcessName.PhysicalOffset.StartingVPN.EndingVPN.dmp

PhysicalOffset 이 필드가 존재하는 이유는 같은 이름의 두가지 프로세스들을 구별하기 위해서이다.

5. 커널 메모리와 오브젝트

5.1. modules

시스템에서 불러진 커널 드라이버들의 목록을 보려면 modules 명령어를 사용해야 한다. 이것은 PaLoadedModuleList 에 의해 지정된 LDR_DATA_TABLE_ENTRY 구조들의 이중 링크된 목록을 보여준다. CommandReference221#pslist 명령어와 비슷하게 이 명령어는 KDBG 구조를 찾는데 의지한다. 드문 경우로, 가장 적합한 KDBG 구조 주소를 찾는데 CommandReference21#kdbgscan 을 필요로 할 수도 있고 그 다음에 이 플러그인에 --kdbg=ADDRESS 처럼 제공하면 된다.

숨겨지거나 연결되지 않은 커널 드라이버들은 찾을 수 없지만, CommandReference21#modscan 플러그인은 찾아 낼 수 있다. 또한, 이 플러그인이 목록을 통한 기술을 사용함으로써, 출력에서 보여지는 모듈들의 순서는 시스템에서 불러지는 순서라고 추측 할 수 있다. 예를 들어, 아래에서 ntoskml.exe 가 첫번째 불러져 온다음에 hal.dll 이 뒤따라온다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 modules
Volatile Systems Volatility Framework 2.1_alpha
Offset(V)      Name      Base      Size File
-----
0xfffffa80004a11a0 ntoskrnl.exe 0xfffff8000261a000 0x5dd000 \SystemRoot\system32\ntoskrnl.exe
0xfffffa80004a10b0 hal.dll      0xfffff80002bf7000 0x49000 \SystemRoot\system32\hal.dll
0xfffffa80004a7950 kdcom.dll   0xfffff80000bb4000 0xa000 \SystemRoot\system32\kdcom.dll
0xfffffa80004a7860 mcupdate.dll 0xfffff88000c3a000 0x44000
\SystemRoot\system32\mcupdate_GenuineIntel.dll
0xfffffa80004a7780 PSCHED.dll  0xfffff88000c7e000 0x14000 \SystemRoot\system32\PSCHED.dll
0xfffffa80004a7690 CLFS.SYS  0xfffff88000c92000 0x5e000 \SystemRoot\system32\CLFS.SYS
0xfffffa80004a8010 CI.dll      0xfffff88000cf0000 0xc0000 \SystemRoot\system32\CI.dll
[snip]
```

이 출력은 LDR_DATA_TABLE_ENTRY 구조의 오프셋 값을 보여준다. 이것은 기본으로 설정된 가상 주소이고 -P 스위치를 사용하여 아래와 같이 물리적 주소를 나타내게 할 수도 있다. 어느 경우에도 Base 행은 커널 메모리(PE 헤더를 찾을 수 있는 곳)에서 모듈의 기본이 되는 가상 주소를 나타낸다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 modules -P
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      Name      Base      Size File
-----
0x0000000017fe01a0 ntoskrnl.exe 0xfffff8000261a000 0x5dd000
\SystemRoot\system32\ntoskrnl.exe
0x0000000017fe00b0 hal.dll      0xfffff80002bf7000 0x49000 \SystemRoot\system32\hal.dll
0x0000000017fe6950 kdcom.dll   0xfffff80000bb4000 0xa000 \SystemRoot\system32\kdcom.dll
0x0000000017fe6860 mcupdate.dll 0xfffff88000c3a000 0x44000
\SystemRoot\system32\mcupdate_GenuineIntel.dll
0x0000000017fe6780 PSCHED.dll  0xfffff88000c7e000 0x14000 \SystemRoot\system32\PSCHED.dll
0x0000000017fe6690 CLFS.SYS  0xfffff88000c92000 0x5e000 \SystemRoot\system32\CLFS.SYS
0x0000000017fe7010 CI.dll      0xfffff88000cf0000 0xc0000 \SystemRoot\system32\CI.dll
[snip]
```

5.2. modules

modscan 명령어는 pool 태그로 물리적 메모리 스캐닝을 함으로써 LDR_DATA_TABLE_ENTRY 구조들을 찾을 수 있다. 이것은 전에 불러지지 않은 드라이버들과 루트킷에 의해 숨겨지거나 연결이 끊어진 드라이버들을 발견 할 수 있다. CommandReference21#modlist 와는 다르게 결과들의 순서는 드라이버들이 불러진 순서와는 전혀 관계가 없다. 아래에 보이는 것과 같이 DumpIt.sys 는 제일 낮은 물리적 오프셋에서 발견 되었지만, 메모리 획득을 위해 사용되는 한 아마도 제일 마지막에 로드된 드라이버 일 것이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 modscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      Name      Base      Size File
-----
0x00000000173b90b0 DumpIt.sys 0xfffff88003980000 0x11000
\\??\C:\Windows\SysWOW64\Drivers\DumpIt.sys
0x000000001745b180 mouhid.sys 0xfffff880037e9000 0xd000
\SystemRoot\system32\DRIVERS\mouhid.sys
0x0000000017473010 lldio.sys 0xfffff88002585000 0x15000
\SystemRoot\system32\DRIVERS\lldio.sys
0x000000001747f010 rspndr.sys 0xfffff8800259a000 0x18000
\SystemRoot\system32\DRIVERS\rspndr.sys
0x00000000174cac40 dxg.sys 0xfffff96000440000 0x1e000
\SystemRoot\System32\drivers\dxg.sys
0x0000000017600190 monitor.sys 0xfffff8800360c000 0xe000
\SystemRoot\system32\DRIVERS\monitor.sys
0x0000000017601170 HIDPARSE.SYS 0xfffff880037de000 0x9000
\SystemRoot\system32\DRIVERS\HIDPARSE.SYS
0x0000000017604180 USB.D.SYS 0xfffff880037e7000 0x2000
\SystemRoot\system32\DRIVERS\USB.D.SYS
0x0000000017611d70 cdrom.sys 0xfffff88001944000 0x2a000
\SystemRoot\system32\DRIVERS\cdrom.sys
[snip]
```

5.3. moddump

커널 드라이버를 파일로 추출 하기 위해서는 moddump 명령어를 사용해야 한다. -D 또는 --dump-dir=DIR 로 출력 디렉토리를 제공한다. 어떠한 추가적인 파라미터들 없이는, 모든 드라이버들은 CommandReference#21modlist 로 식별되어 저장될 것이다. 만약 어느 특정한 드라이버를 원한다면 -regex=REGEX 또는 --base=BASE 로 모듈 기본 주소를 사용하여 드라이버 이름의 수식을 제공한다.

더 많은 정보는 BDG 의 Plugin Post:Moddump 를 보면 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 moddump -D drivers/
Volatile Systems Volatility Framework 2.1_alpha
Dumping ntoskrnl.exe, Base: fffff8000261a000 output: driver.fffff8000261a000.sys
Dumping hal.dll, Base: fffff80002bf7000 output: driver.fffff80002bf7000.sys
Dumping intelide.sys, Base: fffff80000e5c000 output: driver.fffff80000e5c000.sys
Dumping mouclass.sys, Base: fffff8000349b000 output: driver.fffff8000349b000.sys
Dumping msisadrv.sys, Base: fffff80000f7c000 output: driver.fffff80000f7c000.sys
Dumping ndistapi.sys, Base: fffff800035c3000 output: driver.fffff800035c3000.sys
Dumping pacer.sys, Base: fffff80002c5d000 output: driver.fffff80002c5d000.sys
Dumping WDFLDR.SYS, Base: fffff80000f0d000 output: driver.fffff80000f0d000.sys
Dumping usbhub.sys, Base: fffff800036be000 output: driver.fffff800036be000.sys
Dumping hwpolicy.sys, Base: fffff8000149f000 output: driver.fffff8000149f000.sys
Dumping kbdclass.sys, Base: fffff8000348c000 output: driver.fffff8000348c000.sys
```

Volatility Command 2.1 번역 문서

```
Dumping amdxdm.sys, Base: fffff8000c000000 output: driver.fffff8000c000000.sys
Dumping crashdump.sys, Base: fffff80003781000 output: driver.fffff80003781000.sys
Dumping swenum.sys, Base: fffff80003461000 output: driver.fffff80003461000.sys
Cannot dump TSDDD.dll at fffff96000670000
Cannot dump framebuf.dll at fffff960008a0000
[snip]
```

CommandReference21#dlldump 와 비슷하게, PE 헤더의 중요한 부분이 메모리 상주가 아니라면 드라이버의 재구축 또는 추출은 아마 실패 할 것이다. 추가적으로 드라이버가 win32k.sys 와 같이 다른 세션에서 매핑되었다면, 드라이버 샘플을 얻을때 어떤 세션이 사용되었는지 명시하는 방법은 현재로서는 없다. 이 것은 Volatility 2.2 출시때 추가될 것이다.

5.4. ssdt

Native 와 GUI SSDTs 에서 함수들을 나열하기 위해서는 ssdt 명령어를 사용해야 한다. 이것은 인덱스, 함수 이름, 그리고 SSDT 에서 각각의 엔트리를 소유하고 있는 드라이버를 보여준다. 다음을 주목해라.

- 윈도우는 기본으로 4 개의 SSDTs 를 가지고 있지만(KeAddSystemServiceTable 로 더 추가 할 수 있다), 오직 두 개만 사용된다. 하나는 NT 모듈에서 Native 함수를 위해, 그리고 나머지 하나는 win32k.sys 모듈에서 GUI 함수들을 위해 사용된다.

- 메모리에서 SSDTs 를 찾아내는 방법은 여러가지가 있다. 대부분의 툴들은 NT 모듈에서 내보내진 KeServiceDescriptorTable 기호를 찾아냄으로써 SSDTs 를 찾아내지만, Volatility 는 이처럼 하지 않는다.

- x86 시스템에서 Volatility 는 ETHREAD 오브젝트들(CommandReference#thrdscan 명령어 참조)을 스캔하고 모든 고유의 ETHREAD.Tcb.ServiceTable 포인터들을 수집한다. 이러한 방법은 루트킷이 기존의 SSDTs 를 복사하고 이 것들을 특정한 쓰레드로 지정하는 것을 탐지해 낼 수 있기 때문에 더 탄탄하고 완전하다. CommandReference#thread 명령어도 참조해라.

- x64 시스템(ETHREAD.Tcb.ServiceTable 를 가지고 있지 않는)에서 Volatility 는 nt!KeAddSystemServiceTable 에서 코드를 분해하고 KeServiceDescriptorTable 과 KeServiceDescriptorTableShadow 기호에 맞는 참조들을 찾아낸다.

- SSDTs 에서 순서와 전체 함수들의 숫자는 운영체제 버전에 따라 다 다르다. 따라서, Volatility 는 per-profile(OS) 사전에 정보를 저장한다. per-profile(OS) 사전은 각각의 시스템으로부터 ntoskrnl.exe, ntdll.dll, win32k.sys, user32.dll, 그리고 gdi32.dll 모듈들을 사용하여 자동 생성하고 전반에 걸쳐 참조한다.

Volatility Command 2.1 번역 문서

- 더 많은 정보는 BDG 의 Auditing the System Call Table 을 보면 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 ssdt
Volatile Systems Volatility Framework 2.1_alpha
[x64] Gathering all referenced SSDTs from KeAddSystemServiceTable...
Finding appropriate address space for tables...
SSDT[0] at fffff8000268cb00 with 401 entries
  Entry 0x0000: 0xfffff80002a9d190 (NtMapUserPhysicalPagesScatter) owned by ntoskrnl.exe
  Entry 0x0001: 0xfffff80002983a00 (NtWaitForSingleObject) owned by ntoskrnl.exe
  Entry 0x0002: 0xfffff80002683dd0 (NtCallbackReturn) owned by ntoskrnl.exe
  Entry 0x0003: 0xfffff800029a6b10 (NtReadFile) owned by ntoskrnl.exe
  Entry 0x0004: 0xfffff800029a4bb0 (NtDeviceIoControlFile) owned by ntoskrnl.exe
  Entry 0x0005: 0xfffff8000299fee0 (NtWriteFile) owned by ntoskrnl.exe
  Entry 0x0006: 0xfffff80002945dc0 (NtRemoveIoCompletion) owned by ntoskrnl.exe
  Entry 0x0007: 0xfffff80002942f10 (NtReleaseSemaphore) owned by ntoskrnl.exe
  Entry 0x0008: 0xfffff8000299ada0 (NtReplyWaitReceivePort) owned by ntoskrnl.exe
  Entry 0x0009: 0xfffff80002a6ce20 (NtReplyPort) owned by ntoskrnl.exe

[snip]

SSDT[1] at fffff96000101c00 with 827 entries
  Entry 0x1000: 0xfffff960000f5580 (NtUserGetThreadState) owned by win32k.sys
  Entry 0x1001: 0xfffff960000f2630 (NtUserPeekMessage) owned by win32k.sys
  Entry 0x1002: 0xfffff96000103c6c (NtUserCallOneParam) owned by win32k.sys
  Entry 0x1003: 0xfffff96000111dd0 (NtUserGetKeyState) owned by win32k.sys
  Entry 0x1004: 0xfffff9600010b1ac (NtUserInvalidateRect) owned by win32k.sys
  Entry 0x1005: 0xfffff96000103e70 (NtUserCallNoParam) owned by win32k.sys
  Entry 0x1006: 0xfffff960000fb5a0 (NtUserGetMessage) owned by win32k.sys
  Entry 0x1007: 0xfffff960000dfbec (NtUserMessageCall) owned by win32k.sys
  Entry 0x1008: 0xfffff960001056c4 (NtGdiBitBlt) owned by win32k.sys
  Entry 0x1009: 0xfffff960001fd750 (NtGdiGetCharSet) owned by win32k.sys

[snip]
```

ntoskrnl.exe 과 win32k.sys 를 가리키는 모든 함수들을 걸러내기 위해서는 명령어 라인에 egrep 을 사용하면 된다. 이것은 연결된 SSDT 함수들만 보여줄 것이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 ssdt | egrep -v '(ntos|win32k)'
```

당신의 시스템에서 NT 모듈은 아마도 ntkrnlpa.exe 또는 ntkrnlmp.exe 일 것이다. 따라서 잘못된 모듈 이름을 필터링 할 수 있으니 egrep 을 사용하기 전에 반드시 확인해야 한다. win32ktesting.sys 로 명명된 드라이버를 불러올 수 있는 악성코드는 필터를 우회할 수 있기 때문에 이 방법은 hook 들을 찾아내는 확립된 기술이 아니란 것을 명심해야 한다.

5.5. driverscan

pool 태그 스캐닝을 사용하여 물리적 메모리에서 DRIVER_OBJECT 들을 찾아내기 위해서는 driverscan 명령어를 사용하면 된다. 이것은 모든 커널 모듈들이 연관된 DRIVER_OBJECT 를 가지고 있지는 않지만, 커널 모듈을 찾아내는 또 다른 방법이다. DRIVER_OBJECT 는 28IRP(주요 함수) 테이블들을 포함하고 있고, 이 때문에 CommandReference21#driverirp 명령어는 driverscan 에 의해 사용된 방법론에 기초하고 있다.

더 많은 정보는 Andreas Schuster 의 Scanning for Drivers 를 확인하면 된다.

Volatility Command 2.1 번역 문서

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 driverscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      #Ptr #Hnd Start      Size Service Key      Name      Driver Name
-----
0x00000000174c6350 3 0 0xfffff880037e9000 0xd000
mouhid          mouhid      \Driver\mouhid
0x0000000017660cb0 3 0 0xfffff8800259a000 0x18000 rspndr      rspndr      \Driver\rspndr
0x0000000017663e70 3 0 0xfffff88002585000 0x15000 lldio       lldio       \Driver\lldio
0x0000000017691d70 3 0 0xfffff88001944000 0x2a000 cdrom       cdrom       \Driver\cdrom
0x0000000017692a50 3 0 0xfffff8800196e000 0x9000 Null       Null       \Driver\Null
0x0000000017695e70 3 0 0xfffff88001977000 0x7000 Beep       Beep       \Driver\Beep
0x00000000176965c0 3 0 0xfffff8800197e000 0xe000
VgaSave         VgaSave     \Driver\VgaSave
0x000000001769fb00 4 0 0xfffff880019c1000 0x9000
RDPCDD          RDPCDD      \Driver\RDPCDD
0x00000000176a1720 3 0 0xfffff880019ca000 0x9000
RDPENCDD        RDPENCDD    \Driver\RDPENCDD
0x00000000176a2230 3 0 0xfffff880019d3000 0x9000
RDPREFMP        RDPREFMP    \Driver\RDPREFMP
[snip]
```

5.6. filescan

pool 태그 스캐닝을 사용하여 물리적 메모리에서 FILE_OBJECT 들을 찾아내려면 filescan 명령어를 사용해야 한다. 이것은 루트킷이 디스크에서 파일을 숨겼을지라도 열려있는 파일들을 찾아낼 것이고 루트킷이 활성 시스템에서 열린 핸들을 숨기도록 몇몇 API 함수들을 연결시켜도 찾아낼 것이다. 출력 값은 FILE_OBJECT 의 물리적 오프셋, 파일이름, 오브젝트에 연결된 포인터 갯수, 오브젝트의 핸들 갯수, 그리고 오브젝트에 수여된 실질적 권한들이 보여질 것이다.

더 많은 정보는 Andreas Schuster 의 Scanning for File Objects 와 Linking File Objects To Processes 를 확인하면 된다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 filescan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      #Ptr #Hnd Access Name
-----
0x000000000126f3a0 14 0 R--r-d \Windows\System32\mswsock.dll
0x000000000126fdc0 11 0 R--r-d \Windows\System32\ssdpsrv.dll
0x000000000468f7e0 6 0 R--r-d \Windows\System32\cryptsp.dll
0x000000000468fdc0 16 0 R--r-d \Windows\System32\Applpdm.dll
0x00000000048223a0 1 1 ----- \Endpoint
0x0000000004822a30 16 0 R--r-d \Windows\System32\kerberos.dll
0x0000000004906070 13 0 R--r-d \Windows\System32\wbem\repdrvfs.dll
0x0000000004906580 9 0 R--r-d \Windows\SysWOW64\netprofm.dll
0x0000000004906bf0 9 0 R--r-d \Windows\System32\wbem\wmiutils.dll
0x00000000049ce8e0 2 1 R--rwd \${Extend}\$ObjId
0x00000000049cedd0 1 1 R--r-d \Windows\System32\en-US\vsstrace.dll.mui
0x0000000004a71070 17 1 R--r-d \Windows\System32\en-US\pnidui.dll.mui
0x0000000004a71440 11 0 R--r-d \Windows\System32\nci.dll
0x0000000004a719c0 1 1 ----- \srsvsc
[snip]
```

5.7. mutantscan

pool 태그 스캐닝을 사용해 KMUTANT 오브젝트의 물리적 메모리를 스캔하려면, mutantscan 명령어를 사용해야 한다. 기본으로 이것은 모든 오브젝트들을 보여주지만 mutexes 로 명명된 것만 볼 수 있도록 -s 또는 --silent 를 사용할 수 있다. CID 행은 존재할 경우 mutex 소유자의 프로세스 ID 와 쓰레드 ID 를 담고 있다.

더 많은 정보는 Andreas Schuster 의 Searching for Mutants 를 확인하면 된다.

```
$ python -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 mutantscan --silent
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      #Ptr #Hnd Signal Thread      CID Name
-----
0x00000000f702630 2 1 1 0x0000000000000000 {A3BD3259-3E4F-428a-84C8-F0463A9D3EB5}
0x00000000102fd930 2 1 1 0x0000000000000000 Feed Arbitration Shared Memory Mutex [ User :
S-1-5-21-2628989162-3383567662-1028919141-1000 ]
0x00000000104e5e60 3 2 1 0x0000000000000000 ZoneAttributeCacheCounterMutex
0x0000000010c29e40 2 1 1 0x0000000000000000 _!MSFTHISTORY!_ LOW!_
0x0000000013035080 2 1 1 0x0000000000000000 c:\users\testing\appdata\local\microsoft\feeds
cache!
0x000000001722dfc0 2 1 1
0x0000000000000000 c:\users\testing\appdata\roaming\microsoft\windows\ietldcache\low!
0x00000000172497f0 2 1 1 0x0000000000000000 LRIEElevationPolicyMutex
0x000000001724bfc0 3 2 1 0x0000000000000000 !BrowserEmulation!SharedMemory!Mutex
0x000000001724f400 2 1 1
0x0000000000000000 c:\users\testing\appdata\local\microsoft\windows\history\low\history.ie5\mshist01201
2022220120223!
0x000000001724f4c0 4 3 1 0x0000000000000000 _!SHMSFTHISTORY!_
0x00000000172517c0 2 1 1 0x0000000000000000 _DDrawExclMode_
0x00000000172783a0 2 1 1 0x0000000000000000 Lowhttp://sourceforge.net/
0x00000000172db840 4 3 1 0x0000000000000000 ConnHashTable<1892>_HashTable_Mutex
0x00000000172de1d0 2 1 1 0x0000000000000000 Feeds Store Mutex S-1-5-21-2628989162-
3383567662-1028919141-1000
0x00000000173b8080 2 1 1 0x0000000000000000 DDrawDriverObjectListMutex
0x00000000173bd340 2 1 0 0xfffffa8000a216d0 1652:2000 ALTTAB_RUNNING_MUTEX
0x0000000017449c40 2 1 1 0x0000000000000000 DDrawWindowListMutex
[snip]
```

5.8. symlinkscan

이 플러그인 스캔은 상징적 오브젝트 링크들과 그것들에 대한 정보 출력을 위한 것이다.

```
$ python -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 symlinkscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      #Ptr #Hnd Creation
time          From      To
-----
0x000000000469780 1 0 2012-02-22 20:03:13 UMB#UMB#1...e1ba19f}
\Device\00000048
0x000000000754560 1 0 2012-02-22
20:03:15 ASYNCMAC \Device\ASYNCMAC
0x000000000ef6cf0 2 1 2012-02-22
19:58:24 0 \BaseNamedObjects
0x0000000014b2a10 1 0 2012-02-22
20:02:10 LanmanRedirector \Device\Mup\;LanmanRedirector
0x0000000053e56f0 1 0 2012-02-22 20:03:15 SW#{eeab7...abac361}
\Device\KSENUM#00000001
0x000000005cc0770 1 0 2012-02-22
19:58:20 WanArpV6 \Device\WANARPV6
0x000000005cc0820 1 0 2012-02-22
19:58:20 WanArp \Device\WANARP
0x0000000008ffa680 1 0 2012-02-22
```

Volatility Command 2.1 번역 문서

```

19:58:24 Global \BaseNamedObjects
0x0000000009594810 1 0 2012-02-22
19:58:24 KnownDllPath C:\Windows\syswow64
0x000000000968f5f0 1 0 2012-02-22
19:58:23 KnownDllPath C:\Windows\system32
0x000000000ab24060 1 0 2012-02-22 19:58:20 Volume{3b...f6e6963}
\Device\CdRom0
0x000000000ab24220 1 0 2012-02-22 19:58:21 {EE0434CC...863ACC2}
\Device\NDMP2
0x000000000abd3460 1 0 2012-02-22 19:58:21 ACPI#PNP0...91405dd}
\Device\00000041
0x000000000abd36f0 1 0 2012-02-22 19:58:21 {802389A0...A90C31A} \Device\NDMP3
[snip]

```

5.9. thrdsan

pool 태크 스캐닝을 가지고 물리적 메모리에서 ETHREAD 오브젝트들을 찾기 위해서는 thrdsan 명령어를 사용해야 한다. ETHREAD 는 부모 프로세스를 식별하는 필드를 담고 있기 때문에, 이 기술을 숨겨진 프로세스들을 찾는데 사용 할 수 있다. 하나의 실 사례는 CommandReference21#psxview 에 설명되었다. 또한 더 큰 세부정보를 원한다면, CommandReference21#threads 플러그인을 시도해 보면 된다.

```

$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 thrdsan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) PID TID Start Address Create Time Exit Time
-----
0x0000000008df68d0 280 392 0x77943260 2012-02-22 19:08:18
0x000000000eac3850 2040 144 0x76d73260 2012-02-22 11:28:59 2012-02-22 11:29:04
0x000000000fd82590 880 1944 0x76d73260 2012-02-22 20:02:29 2012-02-22 20:02:29
0x000000000103d15f0 880 884 0x76d73260 2012-02-22 19:58:43
0x000000000103e5480 1652 1788 0xfffff8a0010ed490 2012-02-22 20:03:44
0x000000000105a3940 916 324 0x76d73260 2012-02-22 20:02:07 2012-02-22 20:02:09
0x000000000105b3560 816 824 0x76d73260 2012-02-22 19:58:42
0x000000000106d1710 916 1228 0x76d73260 2012-02-22 20:02:11
0x00000000010a349a0 816 820 0x76d73260 2012-02-22 19:58:41
0x00000000010bd1060 1892 2280 0x76d73260 2012-02-22 11:26:13
0x00000000010f24230 628 660 0x76d73260 2012-02-22 19:58:34
0x00000000010f27060 568 648 0xfffff8a0017c6650 2012-02-22 19:58:34
[snip]

```

6. 네트워크

6.1. connections

메모리 획득 당시 활성화된 TCP 연결들을 보기 위해 connections 명령어를 사용한다. 이는 tcpip.sys 모듈의 non-exported symbol 이 가리키는 connection 구조체들의 단일 리스트를 순회한다.

이 명령어는 x86 과 x64 Windows XP 그리고 Windows 2003 Server 에서만 동작한다.

```
$ python vol.py -f Win2003SP2x64.vmem --profile=Win2003SP2x64 connections
Volatile Systems Volatility Framework 2.1_alpha
Offset(V)      Local Address      Remote Address      Pid
-----
0xfffffadfe6f2e2f0 172.16.237.150:1408 72.246.25.25:80      2136
0xfffffadfe72e8080 172.16.237.150:1369 64.4.11.30:80        2136
0xfffffadfe622d010 172.16.237.150:1403 74.125.229.188:80    2136
0xfffffadfe62e09e0 172.16.237.150:1352 64.4.11.20:80        2136
0xfffffadfe6f2e630 172.16.237.150:1389 209.191.122.70:80    2136
0xfffffadfe5e7a610 172.16.237.150:1419 74.125.229.187:80    2136
0xfffffadfe7321bc0 172.16.237.150:1418 74.125.229.188:80    2136
0xfffffadfe5ea3c90 172.16.237.150:1393 216.115.98.241:80    2136
0xfffffadfe72a3a80 172.16.237.150:1391 209.191.122.70:80    2136
0xfffffadfe5ed8560 172.16.237.150:1402 74.125.229.188:80    2136
```

기본적으로 출력은 _TCPT_OBJECT 의 가상 오프셋 포함한다. 물리 오프셋을 얻고 싶다면 -P 옵션을 사용하면 된다.

6.2. connscan

풀 태그 스캐닝을 사용하여 _TCPT_OBJECT 구조체를 찾기위해, connscan 명령어를 사용한다. 이는 이미 종료되어버린 이전의 연결들로부터 아티팩트를 찾아 낼 수 있고, 게다가 활성화된 것들도 찾아 낼 수 있다. 아래의 출력에서, 부분적으로 덮어쓰여진 몇몇 필드가 보이지만, 몇몇의 정보는 여전히 정확하다. 예를 들면, 마지막 줄의 Pid 필드는 0 이지만 다른 필드들은 여전히 정확하다. 따라서, 때때로 오탐을 발견할 수 있지만, 가능한 많은 정보를 찾을 수 있다는 이점이 있다.

이 명령어는 x86 과 x64 Windows XP 그리고 Windows 2003 Server 에서만 동작한다.

```
$ python vol.py -f Win2K3SP0x64.vmem --profile=Win2003SP2x64 connscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      Local Address      Remote Address      Pid
-----
0x0ea7a610 172.16.237.150:1419 74.125.229.187:80    2136
0x0eaa3c90 172.16.237.150:1393 216.115.98.241:80    2136
0x0eaa4480 172.16.237.150:1398 216.115.98.241:80    2136
0x0ead8560 172.16.237.150:1402 74.125.229.188:80    2136
0x0ee2d010 172.16.237.150:1403 74.125.229.188:80    2136
0x0eee09e0 172.16.237.150:1352 64.4.11.20:80        2136
0x0f9f83c0 172.16.237.150:1425 98.139.240.23:80     2136
```

Volatility Command 2.1 번역 문서

0x0f9fe010	172.16.237.150:1394	216.115.98.241:80	2136
0x0fb2e2f0	172.16.237.150:1408	72.246.25.25:80	2136
0x0fb2e630	172.16.237.150:1389	209.191.122.70:80	2136
0x0fb72730	172.16.237.150:1424	98.139.240.23:80	2136
0x0fea3a80	172.16.237.150:1391	209.191.122.70:80	2136
0x0fee8080	172.16.237.150:1369	64.4.11.30:80	2136
0x0ff21bc0	172.16.237.150:1418	74.125.229.188:80	2136
0x1019ec90	172.16.237.150:1397	216.115.98.241:80	2136
0x179099e0	172.16.237.150:1115	66.150.117.33:80	2856
0x2cdb1bf0	172.16.237.150:139	172.16.237.1:63369	4
0x339c2c00	172.16.237.150:1138	23.45.66.43:80	1332
0x39b10010	172.16.237.150:1148	172.16.237.138:139	0

6.3. sockets

어떤 프로토콜(TCP, UDP, RAW 등등)의 listen 중인 소켓들을 찾기위해, sockets 명령어를 사용한다. 이는 tcpip.sys 모듈의 non-exported symbol 이 가리키는 socket 구조체들의 단일 리스트를 순회한다.

이 명령어는 x86 과 x64 Windows XP 그리고 Windows 2003 Server 에서만 동작한다.

```
$ python vol.py -f Win2K3SP0x64.vmem --profile=Win2003SP2x64 sockets
Volatile Systems Volatility Framework 2.1_alpha
Offset(V)      PID  Port  Proto Protocol  Address      Create Time
-----
0xfffffadfe71bbda0 432  1025   6 TCP      0.0.0.0      2012-01-23 18:20:01
0xfffffadfe7350490 776  1028  17 UDP      0.0.0.0      2012-01-23 18:21:44
0xfffffadfe6281120 804  123   17 UDP     127.0.0.1    2012-06-25 12:40:55
0xfffffadfe7549010 432  500   17 UDP      0.0.0.0      2012-01-23 18:20:09
0xfffffadfe5ee8400 4    0     47 GRE      0.0.0.0      2012-02-24 18:09:07
0xfffffadfe606dc90 4    445   6 TCP      0.0.0.0      2012-01-23 18:19:38
0xfffffadfe6eef770 4    445  17 UDP      0.0.0.0      2012-01-23 18:19:38
0xfffffadfe7055210 2136 1321  17 UDP     127.0.0.1    2012-05-09 02:09:59
0xfffffadfe750c010 4    139   6 TCP     172.16.237.150 2012-06-25 12:40:55
0xfffffadfe745f610 4    138  17 UDP     172.16.237.150 2012-06-25 12:40:55
0xfffffadfe6096560 4    137  17 UDP     172.16.237.150 2012-06-25 12:40:55
0xfffffadfe7236da0 720  135   6 TCP      0.0.0.0      2012-01-23 18:19:51
0xfffffadfe755c5b0 2136 1419  6 TCP      0.0.0.0      2012-06-25 12:42:37
0xfffffadfe6f36510 2136 1418  6 TCP      0.0.0.0      2012-06-25 12:42:37
[snip]
```

기본적으로 출력은 _ADDRESS_OBJECT 의 가상 오프셋을 포함한다. 물리 오프셋을 얻고 싶다면 -P 옵션을 사용하면 된다.

6.4. sockscan

풀 태그 스캐닝을 사용하여 _ADDRESS_OBJECT 구조체를 찾기위해, sockscan 명령어를 사용한다. connscan 과 마찬가지로, 이는 이전의 socket 으로부터 존재했던 데이터와 아티팩트들을 추출할 수있다.

이 명령어는 x86 과 x64 Windows XP 그리고 Windows 2003 서버에서만 동작한다.

Volatility Command 2.1 번역 문서

```
$ python vol.py -f Win2K3SP0x64.vmem --profile=Win2003SP2x64 sockscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) PID Port Proto Protocol Address Create Time
-----
0x000000000608010 804 123 17 UDP 172.16.237.150 2012-05-08 22:17:44
0x00000000eae8400 4 0 47 GRE 0.0.0.0 2012-02-24 18:09:07
0x00000000eaf1240 2136 1403 6 TCP 0.0.0.0 2012-06-25 12:42:37
0x00000000ec6dc90 4 445 6 TCP 0.0.0.0 2012-01-23 18:19:38
0x00000000ec96560 4 137 17 UDP 172.16.237.150 2012-06-25 12:40:55
0x00000000ecf7d20 2136 1408 6 TCP 0.0.0.0 2012-06-25 12:42:37
0x00000000ed5a010 2136 1352 6 TCP 0.0.0.0 2012-06-25 12:42:18
0x00000000ed84ca0 804 123 17 UDP 172.16.237.150 2012-06-25 12:40:55
0x00000000ee2d380 2136 1393 6 TCP 0.0.0.0 2012-06-25 12:42:37
0x00000000ee81120 804 123 17 UDP 127.0.0.1 2012-06-25 12:40:55
0x00000000eeda8c0 776 1363 17 UDP 0.0.0.0 2012-06-25 12:42:20
0x00000000f0be1a0 2136 1402 6 TCP 0.0.0.0 2012-06-25 12:42:37
0x00000000f0d0890 4 1133 6 TCP 0.0.0.0 2012-02-24 18:09:07
[snip]
```

6.5. netscan

32bit 와 64bit Windows Vista, Windows 2008 Server 그리고 Windows 7 메모리 덤프에서 네트워크 아티팩트들을 찾기위해, netscan 명령어를 사용한다. 이는 TCP endpoints, TCP listeners, UDP endpoints, and UDP listeners 를 찾아준다. 이는 IPv4 와 IPv6 을 구별하며 Local 과 Remote IP(가능하다면), Local 과 Remote 포트(가능하다면), 소켓이 bind 되거나 연결이 수립된 시간, 그리고 현재 상태(TCP 연결만 지원)를 보여준다. 더 많은 정보를 원한다면 Volatility 의 새로운 Netscan 모듈을 보라.

아래를 주목하라:

- netscan 명령어는 풀 태그 스캐닝을 사용한다.
- Vista 이상의 운영체제에서는 연결 과 소켓들을 열거하기위해 적어도 2 가지 방법을 번갈아 사용한다. 그중 하나는 윈도우 시스템에서 netstat.exe 도구가 동작하는것 처럼 파티션과 동적 해시 테이블을 사용하는 것이다. 다른 것들은 bitmap 과 port pool 이다. 이런 두가지 전체 방법을 위한 플러그인들이 Volatility 2.1 에 존재하지만, 현재 대중적이지는 않다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 netscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) Proto Local Address Foreign Address State Pid Owner Created
-----
0xf882a30 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 628 svchost.exe
0xfc13770 TCPv4 0.0.0.0:49154 0.0.0.0:0 LISTENING 916 svchost.exe
0xfdda1e0 TCPv4 0.0.0.0:49154 0.0.0.0:0 LISTENING 916 svchost.exe
0xfdda1e0 TCPv6 :::49154 :::0 LISTENING 916 svchost.exe
0x1121b7b0 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 628 svchost.exe
0x1121b7b0 TCPv6 :::135 :::0 LISTENING 628 svchost.exe
0x11431360 TCPv4 0.0.0.0:49152 0.0.0.0:0 LISTENING 332 wininit.exe
0x11431360 TCPv6 :::49152 :::0 LISTENING 332 wininit.exe
[snip]
0x17de8980 TCPv6 :::49153 :::0 LISTENING 444 lsass.exe
0x17f35240 TCPv4 0.0.0.0:49155 0.0.0.0:0 LISTENING 880 svchost.exe
0x17f362b0 TCPv4 0.0.0.0:49155 0.0.0.0:0 LISTENING 880 svchost.exe
```


Volatility Command 2.1 번역 문서

```

0x17f362b0 TCPv6 :::49155 :::0 LISTENING 880 svchost.exe
0xfd96570 TCPv4 -:0 232.9.125.0:0 CLOSED 1 ?C?
0x17236010 TCPv4 -:49227 184.26.31.55:80 CLOSED 2820 iexplore.exe
0x1725d010 TCPv4 -:49359 93.184.220.20:80 CLOSED 2820 iexplore.exe
0x17270530 TCPv4 10.0.2.15:49363 173.194.35.38:80 ESTABLISHED 2820 iexplore.exe
0x17285010 TCPv4 -:49341 82.165.218.111:80 CLOSED 2820 iexplore.exe
0x17288a90 TCPv4 10.0.2.15:49254 74.125.31.157:80 CLOSE_WAIT 2820 iexplore.exe
0x1728f6b0 TCPv4 10.0.2.15:49171 204.245.34.130:80 ESTABLISHED 2820 iexplore.exe
0x17291ba0 TCPv4 10.0.2.15:49347 173.194.35.36:80 CLOSE_WAIT 2820 iexplore.exe

```

[snip]

```

0x17854010 TCPv4 -:49168 157.55.15.32:80 CLOSED 2820 iexplore.exe
0x178a2a20 TCPv4 -:0 88.183.123.0:0 CLOSED 504 svchost.exe
0x178f5b00 TCPv4 10.0.2.15:49362 173.194.35.38:80 CLOSE_WAIT 2820 iexplore.exe
0x17922910 TCPv4 -:49262 184.26.31.55:80 CLOSED 2820 iexplore.exe
0x17a9d860 TCPv4 10.0.2.15:49221 204.245.34.130:80 ESTABLISHED 2820 iexplore.exe
0x17ac84d0 TCPv4 10.0.2.15:49241 74.125.31.157:80 CLOSE_WAIT 2820 iexplore.exe
0x17b9acf0 TCPv4 10.0.2.15:49319 74.125.127.148:80 CLOSE_WAIT 2820 iexplore.exe
0x10f38d70 UDPv4 10.0.2.15:1900 *:1736 svchost.exe 2012-
02-22 20:04:12
0x173b3dc0 UDPv4 0.0.0.0:59362 *:1736 svchost.exe 2012-
02-22 20:02:27
0x173b3dc0 UDPv6 :::59362 *:1736 svchost.exe 2012-02-
22 20:02:27
0x173b4cf0 UDPv4 0.0.0.0:3702 *:1736 svchost.exe 2012-02-
22 20:02:27
0x173b4cf0 UDPv6 :::3702 *:1736 svchost.exe 2012-02-
22 20:02:27

```

[snip]

7. 레지스트리

7.1. hivescan

메모리 안에 있는 CMHIVES (레지스터리 hives) 물리적인 주소를 검색하기 위해서, hivescan 명령어를 사용한다. 더 많은 정보를 원하면, BDG's [Enumerating Registry Hives](#) 를 참고하라.

이 플러그인은 많이 사용되는 편은 아니다. 아래 설명드릴 CMHIVES 안에 있는 찾은 정보를 생성하고 제어하는 다른 플러그인([CommandReference21#hivelist](#))에 의해 상속 받았다.

```
$python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 hivescan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)
-----
0x0000000008c95010
0x000000000aa1a010
0x000000000acf9010
0x000000000b1a9010
0x000000000c2b4010
0x000000000cd20010
0x000000000da51010
[snip]
```

7.2. hivelist

메모리의 레지스터리 하이브들(hives)의 가상메모리 주소들을 위치 파악하고, 디스크의 관련된 하이브에 전체 경로를 표시 되기 위해서는 hivelist 명령어를 사용한다. 만약 특정 하이브로부터 값을 표시하기 원한다면, 이 명령어를 실행시켜서 그 하이브의 주소 값을 볼 수 있다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 hivelist
Volatile Systems Volatility Framework 2.1_alpha
Virtual          Physical          Name
-----
0xfffff8a001053010 0x000000000b1a9010 \??\C:\System Volume Information\Syscache.hve
0xfffff8a0016a7420 0x00000000012329420 \REGISTRY\MACHINE\SAM
0xfffff8a0017462a0 0x000000000101822a0 \??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0xfffff8a001abe420 0x000000000eae0420 \??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0xfffff8a002ccf010 0x00000000014659010 \??\C:\Users\testing\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff80002b53b10 0x000000000a441b10 [no name]
0xfffff8a00000d010 0x000000000ddc6010 [no name]
0xfffff8a000022010 0x000000000da51010 \REGISTRY\MACHINE\SYSTEM
0xfffff8a00005c010 0x000000000dacd010 \REGISTRY\MACHINE\HARDWARE
0xfffff8a00021d010 0x000000000cd20010 \SystemRoot\System32\Config\SECURITY
0xfffff8a0009f1010 0x000000000aa1a010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8a000a15010 0x000000000acf9010 \SystemRoot\System32\Config\SOFTWARE
0xfffff8a000ce5010 0x0000000008c95010 \SystemRoot\System32\Config\DEFAULT
0xfffff8a000f95010 0x000000000c2b4010 \??\C:\Users\testing\ntuser.dat
```

7.3. printkey

특정 레지스트리 키에 포함되어 있는 서브키(subkeys), 값(values), 데이터(data)을 표시하기 위해서, printkey 명령어를 사용한다. 기본적으로, printkey 를 모든 하이브들(hives)을 검색하고, 요청한 키의 키 정보(만약 찾는다면)를 출력한다. 그래서 그 키가 하나의 하이브보다 더 위치하고 있다면, 그 값을 위한 정보는 그것이 포함되어 있는 각 하이브에 출력이 된다.

HKEY_LOCAL_MACHINE\Microsoft\Security Center\Svc key 안을 둘러보고 싶다면, 아래와 같은 방식으로 진행할 수 있다.

참고 : Windows 환경에서의 Volatility 를 사용하고 있다면, 더블 쿼터 (") 안에 키 값을 포함시켜야 한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 printkey -K "Microsoft\Security Center\Svc"
Volatile Systems Volatility Framework 2.1_alpha
Legend: (S) = Stable (V) = Volatile

-----
Registry: \SystemRoot\System32\Config\SOFTWARE
Key name: Svc (S)
Last updated: 2012-02-22 20:04:44

Subkeys:
(V) Vol

Values:
REG_QWORD VistaSp1 : (S) 128920218544262440
REG_DWORD AntiSpywareOverride : (S) 0
REG_DWORD ConfigMask : (S) 4361
```

Multiple 하이브들이 같은 키 "Software\Microsoft\Windows NT\CurrentVersion" 이 포함될때, 아래 예제에서 어떻게 출력이 되는지 볼 수 있다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 printkey -K "Software\Microsoft\Windows NT\CurrentVersion"
Volatile Systems Volatility Framework 2.1_alpha
Legend: (S) = Stable (V) = Volatile

-----
Registry: \SystemRoot\System32\Config\DEFAULT
Key name: CurrentVersion (S)
Last updated: 2009-07-14 04:53:31

Subkeys:
(S) Devices
(S) PrinterPorts

Values:
-----
Registry: \??\C:\Users\testing\ntuser.dat
Key name: CurrentVersion (S)
Last updated: 2012-02-22 11:26:13

Subkeys:
(S) Devices
(S) EFS
(S) MsiCorruptedFileRecovery
(S) Network
(S) PeerNet
(S) PrinterPorts
```

Volatility Command 2.1 번역 문서

(S) Windows
(S) Winlogon

[snip]

특정한 하이브를 검색하는데 제한이 있다면, printkey 는 하이브의 가상 주소를 통해 확인할 수 있다. 예를 들면, HKEY_LOCAL_MACHINE 의 내용을 보고 싶다면 아래 예제의 명령어를 사용하면 된다.

참고 : offset 은 위의 예제인 [CommandReference21#hivelist](#) 에서 가져온 값이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 printkey -o 0xfffff8a000a15010
Volatile Systems Volatility Framework 2.1_alpha
Legend: (S) = Stable (V) = Volatile
```

Registry: User Specified

Key name: CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902} (S)

Last updated: 2009-07-14 07:13:38

Subkeys:

- (S) ATI Technologies
- (S) Classes
- (S) Clients
- (S) Intel
- (S) Microsoft
- (S) ODBC
- (S) Policies
- (S) RegisteredApplications
- (S) Sonic
- (S) Wow6432Node

7.4. hivedump

하이브 안의 모든 서브키들을 재귀적으로 목록화 하려면, hivedump 명령어를 써야 하고 원하는 하이브로 가상 주소를 보내야한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 hivedump -o 0xfffff8a000a15010
Volatile Systems Volatility Framework 2.1_alpha
Last Written      Key
2009-07-14 07:13:38 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}
2009-07-14 04:48:57 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\ATI Technologies
2009-07-14 04:48:57 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\ATI Technologies\Install
2009-07-14 04:48:57 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\ATI
Technologies\Install\South Bridge
2009-07-14 04:48:57 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\ATI
Technologies\Install\South Bridge\ATI_AHCI_RAID
2009-07-14 07:13:39 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\Classes
2009-07-14 04:53:38 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\Classes\*
2009-07-14 04:53:38 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\Classes\*\OpenWithList
2009-07-14 04:53:38 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\Classes\*\OpenWithList\Excel.exe
2009-07-14 04:53:38 \CMI-CreateHive{199DAFC2-6F16-4946-BF90-5A3FC3A60902}\Classes\*\OpenWithList\IExplore.exe
[snip]
```

7.5. hashdump

레지스트리에 저장되어 은닉된 도메인 중요 정보들을 추출하고 복호화하기 위해서는 hashdump 명령어를 사용하면 된다. 더 많은 정보는 BDG 의 Cached Domain Credentials 와 SANS Forensics 2009 - Memory Forensics and Registry Analysis 를 보면 된다.

hashdump 를 사용하기 위해서는 SYSTEM 하이브의 가상 주소를 -y, 그리고 SAM 하이브의 가상 주소를 -s 로 다음과 같이 설정하면 된다.

```
$ python vol.py hashdump -f image.dd -y 0xe1035b60 -s 0xe165cb60
Administrator:500:08f3a52bdd35f179c81667e9d738c5d9:ed88cccbc08d1c18bcded317112555f4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:ddd4c9c883a8ecb2078f88d729ba2e67:e78d693bc40f92a534197dc1d3a6d34f:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:8bfd47482583168a0ae5ab020e1186a9:::
phoenix:1003:07b8418e83fad948aad3b435b51404ee:53905140b80b6d8cbe1ab5953f7c1c51:::
ASPNET:1004:2b5f618079400df84f9346ce3e830467:aef73a8bb65a0f01d9470fad55a411c:::
S-----:1006:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

해쉬는 John the Ripper, rainbow tables 등을 사용하여 크랙할 수 있다.

레지스트리 키가 메모리안에 존재하지 않을 수도 있다. 이럴 경우에는 다음과 같은 오류를 볼 것이다.

```
"ERROR : volatility.plugins.registry.lsadump: Unable to read hashes from registry"
```

만약 맞는 키가 있다면 SYSTEM 에서는 "CurrentControlSet\Control\lsa" 그리고 SAM 에서는 "SAM\Domains\Account" 를 시도해 볼 수 있다. 먼저 "CurrentControlSet"을 얻어야 하며, 이것은 다음과 같이 volshell(hivelist 에서 얻은 오프셋값으로 [SYSTEM REGISTRY ADDRESS] 다음 값을 바꾸면 된다)으로 사용하면 된다.

```
$ ./vol.py -f XPSP3.vmem --profile=WinXPSP3x86 volshell
Volatile Systems Volatility Framework 2.1_alpha
Current context: process System, pid=4, ppid=0 DTB=0x319000
Welcome to volshell! Current memory image is:
file:///XPSP3.vmem
To get help, type 'hh()'
>>> import volatility.win32.hashdump as h
>>> import volatility.win32.hive as hive
>>> addr_space = utils.load_as(self._config)
>>> sysaddr = hive.HiveAddressSpace(addr_space, self._config, [SYSTEM REGISTRY ADDRESS])
>>> print h.find_control_set(sysaddr)
1
>>> ^D
```

그 다음에는 printkey 플러그인을 사용하여 키와 데이터가 있는지 확인할 수 있다. 이전 예제에서 "CurrentControlSet"값이 1 이였기 때문에 첫번째 명령어에서 "ControlSet001"을 사용하면 된다.

```
$ ./vol.py -f XPSP3.vmem --profile=WinXPSP3x86 printkey -K "ControlSet001\Control\lsa"
$ ./vol.py -f XPSP3.vmem --profile=WinXPSP3x86 printkey -K "SAM\Domains\Account"
```

만약 키가 없다면 다음과 같은 오류 메시지를 볼 것이다.

"The requested key could not be found in the hive(s) searched"

7.6. Isadump

레지스트리에서 LSA 비밀번호를 덤프하기 위해서는 isadump 명령어를 사용하면 된다. 이것은 기본 비밀번호(autologin 이 활성화된 시스템), RDP 공공키, 그리고 DPAPI 로 사용되는 중요정보들과 같은 정보들을 노출 시킨다.

더 많은 정보는 BDG 의 Decrypting LSA Secrets 를 보면 된다.

isadump 를 사용하기 위해서는 SYSTEM 하이브의 가상 주소를 -y 패러미터로 설정하고 SECURITY 하이브의 가상 주소는 -s 패러미터로 설정하면 된다.

```
$ python vol.py -f laqma.vmem isadump -y 0xe1035b60 -s 0xe16a6b60
Volatile Systems Volatility Framework 2.0
L$RTMTIMEBOMB_1320153D-8DA3-4e8e-B27B-0D888223A588

0000  00 92 8D 60 01 FF C8 01          ....`....
_SC_Dnscache

L$HYDRAENCKEY_28ada6da-d622-11d1-9cb9-00c04fb16e75

0000  52 53 41 32 48 00 00 00 02 00 00 3F 00 00 00  RSA2H.....?...
0010  01 00 01 00 37 CE 0C C0 EF EC 13 C8 A4 C5 BC B8  ....7.....
0020  AA F5 1A 7C 50 95 A4 E9 3B BA 41 C8 53 D7 CE C6  ...|P...;.A.S...
0030  CB A0 6A 46 7C 70 F3 21 17 1C FB 79 5C C1 83 68  ..jF|p.!...y...h
0040  91 E5 62 5E 2C AC 21 1E 79 07 A9 21 BB F0 74 E8  ..b^,..!..y..!..t.
0050  85 66 F4 C4 00 00 00 00 00 00 00 00 00 00 F9 D7 AD 5C  .f.....
0060  B4 7C FB F6 88 89 9D 2E 91 F2 60 07 10 42 CA 5A  .|......`..B.Z
0070  FC F0 D1 00 0F 86 29 B5 2E 1E 8C E0 00 00 00 00  .....).
0080  AF 43 30 5F 0D 0E 55 04 57 F9 0D 70 4A C8 36 01  .CO_.U.W..pJ.6.
0090  C2 63 45 59 27 62 B5 77 59 84 B7 65 8E DB 8A E0  .cEY'b.wY..e....
00A0  00 00 00 00 89 19 5E D8 CB 0E 03 39 E2 52 04 37  .....^....9.R.7
00B0  20 DC 03 C8 47 B5 2A B3 9C 01 65 15 FF 0F FF 8F  ...G.*...e.....
00C0  17 9F C1 47 00 00 00 00 1B AC BF 62 4E 81 D6 2A  ...G.....bN..*
00D0  32 98 36 3A 11 88 2D 99 3A EA 59 DE 4D 45 2B 9E  .2.6:..-.:Y.ME+.
00E0  74 15 14 E1 F2 B5 B2 80 00 00 00 00 00 75 BD A0 36  t.....u..6
00F0  20 AD 29 0E 88 E0 FD 5B AD 67 CA 88 FC 85 B9 82  .)....[.g.....
0100  94 15 33 1A F1 65 45 D1 CA F9 D8 4C 00 00 00 00  ..3..eE....L....
0110  71 F0 0B 11 F2 F1 AA C5 0C 22 44 06 E1 38 6C ED  q....."D..8l.
0120  6E 38 51 18 E8 44 5F AD C2 CE 0A 0A 1E 8C 68 4F  n8Q..D_.....hO
0130  4D 91 69 07 DE AA 1A EC E6 36 2A 9C 9C B6 49 1F  M.i.....6*...I.
0140  B3 DD 89 18 52 7C F8 96 4F AF 05 29 DF 17 D8 48  ....R|..O..)...H
0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

DPAPI_SYSTEM

0000  01 00 00 00 24 04 D6 B0 DA D1 3C 40 BB EE EC 89  ....$......<@....
0010  B4 BB 90 5B 9A BF 60 7D 3E 96 72 CD 9A F6 F8 BE  ...[. ` }>.r.....
0020  D3 91 5C FA A5 8B E6 B4 81 0D B6 D4          .....
```

7.7. userassist

샘플에서 UserAssist 키들을 얻기 위해서는 userassist 플러그인을 사용하면 된다. 더 많은 정보는 Gleeda 의 Volatility UserAssist Plugin 글을 보면 된다.

```
$ ./vol.py -f win7.vmem --profile=Win7SP0x86 userassist
Volatile Systems Volatility Framework 2.0
-----
Registry: \??\C:\Users\admin\ntuser.dat
Key name: Count
Last updated: 2010-07-06 22:40:25

Subkeys:

Values:
REG_BINARY    Microsoft.Windows.GettingStarted :
Count:        14
Focus Count:   21
Time Focused:  0:07:00.500000
Last updated:  2010-03-09 19:49:20

0000  00 00 00 00 0E 00 00 00 15 00 00 00 A0 68 06 00  .....h..
0010  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0020  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0030  00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C  .....{.
0040  C1 BF CA 01 00 00 00 00  .....

REG_BINARY    UEME_CTLSESSION :
Count:        187
Focus Count:   1205
Time Focused:  6:25:06.216000
Last updated:  1970-01-01 00:00:00

[snip]

REG_BINARY    %windir%\system32\calc.exe :
Count:        12
Focus Count:   17
Time Focused:  0:05:40.500000
Last updated:  2010-03-09 19:49:20

0000  00 00 00 00 0C 00 00 00 11 00 00 00 20 30 05 00  .....0..
0010  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0020  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0030  00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C  .....{.
0040  C1 BF CA 01 00 00 00 00  .....

.....

REG_BINARY    Z:\vmware-share\apps\odbg110\OLLYDBG.EXE :
Count:        11
Focus Count:   266
Time Focused:  1:19:58.045000
Last updated:  2010-03-18 01:56:31

0000  00 00 00 00 0B 00 00 00 0A 01 00 00 69 34 49 00  .....i4I.
0010  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0020  00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF  .....
0030  00 00 80 BF 00 00 80 BF FF FF FF FF 70 3B CB 3A  .....p;.:
0040  3E C6 CA 01 00 00 00 00  .....>.....

[snip]
```

7.8. shimcache

이 플러그인은 Application Compatibility Shim Cache 레지스트리 키를 분석한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 shimcache
Volatile Systems Volatility Framework 2.1_alpha
Last Modified: 2009-07-14 01:39:15 , Path: ???C:\Windows\system32\LogonUI.exe
Last Modified: 2009-07-14 01:39:46 , Path: ???C:\Windows\System32\svchost.exe
Last Modified: 2009-07-14 01:39:50 , Path: ???C:\Windows\system32\vssvc.exe
Last Modified: 2009-06-10 20:39:58 , Path: ???C:\Windows\Microsoft.NET\Framework64\v2.0.50727\mscorsvw.exe
Last Modified: 2009-06-10 21:23:09 , Path: ???C:\Windows\Microsoft.NET\Framework\v2.0.50727\mscorsvw.exe
Last Modified: 2009-06-10 20:39:44 , Path: ???C:\Windows\WinSxS\amd64_netfx-
clrgc_b03f5f7f11d50a3a_6.1.7600.16385_none_ada52b8ba0da82ba\clrgc.exe
Last Modified: 2009-07-14 01:39:25 , Path: ???C:\Windows\System32\netsh.exe
Last Modified: 2009-07-14 01:39:07 , Path: ???C:\Windows\system32\DrvInst.exe
```

8. 충돌, 절전, 전환

8.1. crashinfo

crashdump 헤더의 정보들은 crashinfo 명령어를 통해 출력될 수 있다. 마이크로소프트의 dumpcheck 유틸리티와 같은 정보를 볼 수 있을 것이다.

```
$ python vol.py -f win7_x64.dmp --profile=Win7SP0x64 crashinfo
Volatile Systems Volatility Framework 2.1_alpha
_DMP_HEADER64:
Majorversion:      0x0000000f (15)
Minorversion:      0x00001db0 (7600)
KdSecondaryVersion 0x00000000
DirectoryTableBase 0x32a44000
PfnDataBase         0xfffff80002aa8220
PsLoadedModuleList  0xfffff80002a3de50
PsActiveProcessHead 0xfffff80002a1fb30
MachineImageType    0x00008664
NumberProcessors    0x00000002
BugCheckCode        0x00000000
KdDebuggerDataBlock 0xfffff800029e9070
ProductType         0x00000001
SuiteMask           0x00000110
WriterStatus        0x00000000
Comment             PAGEPAGEPAGEPAGEPAGEPAGE[snip]

Physical Memory Description:
Number of runs: 3
FileOffset  Start Address  Length
00002000    00001000      0009e000
000a0000    00100000      3fde0000
3fe80000    3ff00000      00100000
3ff7f000    3ffff000
```

8.2. hiberno

hibinfo 명령어는 CR0 와 같은 컨트롤 레지스터들의 상태를 포함한 hibernation 파일에 저장되어 있는 추가적인 정보들을 드러낸다. 또한 hibernation 파일이 언제 만들어졌는지에 대한 시간, 상태, 그리고 절전이 된 윈도우의 버전에 대해 식별한다. 이 기능이 출력된 예시는 다음과 같다.

```
$ python vol.py -f hiberfil.sys --profile=Win7SP1x64 hibinfo
IMAGE_HIBER_HEADER:
Signature: HIBR
SystemTime: 2011-12-23 16:34:27

Control registers flags
CR0: 80050031
CR0[PAGING]: 1
CR3: 00187000
CR4: 000006f8
CR4[PSE]: 1
CR4[PAE]: 1

Windows Version is 6.1 (7601)
```


8.3. imagecopy

imagecopy 명령어는 어떠한 주소공간 종류라도(crashdump, hibernation 파일, 또는 라이브 firewire 세션과 같은) raw 메모리 이미지로 변환할 수 있도록 한다. 이 변환은 사용하는 다른 포렌식 툴들이 raw 메모리 덤프들만 읽는것을 지원할때 필요하다.

이 명령어를 위해서는 프로파일이 정확히 명시 되어야 한다. 만약 미리 모른다면, CommandReference21#imageinfo 또는 CommandReference21#kdbgscan 명령어를 먼저 사용한다. 출력파일은 -O 플래그로 명시되어야 한다. 진행은 변환된 파일로 업데이트 된다.

```
$ python vol.py -f win7_x64.dmp --profile=Win7SP0x64 imagecopy -O copy.raw
Volatile Systems Volatility Framework 2.1_alpha
Writing data (5.00 MB chunks): |.....|
```

8.4. raw2dmp

raw 메모리 덤프(win32dd 로 획득 또는 VMware 의 .vmem 파일의 샘플)를 마이크로소프트 충돌 덤프로 변환하려면, raw2dmp 명령어를 사용해야 한다. 분석할때 WinDbg 커널 디버거에서 메모리를 불러오길 원한다면 유용하다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 raw2dmp -O copy.dmp
Volatile Systems Volatility Framework 2.1_alpha
Writing data (5.00 MB chunks): |.....|
```

9. 악성 코드, 루트킷

9.1. malfind

malfind 명령어는 VAD 태그와 페이지 권한들 같은 문자들을 기반으로 사용자 모드(user mode) 메모리에 숨겨져 있거나 삽입(injected) 되어 있는 코드/DLLs 를 찾아내는데 도움을 준다.

참고 : malfind 는 CreateRemoteThread->LoadLibrary 로 사용되는 프로세스에 삽입되는 DLLs 은 탐지하지 않는다. 이 기술로 삽입된 DLLs 는 숨겨지지 않으며 [CommandReference21#dlllist](#) 에서 이것들을 확인할 수 있다. malfind 의 목적은 기본적인 메소드들/도구들이 보지 못하는 것을 DLLs 의 위치를 찾아내는 것이다.

아래 Zeus 의 존재를 탐지하는 예제를 보여준다. 첫번째 메모리 세그먼트(0x01600000 부터 시작)가 private 로 표시되고 (다른 프로세스와 공유하지 않는), 미리 공간에 할당되는 매핑된 파일이 없다는 없다는 의미의 VadS 태그를 가지고 있는 실행 부분에서 탐지된다.

이 주소에서 찾은 데이터의 어셈블리를 기준으로, 어떤 API Hook trampoline stubs 가 포함되어 있는 것으로 보인다.

PEB's 모듈 리스트들에 포함되지 않는 곳에 실행 부분이 포함되어 있기 때문에 두번째 메모리 세그먼트 (0x015D000 으로 시작)가 탐지 되었다.

만약 malfind 에 의해서 인식되는 메모리 세그먼트의 압축된 복사파일을 저장하고 싶다면, -D 혹은 -dump-dr=DIR 과 옵션과 함께 결과가 저장되는 디렉터리를 지정해주면 됩니다. 이 경우에는 explorer.exe 파일에 삽입된 Zeus 바이너리의 언팩된 복사본은 디스크에 저장될 것이다.

```
$ python vol.py -f zeus.vmem malfind -p 1724
Volatile Systems Volatility Framework 2.1_alpha

Process: explorer.exe Pid: 1724 Address: 0x1600000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01600000 b8 35 00 00 00 e9 cd d7 30 7b b8 91 00 00 00 e9 .5.....0{.....
0x01600010 4f df 30 7b 8b ff 55 8b ec e9 ef 17 c1 75 8b ff O.0{.U.....u..
0x01600020 55 8b ec e9 95 76 bc 75 8b ff 55 8b ec e9 be 53 U...v.u.U...S
0x01600030 bd 75 8b ff 55 8b ec e9 d6 18 c1 75 8b ff 55 8b ..u..U.....u..U.

0x1600000 b835000000 MOV EAX, 0x35
0x1600005 e9cdd7307b JMP 0x7c90d7d7
0x160000a b891000000 MOV EAX, 0x91
0x160000f e94fdf307b JMP 0x7c90df63
```

Volatility Command 2.1 번역 문서

```
0x1600014 8bff      MOV EDI, EDI
0x1600016 55        PUSH EBP

Process: explorer.exe Pid: 1724 Address: 0x15d0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 38, MemCommit: 1, PrivateMemory: 1, Protection: 6
```

```
0x015d0000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00 MZ.....
0x015d0010 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
0x015d0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x015d0030 00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00 .....
```

```
0x15d0000 4d      DEC EBP
0x15d0001 5a      POP EDX
0x15d0002 90      NOP
0x15d0003 0003     ADD [EBX], AL
0x15d0005 0000     ADD [EAX], AL
0x15d0007 000400    ADD [EAX+EAX], AL
0x15d000a 0000     ADD [EAX], AL
```

9.2. yarascan

Volatility 는 물리적 혹은 가상적 메모리 공간들에서 poll 태그들 같은 간단한 패턴을 찾는 것을 도움주기 위해서 내장되어 있는 검색 엔진들을 가지고 있다. 그러나, 만약 정규식(regular expressions) 이나 복잡한 패턴(compound rules) 같이 더욱 복잡한 것을 검색할 필요가 있다면, yarascan 플러그인을 사용할 수 있다. 이 플러그인은 사용자 모드와 커널 모드에 포함되어 있는 바이트들의 순서의 위치(와일드 카드들의 어셈블리 명령어와 같은), 정규 표현식들, ANSI 문자들, Unicode 문자들을 검색하는데 도움을 준다.

YARA 패턴(Rules)을 만들 수 있고, --yara-file=RULESFILE 옵션을 이용하여 명시할 수 있다. 혹은 어떤 것을 간단하게 보거나, 몇 번만 검색할 계획이라면, --yara-file=RLUESTEXT 같이 기준을 명시할 수 있다.

프로세스에서 rules.yar 파일내에 정의되어 있는 문자들을 검색하기 위해서 아래와 같은 명령어를 입력하고, 간단하게 결과를 확인할 수 있다.

```
$ python vol.py -f zeus.vmem yarascan --yara-file=/path/to/rules.yar
```

프로세스에서 간단한 문자를 검색하고, 매칭되는 문자가 포함되는 메모리 세그먼트들을 덤프하기 위해서는 아래 명령어를 입력하면 된다.

```
$ python vol.py -f zeus.vmem yarascan -D dump_files --yara-rules="simpleStringToFind"
```

Volatility Command 2.1 번역 문서

커널 메모리에서 바이트 패턴을 검색하기 위해서, 아래 기술을 사용한다. 이것은 모든 세션들에서, 1MB 단위의 덩어리(chunks) 안에 있는 메모리를 통해 검색한다. TDL3 악성코드는 디스크의 SCSI 어댑터들의 하드 패치에 적용한다(때때로 atapi.sys 나 vm SCSI.sys). 특히, 이것은 파일의 .rsrc 세션에 셸코드를 추가한다그리고 AddressOfEntryPoint 를 수정합니다. 그래서 이것은 셸코드를 가르키게 된다. 이것은 TDL3 의 주요 지속성 메소드 이다. 셸코드의 독특한 명령어들의 하나는 cmp dword ptr [eax] '3LDT' 이다. 그래서 이것들의 opcodes 로부터 yara 패턴을 만들게 된다.

```
$ python vol.py -f tdl3.vmem yarascan --yara-rules="{8B 00 81 38 54 44 4C 33 75 5A}" -K
Volatile Systems Volatility Framework 2.1_alpha
Rule: r1
Owner: (Unknown Kernel Memory)
0x8138dcc0 8b 00 81 38 54 44 4c 33 75 5a 8b 45 f4 05 fd 29 ...8TDL3uZ.E...)
0x8138dcd0 b7 f0 50 b8 08 03 00 00 8b 80 00 00 df ff ff b0 ..P.....
0x8138dce0 00 01 00 00 b8 08 03 00 00 8b 80 00 00 df ff 8b .....
0x8138dcf0 40 04 8b 4d ec 03 41 20 ff d0 ff 75 e0 b8 08 03 @..M..A....u....
Rule: r1
Owner: dump_vm SCSI.sys
0xf94bb4c3 8b 00 81 38 54 44 4c 33 75 5a 8b 45 f4 05 fd 29 ...8TDL3uZ.E...)
0xf94bb4d3 b7 f0 50 b8 08 03 00 00 8b 80 00 00 df ff ff b0 ..P.....
0xf94bb4e3 00 01 00 00 b8 08 03 00 00 8b 80 00 00 df ff 8b .....
0xf94bb4f3 40 04 8b 4d ec 03 41 20 ff d0 ff 75 e0 b8 08 03 @..M..A....u....
Rule: r1
Owner: vm SCSI.sys
0xf9dba4c3 8b 00 81 38 54 44 4c 33 75 5a 8b 45 f4 05 fd 29 ...8TDL3uZ.E...)
0xf9dba4d3 b7 f0 50 b8 08 03 00 00 8b 80 00 00 df ff ff b0 ..P.....
0xf9dba4e3 00 01 00 00 b8 08 03 00 00 8b 80 00 00 df ff 8b .....
0xf9dba4f3 40 04 8b 4d ec 03 41 20 ff d0 ff 75 e0 b8 08 03 @..M..A....u....
```

각 프로세스안에 있는 byte 패턴을 얻기 위한 검색은 아래 예제와 같다.

```
$ python vol.py -f zeus.vmem yarascan --yara-rules="{eb 90 ff e4 88 32 0d}" --pid=624
```

각 프로세스안에 있는 정규 표현식에 대한 검색은 아래 예제와 같다.

```
$ python vol.py -f zeus.vmem yarascan --yara-rules="/my(regular|expression{0,2})/" --pid=624
```

9.3. svcscan

Volatility 는 동작하고 있는 머신에서 Windows API 사용 없이 서비스들을 검색할 수 있는 능력을 가지고 있는 유일한 메모리 포렌직 프레임워크(Memory Forensics Framework) 이다.

어떤 서비스들이 메모리 이미지에 등록되어 있는지를 확인하기 위해서, svcscan 명령어를 사용하면 된다. 그 결과는 각 서비스의 프로세스 ID (만약 사용자 모드 프로세스에서 동작하고 지속되고 있다면), 서비스 이름, 서비스 출력 이름, 서비스 타입, 현재 상태가 나타난다. 또한, 등록되어 있는 서비스의 실행파일(binary)의 경로도 보여진다. - 사용자 모드 서비스들, 커널 모드에서 동작하고 있는 서비스들을 위한 드라이버 이름들의 EXE 파일일 것이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 svcscan
Volatile Systems Volatility Framework 2.1_alpha
Offset: 0xa26e70
Order: 71
Process ID: 1104
Service Name: DPS
```

Volatility Command 2.1 번역 문서

```
Display Name: Diagnostic Policy Service
Service Type: SERVICE_WIN32_SHARE_PROCESS
Service State: SERVICE_RUNNING
Binary Path: C:\Windows\system32\svchost.exe -k LocalServiceNoNetwork
```

```
Offset: 0xa25620
Order: 70
Process ID: -
Service Name: dot3svc
Display Name: Wired AutoConfig
Service Type: SERVICE_WIN32_SHARE_PROCESS
Service State: SERVICE_STOPPED
Binary Path: -
```

```
Offset: 0xa25440
Order: 68
Process ID: -
Service Name: Disk
Display Name: Disk Driver
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\Disk
```

[snip]

9.4. Ldrmodules

DLL 파일을 숨기는 방법에는 여러가지가 있다. PEB(Process Environment Block) 에 링크드된 목록들의 하나(아니면 모두)에서 DLL 의 연결을 해체하는 것도 하나의 방법이다. 그러나 이렇게 할 시에는, DLL, 디스크의 모든 경로의 정보의 기본 주소(Base Address)가 정의되어 있는 VAD(Virtual Address Descriptor - 가상 메모리 디스트리뷰터) 가 포함된 정보가 여전히 남아 있다. 3PEB 목록들과 같이 이 정보를(메모리 매핑된 파일이라고도 알려짐) 상호 참조하기 위해서, Ldrmodules 명령어를 사용한다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 Ldrmodules
```

Volatility Systems Volatility Framework 2.1_alpha

Pid	Process	Base	InLoad	InInit	InMem	MappedPath
208	smss.exe	0x0000000047a90000	True	False	True	\Windows\System32\smss.exe
296	csrss.exe	0x0000000049700000	True	False	True	\Windows\System32\csrss.exe
344	csrss.exe	0x0000000000390000	False	False	False	\Windows\Fonts\vgasys.fon
344	csrss.exe	0x00000000007a0000	False	False	False	\Windows\Fonts\vgaoem.fon
344	csrss.exe	0x00000000020e0000	False	False	False	\Windows\Fonts\ega40woa.fon
344	csrss.exe	0x0000000000a60000	False	False	False	\Windows\Fonts\dosapp.fon
344	csrss.exe	0x0000000000a70000	False	False	False	\Windows\Fonts\cga40woa.fon
344	csrss.exe	0x00000000020d0000	False	False	False	\Windows\Fonts\cga80woa.fon
428	services.exe	0x0000000000020000	False	False	False	\Windows\System32\en-
US\services.exe.mui						
428	services.exe	0x00000000ff670000	True	False	True	\Windows\System32\services.exe
444	lsass.exe	0x0000000000180000	False	False	False	\Windows\System32\en-
US\crypt32.dll.mui						
444	lsass.exe	0x0000000076b20000	True	True	True	\Windows\System32\kernel32.dll
444	lsass.exe	0x0000000076c40000	True	True	True	\Windows\System32\user32.dll
444	lsass.exe	0x0000000074a70000	True	True	True	\Windows\System32\msprv.dll
444	lsass.exe	0x0000000076d40000	True	True	True	\Windows\System32\ntdll.dll
568	svchost.exe	0x00000000001e0000	False	False	False	\Windows\System32\en-
US\umpnpgm.dll.mui						

[snip]

Volatility Command 2.1 번역 문서

이것들이 포함된 PEB, DLL 목록들이 사용자 모드에서 모두 존재한 뒤로, 경로는 간단하게 덮어쓰기 함으로써 DLL 을 숨기는 것(이해하기 힘들게)으로 악성코드에 이용되어 왔다. 단지 링크가 해체된(unlinked)한 엔트리들을 찾아내기 위한 도구들은 악성코드가 C:\bad.dll 을 C:\windows\system32\kernel32.dll 를 덮어쓰기 할 수 있다는 사실을 놓치게 된다. 그래서 모든 엔트리들의 전체 경로를 보기 위해서 ldrmodules 를 사용할 시 -v 혹은 -verbose 옵션을 통해 확인할 수 있다.

더욱더 구체적인 예제를 보려면, [ZeroAccess Misleads Memory-File Link](#) 와 [QuickPost: Flame & Volatility](#) 를 참조하라.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 ldrmodules -v
Volatile Systems Volatility Framework 2.1_alpha
Pid      Process      Base      InLoad InInit InMem MappedPath
-----
208 smss.exe   0x0000000047a90000 True  False True  \Windows\System32\smss.exe
Load Path: \SystemRoot\System32\smss.exe : smss.exe
Mem Path:  \SystemRoot\System32\smss.exe : smss.exe
296 csrss.exe 0x0000000049700000 True  False True  \Windows\System32\csrss.exe
Load Path: C:\Windows\system32\csrss.exe : csrss.exe
Mem Path:  C:\Windows\system32\csrss.exe : csrss.exe
344 csrss.exe 0x0000000000390000 False False False \Windows\Fonts\vgasys.fon
344 csrss.exe 0x00000000007a0000 False False False \Windows\Fonts\vgaoem.fon
344 csrss.exe 0x000000000020e000 False False False \Windows\Fonts\ega40woa.fon
344 csrss.exe 0x00000000000a6000 False False False \Windows\Fonts\dosapp.fon
344 csrss.exe 0x00000000000a7000 False False False \Windows\Fonts\cga40woa.fon
344 csrss.exe 0x000000000020d000 False False False \Windows\Fonts\cga80woa.fon
428 services.exe 0x0000000000020000 False False False \Windows\System32\en-
US\services.exe.mui
428 services.exe 0x00000000ff670000 True  False True  \Windows\System32\services.exe
Load Path: C:\Windows\system32\services.exe : services.exe
Mem Path:  C:\Windows\system32\services.exe : services.exe
444 lsass.exe 0x0000000000018000 False False False \Windows\System32\en-
US\crypt32.dll.mui
444 lsass.exe 0x00000000076b2000 True  True  True  \Windows\System32\kernel32.dll
Load Path: C:\Windows\system32\kernel32.dll : kernel32.dll
Init Path: C:\Windows\system32\kernel32.dll : kernel32.dll
Mem Path:  C:\Windows\system32\kernel32.dll : kernel32.dll
[snip]
```

9.5. impscan

메모리 덤프에서 찾은 코드를 완벽하게 역공학(Reverse engineer)를 하기 위해서는 코드가 임포트 하고 있는 함수(functions)을 보는 것이 필수적이다. 다른 말로 말하면, 그것이 가르키고 있는 API 함수들이다.

[CommandReference21#dlldump](#), [CommandReference21#moddump](#), [CommandReference21#procexedump](#) 을 이용하여 실행파일들을 덤프 할 때, IAT(Import Address Table)은 제대로 복원되지 않을 것이다. the IAT (Import Address Table) may not properly be reconstructed due to the high likelihood that one or more pages in the PE header or IAT are not memory resident (paged).

Volatility Command 2.1 번역 문서

그래서 우리는 impscan 을 개발하였다. impscan 은 PE 의 IAT 을 파싱할 필요 없이 API 들을 불러와 정의할 수 있다. 만약 악성코드가 완벽하게 PE Header 를 지우고, 커널 드라이브에서 실행되고 있다면, 이것은 작동되지 않을 것이다.

impscan 의 이전 버전들에서는 IDA Pro 와 같이 사용해서 정의된 IDB 를 생성할 수 있었다. 이 기능은 일시적으로 사용이 중단되었다. 그러나 다른 비슷한 기능이 소개 된다면, 후에는 다시 도입을 할 것이다.

```
$ python vol.py -f coreflood.vmem -p 2044 malfind
Volatile Systems Volatility Framework 2.1_alpha

Process: IEXPLORE.EXE Pid: 2044 Address: 0x7ff80000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6

0x7ff80000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x7ff80010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x7ff80020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x7ff80030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

0x7ff80000 0000          ADD [EAX], AL
0x7ff80002 0000          ADD [EAX], AL
0x7ff80004 0000          ADD [EAX], AL
0x7ff80006 0000          ADD [EAX], AL
```

이제 Coreflood 의 압축을 해제하고 임포트(import)되는 API 들을 보기 원한다고 합시다. malfind 플러그인에서 찾아낸 베이스 주소(base address) 구체적으로 확인하기 위해서 impscan 을 사용한다. 우리는 실제 ImageBase 에서 누락된 페이지를 파악하기 위해서 0x10000 로 베이스 주소를 수정한다.

```
$ python vol.py -f coreflood.vmem -p 2044 impscan -b 0x7ff81000
Volatile Systems Volatility Framework 2.1_alpha
IAT      Call      Module      Function
-----
0x7ff9e000 0x77dd77b3 ADVAPI32.dll SetSecurityDescriptorDacl
0x7ff9e004 0x77dfd4c9 ADVAPI32.dll GetUserNameA
0x7ff9e008 0x77dd6bf0 ADVAPI32.dll RegCloseKey
0x7ff9e00c 0x77ddeaf4 ADVAPI32.dll RegCreateKeyExA
0x7ff9e010 0x77dfc123 ADVAPI32.dll RegDeleteKeyA
0x7ff9e014 0x77ddede5 ADVAPI32.dll RegDeleteValueA
0x7ff9e018 0x77ddd966 ADVAPI32.dll RegNotifyChangeKeyValue
0x7ff9e01c 0x77dd761b ADVAPI32.dll RegOpenKeyExA
0x7ff9e020 0x77dd7883 ADVAPI32.dll RegQueryValueExA
0x7ff9e024 0x77ddebe7 ADVAPI32.dll RegSetValueExA
0x7ff9e028 0x77dfc534 ADVAPI32.dll AdjustTokenPrivileges
0x7ff9e02c 0x77e34c3f ADVAPI32.dll InitiateSystemShutdownA
0x7ff9e030 0x77dfd11b ADVAPI32.dll LookupPrivilegeValueA
0x7ff9e034 0x77dd7753 ADVAPI32.dll OpenProcessToken
0x7ff9e038 0x77dfc8c1 ADVAPI32.dll RegEnumKeyExA
[snip]
```

만약에 -b 나 -base 옵션을 통해서 베이스 주소를 찾아내지 못한다면, 임포트된 함수들을 위해 프로세스의 주요 모듈(예로 -p 2044 로부터 IEEXPLORE.EXE)들을 끝까지 검색할 것이다. 또한, 임포트된 커널 모드 함수들의 드라이버를 스캔하기 위해서 커널 드라이브의 베이스 주소를 찾아낼 수 있다.

Volatility Command 2.1 번역 문서

Laqma 는 lanmandrv.sys 이름으로 커널 드라이버를 저장하고 있다. 만약 [CommandReference21#moddump](#) 으로 압축을 해제하면, IAT 는 에러(corrupt)가 발생할 것이다. 그래서 그것을 다시 수정(rebuild)하기 위해서 impscan 을 사용한다.

```
$ python vol.py -f laqma.vmem impscan -b 0xfca29000
Volatile Systems Volatility Framework 2.1_alpha
IAT      Call      Module      Function
-----
0xfca2a080 0x804ede90 ntoskrnl.exe IofCompleteRequest
0xfca2a084 0x804f058c ntoskrnl.exe IoDeleteDevice
0xfca2a088 0x80568140 ntoskrnl.exe IoDeleteSymbolicLink
0xfca2a08c 0x80567dcc ntoskrnl.exe IoCreateSymbolicLink
0xfca2a090 0x805a2130 ntoskrnl.exe MmGetSystemRoutineAddress
0xfca2a094 0x805699e0 ntoskrnl.exe IoCreateDevice
0xfca2a098 0x80544080 ntoskrnl.exe ExAllocatePoolWithTag
0xfca2a09c 0x80536dc3 ntoskrnl.exe wcscmp
0xfca2a0a0 0x804fdb0c ntoskrnl.exe ZwOpenKey
0xfca2a0a4 0x80535010 ntoskrnl.exe _except_handler3
0xfca2a3ac 0x8056df44 ntoskrnl.exe NtQueryDirectoryFile
0xfca2a3b4 0x8060633e ntoskrnl.exe NtQuerySystemInformation
0xfca2a3bc 0x805bfb78 ntoskrnl.exe NtOpenProcess
```

다음 예제는 x64 드라이버에서 impscan 을 사용하는 것과 render_idc 결과 포맷형식을 출력방법을 이용하는 것을 보여준다. 함수를 호출하는데 이름(label)을 정의하기 위해서 IDA Pro 의 임포트 할 수 있는 것을 IDC 파일로 제공을 해준다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 impscan -b 0xfffff88003980000 --
output=idc --output-file=imps.idc
Volatile Systems Volatility Framework 2.1_alpha

$ cat imps.idc
#include <idc.idc>
static main(void) {
    MakeDword(0xfffff8800398A000);
    MakeName(0xfffff8800398A000, "KeSetEvent");
    MakeDword(0xfffff8800398A008);
    MakeName(0xfffff8800398A008, "PsTerminateSystemThread");
    MakeDword(0xfffff8800398A010);
    MakeName(0xfffff8800398A010, "KeInitializeEvent");
    MakeDword(0xfffff8800398A018);
    MakeName(0xfffff8800398A018, "PsCreateSystemThread");
    MakeDword(0xfffff8800398A020);
    MakeName(0xfffff8800398A020, "KeWaitForSingleObject");
    MakeDword(0xfffff8800398A028);
    MakeName(0xfffff8800398A028, "ZwClose");
    MakeDword(0xfffff8800398A030);
    MakeName(0xfffff8800398A030, "RtlInitUnicodeString");
    [snip]
    MakeDword(0xfffff8800398A220);
    MakeName(0xfffff8800398A220, "RtlAnsiCharToUnicodeChar");
    MakeDword(0xfffff8800398A228);
    MakeName(0xfffff8800398A228, "__C_specific_handler");
    Exit(0);}
```


9.6. apihooks

사용자 모드나 커널모드에서 API 가 hook 하는 것을 찾기 위해서는, apihook 플러그인을 사용한다. 이것은 IAT, EAT, 인라인 형태의 hooks, 그리고 특별한 형태의 hooks 를 찾아낸다. 인라인 hooks 에서는, 직접, 간접 위치들에서 CALLs 그리고 JMPs 를 탐지하고 명령 순서들(instruction sequences)에서는 PUST/RET 를 탐지한다. 이것을 탐지하는 특별한 형태의 hooks 는 ntdll.dll 에서 syscall 후킹을 포함하고, 커널 모드에서 알려지지 않은 코드 페이지들을 호출한다.

Volatility 2.1 기준에서 apihooks 는 또한 흑되어 있는 winsock 프로시저 테이블들을 탐지한다. 읽기 쉬운 형태의 출력 포맷을 포함하고, multiple hop 디스어셈블리를 지원한다. 그리고, 중요하지 않은 프로세스들과 DLL 들을 무시한 메모리를 통해 더욱더 빠르게 선택적으로 검색을 할 수 있다.

여기 Coreflood 로 인해서 설치된 IAT hooks 을 탐지하는 예제가 있다. 루트킷 코드가 존재하고 있는 메모리에 위치하고 있는 모듈(DLL) 이 없기 때문에 후킹된 모듈은 알려지지 않는다. 만약 그 hooks 가 포함된 코드 압축해체를 원한다면 몇개를 선택할 수 있다.

1. [CommandReference21#malfind](#) 으로 자동으로 검색하고 그것을 해체할 수 있는지 보라.
2. [CommandReference21#volshell](#) dd/db 명령어를 backwards 를 검색하기 위해 명령하고 MZ Header 를 보라. 그리고 -base 옵션을 통해 [CommandReference#dlldump](#) 위치 함으로써 통과해라.
3. 각각의 파일(처음과 마지막의 주소에 따라 표시된)들의 코드 세그먼트들을 압축해체하기 위해서 [CommandReference21#vaddump](#) 을 사용하고, 0x7ff82 범위들 안에 포함되어 있는 파일을 찾는다.

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

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

*****
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 2044 (IEXPLORE.EXE)
Victim module: iexplore.exe (0x400000 - 0x419000)
Function: kernel32.dll!LoadLibraryA at 0x7ff82a50
Hook address: 0x7ff82a50
Hooking module: <unknown>
```

Volatility Command 2.1 번역 문서

```
Disassembly(0):
0x7ff82a50 51      PUSH ECX
0x7ff82a51 e80aefffff  CALL 0x7ff81960
0x7ff82a56 84c0      TEST AL, AL
0x7ff82a58 7414      JZ 0x7ff82a6e
0x7ff82a5a 8b442408   MOV EAX, [ESP+0x8]
0x7ff82a5e 8b0d0054fa7f  MOV ECX, [0x7ffa5400]
0x7ff82a64 8b512c     MOV EDX, [ECX+0x2c]
0x7ff82a67 50        PUSH EAX
```

[snip]

여기 Slintbacker 에 의해서 설치된 인라인 hooks 를 탐지하는 예제가 있다. 2.1 에서 새로 추가된 multiple hop 어셈블리라는 것을 주목하자. 이것은 0xe50000 에 점프를 해서 0x7c81caa2 에 hook 의 첫번째 hop 으로 보인다. 그리고 트램폴린(trampoline)의 나머지에 실행되고 있는 0xe50000 의 코드의 어셈블리를 확인할 수 있다.

```
$ python vol.py -f silentbanker.vmem -p 1884 apihooks
Volatile Systems Volatility Framework 2.1_alpha
*****
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1884 (IEXPLORE.EXE)
Victim module: kernel32.dll (0x7c800000 - 0x7c8f4000)
Function: kernel32.dll!ExitProcess at 0x7c81caa2
Hook address: 0xe50000
Hooking module: <unknown>

Disassembly(0):
0x7c81caa2 e959356384   JMP 0xe50000
0x7c81caa7 6aff        PUSH -0x1
0x7c81caa9 68b0f3e877   PUSH DWORD 0x77e8f3b0
0x7c81caae ff7508      PUSH DWORD [EBP+0x8]
0x7c81cab1 e846fffff    CALL 0x7c81c9fc

Disassembly(1):
0xe50000 58          POP EAX
0xe50001 680500e600   PUSH DWORD 0xe60005
0xe50006 6800000000   PUSH DWORD 0x0
0xe5000b 680000807c   PUSH DWORD 0x7c800000
0xe50010 6828180310   PUSH DWORD 0x10031828
0xe50015 50          PUSH EAX

[snip]
```

여기 Laqma 에 의해서 설치된 PUSH/RET 인라인 hooks 를 탐지하고 있는 예제가 있다.

```
$ python vol.py -f laqma.vmem -p 1624 apihooks
Volatile Systems Volatility Framework 2.1_alpha
*****
Hook mode: Usermode
Hook type: Inline/Trampoline
Process: 1624 (explorer.exe)
Victim module: USER32.dll (0x7e410000 - 0x7e4a0000)
Function: USER32.dll!MessageBoxA at 0x7e45058a
Hook address: 0xac10aa
Hooking module: Dll.dll

Disassembly(0):
0x7e45058a 68aa10ac00   PUSH DWORD 0xac10aa
0x7e45058f c3          RET
0x7e450590 3dbc04477e   CMP EAX, 0x7e4704bc
0x7e450595 00742464     ADD [ESP+0x64], DH
0x7e450599 a118000000   MOV EAX, [0x18]
```

Volatility Command 2.1 번역 문서

```
0x7e45059e 6a00    PUSH 0x0
0x7e4505a0 ff      DB 0xff
0x7e4505a1 70      DB 0x70
```

```
Disassembly(1):
0xac10aa 53      PUSH EBX
0xac10ab 56      PUSH ESI
0xac10ac 57      PUSH EDI
0xac10ad 90      NOP
0xac10ae 90      NOP
```

[snip]

여기 ntdll.dll (Carberp 샘플을 사용함) 에 syscall 패치들을 탐지하기 위해 apihooks 을 사용하는 예제가 있다.

```
$ python vol.py -f carberp.vmem -p 1004 apihooks
Volatile Systems Volatility Framework 2.1_alpha
*****
Hook mode: Usermode
Hook type: NT Syscall
Process: 1004 (explorer.exe)
Victim module: ntdll.dll (0x7c900000 - 0x7c9af000)
Function: NtQueryDirectoryFile
Hook address: 0x1da658f
Hooking module: <unknown>

Disassembly(0):
0x7c90d750 b891000000    MOV EAX, 0x91
0x7c90d755 ba84ddda01    MOV EDX, 0x1dadd84
0x7c90d75a ff12      CALL DWORD [EDX]
0x7c90d75c c22c00      RET 0x2c
0x7c90d75f 90      NOP
0x7c90d760 b892000000    MOV EAX, 0x92
0x7c90d765 ba      DB 0xba
0x7c90d766 0003      ADD [EBX], AL

Disassembly(1):
0x1da658f 58      POP EAX
0x1da6590 8d056663da01    LEA EAX, [0x1da6366]
0x1da6596 ffe0      JMP EAX
0x1da6598 c3      RET
0x1da6599 55      PUSH EBP
0x1da659a 8bec      MOV EBP, ESP
0x1da659c 51      PUSH ECX
0x1da659d 8365fc00      AND DWORD [EBP+0xfffffc], 0x0
0x1da65a1 688f88d69b      PUSH DWORD 0x9bd6888f

[snip]
```

여기 커널 모드 함수의 인라인 hook 을 탐지하기 위해 apihooks 를 사용하는 예제가 있다.

```
$ python vol.py apihooks -f rustock.vmem
*****
Hook mode: Kernelmode
Hook type: Inline/Trampoline
Victim module: ntoskrnl.exe (0x804d7000 - 0x806cf980)
Function: ntoskrnl.exe!IoofCallDriver at 0x804ee130
Hook address: 0xb17a189d
Hooking module: <unknown>

Disassembly(0):
0x804ee130 ff2580c25480    JMP DWORD [0x8054c280]
0x804ee136 cc      INT 3
0x804ee137 cc      INT 3
0x804ee138 cc      INT 3
0x804ee139 cc      INT 3
```

Volatility Command 2.1 번역 문서

```
0x804ee13a cc      INT 3
0x804ee13b cc      INT 3
0x804ee13c 8bff    MOV EDI, EDI
0x804ee13e 55      PUSH EBP
0x804ee13f 8bec    MOV EBP, ESP
0x804ee141 8b4d08   MOV ECX, [EBP+0x8]
0x804ee144 83f929   CMP ECX, 0x29
0x804ee147 72      DB 0x72

Disassembly(1):
0xb17a189d 56      PUSH ESI
0xb17a189e 57      PUSH EDI
0xb17a189f 8bf9    MOV EDI, ECX
0xb17a18a1 8b7708   MOV ESI, [EDI+0x8]
0xb17a18a4 3b35ab6d7ab1   CMP ESI, [0xb17a6dab]
0xb17a18aa 7509     JNZ 0xb17a18b5
0xb17a18ac 52      PUSH EDX
0xb17a18ad 57      PUSH EDI
0xb17a18ae e8c6430000   CALL 0xb17a5c79
0xb17a18b3 eb6a     JMP 0xb17a191f
```

여기 커널 드라이버에서 알려지지 않은 코드 페이지에 calls 를 탐지하기 위해 apihooks 를 사용하는 예제가 있다.

```
$ python vol.py -f rustock-c.vmem apihooks
Volatile Systems Volatility Framework 2.1_alpha
*****
Hook mode: Kernelmode
Hook type: Unknown Code Page Call
Victim module: tcpip.sys (0xf7bac000 - 0xf7c04000)
Function: <unknown>
Hook address: 0x81ecd0c0
Hooking module: <unknown>

Disassembly(0):
0xf7be2514 ff15bcd0ec81   CALL DWORD [0x81ecd0bc]
0xf7be251a 817dfc03010000   CMP DWORD [EBP+0xffffffff], 0x103
0xf7be2521 7506           JNZ 0xf7be2529
0xf7be2523 57           PUSH EDI
0xf7be2524 e8de860000     CALL 0xf7beac07
0xf7be2529 83           DB 0x83
0xf7be252a 66           DB 0x66
0xf7be252b 10           DB 0x10

Disassembly(1):
0x81ecd0c0 0e           PUSH CS
0x81ecd0c1 90           NOP
0x81ecd0c2 83ec04       SUB ESP, 0x4
0x81ecd0c5 c704246119c481   MOV DWORD [ESP], 0x81c41961
0x81ecd0cc cb           RETF

[snip]
```

9.7. idt

시스템의 IDT(Interrupt Descripro Table)을 출력하기 위해서, idt 명령어를 사용한다. 만약 시스템에 멀티 프로세서들이 존재한다면, 각각 독립적인 CPU 의 IDT 가 나타난다. 당신은 CPU 갯수, GDT selector, 현재 주소 및 포함된 모듈을 볼 수 있다. 그리고 IDT 함수가 위치하고 있는 PE section 의 이름을 확인할 수 있다.

Volatility Command 2.1 번역 문서

어떤 루트킷들은 KtSystemService 를 위한 IDT 엔트리를 hook 한다. 그러나 NT 모듈 내부 루틴(KiSystemService 가 가르켜야 할 곳)을 가르킨다. 그러나, 그 주소에 인라인 hook 이 있다. 아래 출력은 어떻게 Volatility 가 결과를 내놓는지 보여준다. 어떻게 KiSystemService 를 위한 0x2E 엔트리가 모든 다른 것처럼 .text 대신에 ntoskrnl.exe 의 .rsrc 세션안에 있는지 주목해보자.

```
$ python vol.py -f rustock.vmem idt
Volatile Systems Volatility Framework 2.1_alpha
CPU Index Selector Value Module Section
-----
0 0 8 0x8053e1cc ntoskrnl.exe .text
0 1 8 0x8053e344 ntoskrnl.exe .text
0 2 88 0x00000000 ntoskrnl.exe
0 3 8 0x8053e714 ntoskrnl.exe .text
0 4 8 0x8053e894 ntoskrnl.exe .text
0 5 8 0x8053e9f0 ntoskrnl.exe .text
0 6 8 0x8053eb64 ntoskrnl.exe .text
0 7 8 0x8053f1cc ntoskrnl.exe .text
0 8 80 0x00000000 ntoskrnl.exe
[snip]
0 2B 8 0x8053db10 ntoskrnl.exe .text
0 2C 8 0x8053dcb0 ntoskrnl.exe .text
0 2D 8 0x8053e5f0 ntoskrnl.exe .text
0 2E 8 0x806b01b8 ntoskrnl.exe .rsrc
[snip]
```

가능한 IDT 변경사항에 대해서 자세한 정보를 얻고 싶다면, --verbose 옵션을 사용하라.

```
$ python vol.py -f rustock.vmem idt --verbose
Volatile Systems Volatility Framework 2.1_alpha
CPU Index Selector Value Module Section
-----
[snip]
0 2E 8 0x806b01b8 ntoskrnl.exe .rsrc
0x806b01b8 e95c2c0f31 JMP 0xb17a2e19
0x806b01bd e9832c0f31 JMP 0xb17a2e45
0x806b01c2 4e DEC ESI
0x806b01c3 44 INC ESP
0x806b01c4 4c DEC ESP
0x806b01c5 45 INC EBP
0x806b01c6 44 INC ESP
0x806b01c7 5f POP EDI
```

9.8. gdt

시스템의 GDT (Global Descriptor Table)을 출력하기 위해서, gdt 명령어를 사용한다. 호출문(call gate)에 설치된 Alipop 같은 루트킷들을 탐지하는데 유용하다. 그래서 사용자 모드 프로그램들은 커널모드에 직접적으로 호출 할 수 있다. (CALL FAR 명령어를 이용함)

만약 시스템이 멀티 CPUs 를 가지고 있다면 각각 프로세서의 GDT 를 보여준다.

아래 출력결과에서, 당신은 selector 0x3e0 이 공격당하고 있다는 것을 볼수 있고, 32-bit 호출문(call gate)의 목적으로 이용되는 것을 확인할 수 있다. 그 호출문 주소는 실행이 진행되고 있는 0x8003f00 이다.

Volatility Command 2.1 번역 문서

```
$ python vol.py -f alipop.vmem gdt
Volatile Systems Volatility Framework 2.1_alpha
CPU      Sel Base      Limit Type      DPL Gr  Pr
-----
0        0x0 0x00ffdf0a  0xdbbb TSS16 Busy      2 By  P
0        0x8 0x00000000 0xffffffff Code RE Ac    0 Pg  P
0        0x10 0x00000000 0xffffffff Data RW Ac    0 Pg  P
0        0x18 0x00000000 0xffffffff Code RE Ac    3 Pg  P
0        0x20 0x00000000 0xffffffff Data RW Ac    3 Pg  P
0        0x28 0x80042000 0x20ab TSS32 Busy      0 By  P
0        0x30 0xffdf0000 0x1fff Data RW Ac    0 Pg  P
0        0x38 0x00000000 0xfff Data RW Ac    3 By  P
0        0x40 0x00000400 0xfff Data RW      3 By  P
0        0x48 0x00000000 0x0 <Reserved>      0 By  Np
[snip]
0        0x3d0 0x00008003 0xf3d8 <Reserved>      0 By  Np
0        0x3d8 0x00008003 0xf3e0 <Reserved>      0 By  Np
0        0x3e0 0x8003f000 0x0 CallGate32      3 -   P
0        0x3e8 0x00000000 0xffffffff Code RE Ac    0 Pg  P
0        0x3f0 0x00008003 0xf3f8 <Reserved>      0 By  Np
0        0x3f8 0x00000000 0x0 <Reserved>      0 By  Np
```

만약 그 공격에 대해서 더 조사하고 싶다면, 당신은 아래 보이는 것처럼 [CommandReference21#volshell](#) 브레이크를 적용할 수 있다. 그리고 그 호출문(call gate)에 코드를 디스어셈블리 할 수 있다.

```
$ python vol.py -f alipop.vmem volshell
Volatile Systems Volatility Framework 2.1_alpha
Current context: process System, pid=4, ppid=0 DTB=0x320000
Welcome to volshell! Current memory image is:
file:///Users/Michael/Desktop/alipop.vmem
To get help, type 'hh()'

>>> dis(0xffdf0adb, length=32)
0xffdf0adb c8000000      ENTER 0x0, 0x0
0xffdf0adf 31c0        XOR EAX, EAX
0xffdf0ae1 60        PUSHA
0xffdf0ae2 8b5508      MOV EDX, [EBP+0x8]
0xffdf0ae5 bb00704d80    MOV EBX, 0x804d7000
0xffdf0aea 8b4b3c      MOV ECX, [EBX+0x3c]
0xffdf0aed 8b6c0b78    MOV EBP, [EBX+ECX+0x78]

>>> db(0xffdf0adb + 75, length = 512)
ffdf0b26 d9 03 1c 81 89 5c 24 1c 61 c9 c2 04 00 7e 00 80 .....$.a....~..
ffdf0b36 00 3b 0b df ff 5c 00 52 00 65 00 67 00 69 00 73 ;.....R.e.g.i.s
ffdf0b46 00 74 00 72 00 79 00 5c 00 4d 00 61 00 63 00 68 .t.r.y...M.a.c.h
ffdf0b56 00 69 00 6e 00 65 00 5c 00 53 00 4f 00 46 00 54 .i.n.e...S.O.F.T
ffdf0b66 00 57 00 41 00 52 00 45 00 5c 00 4d 00 69 00 63 .W.A.R.E...M.i.c
ffdf0b76 00 72 00 6f 00 73 00 6f 00 66 00 74 00 5c 00 57 .r.o.s.o.f.t...W
ffdf0b86 00 69 00 6e 00 64 00 6f 00 77 00 73 00 5c 00 43 .i.n.d.o.w.s...C
ffdf0b96 00 75 00 72 00 72 00 65 00 6e 00 74 00 56 00 65 .u.r.r.e.n.t.V.e
ffdf0ba6 00 72 00 73 00 69 00 6f 00 6e 00 5c 00 52 00 75 .r.s.i.o.n...R.u
ffdf0bb6 00 6e 00 00 00 06 00 08 00 c3 0b df ff 71 00 51 .n.....q.Q
ffdf0bc6 00 00 00 43 00 3a 00 5c 00 57 00 49 00 4e 00 44 ...C...W.I.N.D
ffdf0bd6 00 4f 00 57 00 53 00 5c 00 61 00 6c 00 69 00 2e .O.W.S...a.l.i..
ffdf0be6 00 65 00 78 00 65 00 00 00 26 00 28 00 f7 0b df .e.x.e...&.(....
ffdf0bf6 ff 5c 00 53 00 79 00 73 00 74 00 65 00 6d 00 52 ...S.y.s.t.e.m.R
ffdf0c06 00 6f 00 6f 00 74 00 5c 00 61 00 6c 00 69 00 2e .o.o.t...a.l.i..
ffdf0c16 00 65 00 78 00 65 00 00 00 00 00 e8 03 00 ec 00 .e.x.e.....
ffdf0c26 00 ff ff 00 00 00 9a cf 00 4d 5a 90 00 03 00 00 .....MZ.....
ffdf0c36 00 04 00 00 00 ff ff 00 00 b8 00 00 00 00 00 00 .....
ffdf0c46 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .@.....
ffdf0c56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
ffdf0c66 00 00 00 00 00 e0 00 00 00 0e 1f ba 0e 00 b4 09 .....
ffdf0c76 cd 21 b8 01 4c cd 21 54 68 69 73 20 70 72 6f 67 .!..L.!This prog
ffdf0c86 72 61 6d 20 63 61 6e 6e 6f 74 20 62 65 20 72 75 ram cannot be ru
ffdf0c96 6e 20 69 6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d n in DOS mode...
```

9.9. threads

이 명령어는 당신이 광범위하게 스레드에 대해 상세히 살펴볼 때 사용된다. 각 스레드의 레지스터들의(만약 가능하다면) 정보들, 스레드의 시작 주소에 있는 코드의 어셈블리, 그리고 조사하는 부분과 관련된 다양한 다른 필드들의 정보들도 포함된다. 어떤 주어진 시스템은 수백개의 스레드들을 가지고 있는 이후로 이것을 정렬하기는 힘들지만, 이 명령어는 찾으려는 스레드들에게 기술적인 태그들을 할당한다. 그리고 당신은 -F 나 -filter 옵션을 사용해서 태그 이름을 통해 필터링 할 수 있다.

가능한 tags/filters 의 리스트 항목을 보고 싶다면, -L 옵션을 사용하라.

```
$ python vol.py -f test.vmem threads -L
Volatile Systems Volatility Framework 2.1_alpha
Tag          Description
-----
DkExit        Detect inconsistencies wrt exit times and termination
HwBreakpoints Detect threads with hardware breakpoints
ScannerOnly   Detect threads no longer in a linked list
HideFromDebug Detect threads hidden from debuggers
OrphanThread  Detect orphan threads
AttachedProcess Detect threads attached to another process
HookedSSDT    Detect threads using a hooked SSDT
SystemThread  Detect system threads
```

만약에 특정한 필터를 사용하지 않았다면, 그 명령어는 모든 스레드들의 정보를 출력할 것이다. 달리 말하면, 당신은 구분점(coma)을 이용해서 single 필터와 multiple 필터들 특정할 수 있다. 아래 예제는 스레드를 소유하고 있는 스레드를 이외에 프로세스 맥락에서 현재 실행되고 있는 스레드들을 찾아내는 것을 보여준다.

```
$ python vol.py -f XPSP3.vmem threads -F AttachedProcess
Volatile Systems Volatility Framework 2.1_alpha
-----
ETHREAD: 0x81eda7a0 Pid: 4 Tid: 484
Tags: SystemThread,AttachedProcess,HookedSSDT
Created: 2011-04-18 16:03:38
Exited: -
Owning Process: 0x823c8830 System
Attached Process: 0x81e3c458 services.exe
State: Running
BasePriority: THREAD_PRIORITY_NORMAL
TEB: 0x00000000
StartAddress: 0xb1805f1a windev-5e93-fd3.sys
ServiceTable: 0x80553020
[0] 0x80501bbc
[0x47] NtEnumerateKey 0xb1805944 windev-5e93-fd3.sys
[0x49] NtEnumerateValueKey 0xb1805aca windev-5e93-fd3.sys
[0x91] NtQueryDirectoryFile 0xb18055ee windev-5e93-fd3.sys
[1] -
[2] -
[3] -
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_SYSTEM
b1805f1a: 8bff MOV EDI, EDI
b1805f1c: 55 PUSH EBP
b1805f1d: 8bec MOV EBP, ESP
b1805f1f: 51 PUSH ECX
b1805f20: 51 PUSH ECX
```


Volatility Command 2.1 번역 문서

첫번째, 프로세스 ID 와 스레드 ID 와 함께 ETHREAD 오브젝트의 가상 주소를 볼 수 있다. 다음에는 그 스레드(SystemThread, AttachedProcess, !HookedSSDT), 생성/종료된 시간들, 상태, 우선순위, 시작 주소 등에 할당된 모든 태그들을 볼 수 있다. 각 서비스 테이블의 주소를 포함해서 SSDT base 를 보여주고 그 테이블들 안에 있는 훅(hooked)이 된 함수들을 보여준다. 마지막으로 당신은 스레드의 시작 주소의 어셈블리를 확인할 수 있다.

threads 명령어에 당신 고유의 해결방법을 어떻게 추가할지에 대해 각 tag/filter 그리고 명령어(instructions)들의 자세한 내용을 알고 싶다면, [Investigating Windows Threads with Volatility](#) 을 보라.

참고 : 이 명령어의 소개와 함께, 두개 이전의 명령어는(orphan_threads, ssdt_by_threads) 추천하고 싶지 않다.

9.10. callbacks

Volatility 는 중요한 알림 루틴(notification routines) 과 커널 콜백(callbacks)의 모음을 출력할 수 있는 기능을 가진 메모리 포렌식 플랫폼이다. 루트킷들, 안티 바이러스 세트들, 동적 분석 도구들 (Sysinternals 의 Process Monitor, Tcpview 같은), 많은 윈도우즈 커널의 컴포넌트들은 이벤트들을 모니터링하거나 반응하기 위해서 이런 콜백들을 사용한다.

- PsSetCreateProcessNotifyRoutine (process creation).
- PsSetCreateThreadNotifyRoutine (thread creation).
- PsSetImageLoadNotifyRoutine (DLL/image load).
- IoRegisterFsRegistrationChange (file system registration).
- KeRegisterBugCheck and KeRegisterBugCheckReasonCallback.
- CmRegisterCallback (registry callbacks on XP).
- CmRegisterCallbackEx (registry callbacks on Vista and 7).
- IoRegisterShutdownNotification (shutdown callbacks).
- DbgSetDebugPrintCallback (debug print callbacks on Vista and 7).
- DbgkLkmdRegisterCallback (debug callbacks on 7).

아래 BlackEnergy2 악성코드에 의해 설치된 스레드 생성 콜백을 탐지하는 예제를 볼 수 있다. 당신은 악의적인 콜백을 감지할 수 있다. 왜냐하면, 그 소유자는 00004A2A 이고 BlackEnergy2 는 8 hex 값으로 모듈 이름이 정해져 있기 때문이다.

```
$ python vol.py -f be2.vmem callbacks
Volatile Systems Volatility Framework 2.1_alpha
Type Callback Owner
PsSetCreateThreadNotifyRoutine 0xff0d2ea7 00004A2A
PsSetCreateProcessNotifyRoutine 0xfc58e194 vmci.sys
KeBugCheckCallbackListHead 0xfc1e85ed NDIS.sys (Ndis miniport)
KeBugCheckCallbackListHead 0x806d57ca hal.dll (ACPI 1.0 - APIC platform UP)
KeRegisterBugCheckReasonCallback 0xfc967ac0 mssmbios.sys (SMBiosData)
KeRegisterBugCheckReasonCallback 0xfc967a78 mssmbios.sys (SMBiosRegistry)
```


Volatility Command 2.1 번역 문서

아래 Rustock 루트킷에 의해서 설치되는 악의적인 프로세스 생성 콜백을 탐지하는 것을 볼 수 있다. (WDriverWpe386 에 의해 소유된 메모리를 가르키고 있는)

```
$ python vol.py -f rustock.vmem callbacks
Volatile Systems Volatility Framework 2.1_alpha
Type                Callback Owner
PsSetCreateProcessNotifyRoutine 0xf88bd194 vmci.sys
PsSetCreateProcessNotifyRoutine 0xb17a27ed '\\Driver\\pe386'
KeBugCheckCallbackListHead      0xf83e65ef NDIS.sys (Ndis miniport)
KeBugCheckCallbackListHead      0x806d77cc hal.dll (ACPI 1.0 - APIC platform UP)
KeRegisterBugCheckReasonCallback 0xf8b7aab8 mssmbios.sys (SMBiosData)
KeRegisterBugCheckReasonCallback 0xf8b7aa70 mssmbios.sys (SMBiosRegistry)
KeRegisterBugCheckReasonCallback 0xf8b7aa28 mssmbios.sys (SMBiosDataACPI)
KeRegisterBugCheckReasonCallback 0xf76201be USBPORT.SYS (USBPORT)
KeRegisterBugCheckReasonCallback 0xf762011e USBPORT.SYS (USBPORT)
KeRegisterBugCheckReasonCallback 0xf7637522 VIDEOPT.SYS (Videoprt)
[snip]
```

아래 Ascesso 루트킷에 의해 설치된 악의적인 레지스터리 변경 callback 을 탐지하는 것을 볼 수 있다. 소유자를 가지고 있지 않는 0x8216628f 를 가르키고 있는 하나의 CmRegisterCallback 이 있다. 당신은 또한 같은 주소에 두개의 GenericKernelCallback 을 볼 수 있다. 이것은 notifyroutines 이 다양한 방법으로 콜백들을 찾아내기 때문이다. 이것은 list traversal 과 pool tag 검색이 조합되었다. 만약에 list traversal 이 실패를 한다면, 당신은 pool tag 검색으로 정보를 확인할 수 있다. 그러나 윈도우즈 커널은 다양한 callbacks 을 위해서 pool tags 와 동일한 형태를 사용한다. 그래서 generic 으로 이름을 명한다.

```
$ python vol.py -f ascesso.vmem callbacks
Volatile Systems Volatility Framework 2.1_alpha
Type                Callback Owner
IoRegisterShutdownNotification 0xf853c2be ftdisk.sys (\Driver\Ftdisk)
IoRegisterShutdownNotification 0x805f5d66 ntoskrnl.exe (\Driver\WMIxWDM)
IoRegisterShutdownNotification 0xf83d98f1 Mup.sys (\FileSystem\Mup)
IoRegisterShutdownNotification 0xf86aa73a MountMgr.sys (\Driver\MountMgr)
IoRegisterShutdownNotification 0x805cdef4 ntoskrnl.exe (\FileSystem\RAW)
CmRegisterCallback      0x8216628f UNKNOWN (--)
GenericKernelCallback    0xf888d194 vmci.sys
GenericKernelCallback    0x8216628f UNKNOWN
GenericKernelCallback    0x8216628f UNKNOWN
```

9.11. devicetree

Windows 는 계층화된(layered) 드라이버 아키텍처나 드라이버 체인을 사용합니다. 그래서 여러 개의 드라이버가 검사하거나 IRP 에 응답할 수 있다.

루트 킷은 종종 필터링 용도 목적으로(파일을 숨기거나, 네트워크 연결을 숨기기 위해서, 키 입력이나 마우스 움직임을 훔치기 위해서) 이 체인에 드라이버를 (또는 장치) 삽입합니다. devicetree 플러그인은 장치 (_DRIVER_OBJECT.DeviceObject.NextDevice 를 확인함으로써) 및 첨부된 장치 (_DRIVER_OBJECT.DeviceObject.AttachedDevice)에 드라이버 객체의 관계를 보여줍니다.

Volatility Command 2.1 번역 문서

아래 예제를 보면, Stuxnet 은 악의적인 이름없는 디바이스에 연결함으로써 `WFileSystemWNtfs` 에 영향을 주고 있다. 이 디바이스 자체는 이름이 정해지지 않음에도 불구하고, 이 디바이스 오브젝트는 드라이브를 정의하고 있다.(`WDriverWMRxNet`)

```
$ python vol.py -f stuxnet.vmem devicetree
Volatile Systems Volatility Framework 2.1_alpha
[snip]
DRV 0x0253d180 '\\FileSystem\\Ntfs'
---| DEV 0x82166020 (unnamed) FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x8228c6b0 (unnamed) - '\\FileSystem\\sr' FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x81f47020 (unnamed) - '\\FileSystem\\FltMgr' FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x81fb9680 (unnamed) - '\\Driver\\MRxNet' FILE_DEVICE_DISK_FILE_SYSTEM
---| DEV 0x8224f790 Ntfs FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x81eecdd0 (unnamed) - '\\FileSystem\\sr' FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x81e859c8 (unnamed) - '\\FileSystem\\FltMgr' FILE_DEVICE_DISK_FILE_SYSTEM
-----| ATT 0x81f0ab90 (unnamed) - '\\Driver\\MRxNet' FILE_DEVICE_DISK_FILE_SYSTEM
[snip]
```

devicetree 플러그인은 드라이버들을 나타내기 위해 "DRV"를 사용한다, "DEV"는 디바이스들을 가리킨다. "ATT" 는 연결되어 있는 디바이스들을 나타낸다. (OSR 의 DeviceTree 유틸리티 처럼)

x64 버전은 매우 유사하게 결과가 도출된다.

```
$ /usr/bin/python2.6 vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 devicetreeVolatile
Volatile Systems Volatility Framework 2.1_alpha
DRV 0x174c6350 \Driver\mouhid
---| DEV 0xfffffa8000dfbc90 FILE_DEVICE_MOUSE
-----| ATT 0xfffffa8000ec7060 - \Driver\mouclass FILE_DEVICE_MOUSE
DRV 0x17660cb0 \Driver\rspndr
---| DEV 0xfffffa80005a1c20 rspndr FILE_DEVICE_NETWORK
DRV 0x17663e70 \Driver\ltdio
---| DEV 0xfffffa8000c78b70 ltdio FILE_DEVICE_NETWORK
DRV 0x17691d70 \Driver\cdrom
---| DEV 0xfffffa8000d00060 CdRom0 FILE_DEVICE_CD_ROM
DRV 0x176a7280 \FileSystem\Msfs
---| DEV 0xfffffa8000cac060 Mailslot FILE_DEVICE_MAILSLLOT
DRV 0x176ac6f0 \FileSystem\Npfs
---| DEV 0xfffffa8000cac320 NamedPipe FILE_DEVICE_NAMED_PIPE
DRV 0x176ade70 \Driver\tdx
---| DEV 0xfffffa8000cb8c00 RawIp6 FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cb8e30 RawIp FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cb7510 Udp6 FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cb7740 Udp FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cb7060 Tcp6 FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cad140 Tcp FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000cadbb0 Tdx FILE_DEVICE_TRANSPORT
DRV 0x176ae350 \Driver\Pssched
---| DEV 0xfffffa8000cc4590 Psched FILE_DEVICE_NETWORK
DRV 0x176aee70 \Driver\WfpLwf
DRV 0x176b93a0 \Driver\AFD
---| DEV 0xfffffa8000cb9180 Afd FILE_DEVICE_NAMED_PIPE
DRV 0x176c28a0 \Driver\NetBT
---| DEV 0xfffffa8000db4060 NetBT_Tcpip_{EE0434CC-82D1-47EA-B5EB-AF721863ACC2}
FILE_DEVICE_NETWORK
---| DEV 0xfffffa8000c1d8f0 NetBt_Wins_Export FILE_DEVICE_NETWORK
DRV 0x176c3930 \FileSystem\NetBIOS
---| DEV 0xfffffa8000cc3680 Netbios FILE_DEVICE_TRANSPORT
[snip]
```

9.12. psxview

이 플러그인은 프로세스 목록(Listing)들의 다양한 다른 소스들에 의해 출력되는 것들이 무엇인지, PsActiveProcessHead 에 포함되어 있는 것들과 비교함으로써 숨겨져 있는 프로세스들을 탐지하는데 도움을 준다.

- PsActiveProcessHead linked list
- EPROCESS pool scanning
- ETHREAD pool scanning (EPROCESS 를 참조한다.)
- PspCidTable
- Csrss.exe handle table
- Csrss.exe internal linked list

Vista 와 Windows 7 에서는 csrss.exe 의 프로세스들의 내부 목록들은 불가능하다. 특정 페이지들이 메모리에 위치하고 있지 않는 XP Images 들도 불가능할 것이다.

아래 psxview 플러그인으로 Proloco 악성코드의 탐지하는 예제를 보이고 있다. 각각의 프로세스에서 탐지가 되지 않는다면 컬럼에 "False"라고 표시된다. pslist (PsActiveProcessHead)를 제외하고 모든 컬럼에서 보여지는 것을 보고 "1_doc_RCData_61"이 의심되는 것이라고 말할 수 있다.

```
$ python vol.py -f proloco.vmem psxview
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) Name PID pslist psscan thrdproc pspcidid csrss
-----
0x06499b80 svchost.exe 1148 True True True True True
0x04b5a980 VMwareUser.exe 452 True True True True True
0x0655fc88 VMUpgradeHelper 1788 True True True True True
0x0211ab28 TPAutoConnSvc.e 1968 True True True True True
0x04c2b310 wscntfy.exe 888 True True True True True
0x061ef558 svchost.exe 1088 True True True True True
0x06945da0 spoolsv.exe 1432 True True True True True
0x05471020 smss.exe 544 True True True True False
0x04a544b0 ImmunityDebugge 1136 True True True True True
0x069d5b28 vmttoolsd.exe 1668 True True True True True
0x06384230 vmacthlp.exe 844 True True True True True
0x010f7588 wuauclt.exe 468 True True True True True
0x066f0da0 csrss.exe 608 True True True True False
0x05f027e0 alg.exe 216 True True True True True
0x06015020 services.exe 676 True True True True True
0x04a065d0 explorer.exe 1724 True True True True True
0x049c15f8 TPAutoConnect.e 1084 True True True True True
0x0115b8d8 svchost.exe 856 True True True True True
0x01214660 System 4 True True True True False
0x01122910 svchost.exe 1028 True True True True True
0x04be97e8 VMwareTray.exe 432 True True True True True
0x05f47020 lsass.exe 688 True True True True True
0x063c5560 svchost.exe 936 True True True True True
0x066f0978 winlogon.exe 632 True True True True True
0x0640ac10 msixexec.exe 1144 False True False False False
0x005f23a0 rundll32.exe 1260 False True False False False
0x0113f648 1_doc_RCData_61 1336 False True True True True
```

아래는 x64 시스템에서 보여주는 것이다.

```
$ python vol.py -f ~/Desktop/win7_trial_64bit.raw --profile=Win7SP0x64 psxview
Volatile Systems Volatility Framework 2.1_alpha
Offset(P)      Name      PID pslist psscan thrdproc pspcidid csrss
-----
0x00000000173c5700 lsass.exe      444 True  True  True  True  True
0x00000000176006c0 csrss.exe      296 True  True  True  True  False
0x0000000017803b30 rundll32.exe   2016 True  True  True  True  True
0x0000000017486690 spoolsv.exe    1076 True  True  True  True  True
0x0000000017db7960 svchost.exe     856 True  True  True  True  True
0x0000000017dd09e0 svchost.exe     916 True  True  True  True  True
0x0000000017606b30 csrss.exe      344 True  True  True  True  False
0x000000001769a630 regsvr32.exe   1180 True  True  False  True  False
0x0000000017692300 wininit.exe     332 True  True  True  True  True
[snip]
```

9.13. timers

이 명령어는 설치되어 있는 kernel timers (KTIMER) 들과 연계 되어 있는 DPCs (Deferred Procedure Calls)을 출력한다. Zero Access, Rustock, Stuxnet 같은 루트킷(Rootkit) 들은 DPC 와 함께 timers 가 등록되어 있다. 악성코드가 수많은 다른 방법들을 통해서 커널 숨어들어가더라도 KTIMERs 를 찾고, DPC 의 주소를 확인함으로써, 악의적인 코드 범위를 빨리 찾을 수 있다.

아래 예제가 있다. 아래 부분에 보면 timers 중 하나가 "UNKNOWN" 모듈인 것이 있다. (DPC 는 커널 메모리의 알려지지 않은 범위를 가리키게 된다.) 이것은 거의 확실하게 rootkit 이 숨겨져 있는 곳이다.

```
$ python vol.py -f rustock-c.vmem timers
Volatile Systems Volatility Framework 2.1_alpha
Offset(V) DueTime      Period(ms) Signaled Routine      Module
-----
0xf730a790 0x00000000:0x6db0f0b4      0 -      0xf72fb385 srv.sys
0x80558a40 0x00000000:0x68f10168    1000 Yes    0x80523026 ntoskrnl.exe
0x821cb240 0x00000000:0x68fa8ad0      0 -      0xf84b392e sr.sys
0x8054f288 0x00000000:0x69067692      0 -      0x804e5aec ntoskrnl.exe
0xf7c13fa0 0x00000000:0x74f6fd46    60000 Yes    0xf7c044d3 ipsec.sys
0xf7c13b08 0x00000000:0x74f6fd46      0 -      0xf7c04449 ipsec.sys
0x8055a300 0x00000008:0x61e82b46      0 -      0x80533bf8 ntoskrnl.exe
0xf7c13b70 0x00000008:0x6b719346      0 -      0xf7c04449 ipsec.sys
0xf7befbf0 0x00000000:0x690d9da0      0 -      0xf89aa3f0 TDI.SYS
0x81ea5ee8 0x00000000:0x7036f590      0 -      0x80534016 ntoskrnl.exe
0x81d69180 0x80000000:0x3ae334ee      0 -      0x80534016 ntoskrnl.exe
0xf70d0040 0x00000000:0x703bd2ae      0 -      0xf70c3ae8 HTTP.sys
0xf7a74260 0x00000000:0x75113724    60000 Yes    0xf7a6cf98 ipnat.sys
0x82012e08 0x00000000:0x8a87d2d2      0 -      0xf832653c ks.sys
0x81f01358 0x00000008:0x6b97b8e6      0 -      0xf7b8448a netbt.sys
0x81f41218 0x00000000:0x6933c340      0 -      0xf7b8448a netbt.sys
0x805508d0 0x00000000:0x6ba6cdb6    60000 Yes    0x804f3b72 ntoskrnl.exe
0x80559160 0x00000000:0x695c4b3a      0 -      0x80526bac ntoskrnl.exe
0x820822e4 0x00000000:0xa2a56bb0    150000 Yes    0x81c1642f UNKNOWN
0xf842f150 0x00000000:0xb5cb4e80      0 -      0xf841473e Ntfs.sys
0x821811b0 0x00000131:0x34c6cb8e      0 -      0xf83fafdf NDIS.sys
...
```

Volatility Command 2.1 번역 문서

기억할 것 : timers 는 대상 운영체제에 따라서 다른 방식으로 나타난다. 윈도우즈들은 XP, 2003, 2008 그리고 Vista 대상으로 global 한 값들로 timers 를 저장한다. Windows 7 부터 그 timers 는 KPCR(Kernel Processor Control Region)의 프로세서세서적인 특정 범위안에서 위치한다. Volatility 2.1 에서 만약 멀티 CPUs 라고 한다면 timer 플러그인은 모든 KPCRs 를 찾아낸다. 그리고 각 CPU 에 위치하고 있는 timers 를 출력한다.

timers 오브젝트에 대해 더 많은 정보를 알고 싶다면, [Ain't Nuthin But a K\(Timer\) Thing, Baby](#) 참고하라.

9.14. driverirp

드라이버의 IRP(주요 함수) 테이블을 출력하기 위해서는 driverirp 명령어를 사용해야 한다. 이 명령어는 driverscan 으로부터 상속된 명령어로 DRIVER_OBJECT 들의 위치를 알아낼 수 있게 한다. 그리고 함수 테이블을 순환하며 각각 함수의 목적, 함수 주소, 그리고 모듈 주소를 출력한다.

많은 드라이버들이 타당한 목적으로 IRP 함수들을 다른 드라이버 앞으로 보낸다. 따라서 포함된 모듈에 따라 연결된 IRP 함수들을 찾아내는 것은 좋은 방법이 아니다. 대신에, 이것은 모든 것을 출력하여 직접 판단하게 한다. 명령어는 IRP 함수들의 인라인 혹은 또한 확인하며 부가적으로 IRP 주소(-v 또는 --verbose 로 이것을 활성화 시킬 수 있다)에 있는 분해된 명령들을 출력한다.

이 명령어는 수식 필터로 따로 명시 하지 않는한, 모든 드라이버들의 정보를 출력한다.

```
$ python vol.py -f tdl3.vmem driverirp -r vm SCSI
Volatile Systems Volatility Framework 2.1_alpha

-----
DriverName: vm SCSI
DriverStart: 0xf9db8000
DriverSize: 0x2c00
DriverStartIo: 0xf97ea40e
  0 IRP_MJ_CREATE          0xf9db9cbd vm SCSI.sys
  1 IRP_MJ_CREATE_NAMED_PIPE 0xf9db9cbd vm SCSI.sys
  2 IRP_MJ_CLOSE          0xf9db9cbd vm SCSI.sys
  3 IRP_MJ_READ           0xf9db9cbd vm SCSI.sys
  4 IRP_MJ_WRITE          0xf9db9cbd vm SCSI.sys
  5 IRP_MJ_QUERY_INFORMATION 0xf9db9cbd vm SCSI.sys
  6 IRP_MJ_SET_INFORMATION 0xf9db9cbd vm SCSI.sys
  7 IRP_MJ_QUERY_EA       0xf9db9cbd vm SCSI.sys
[snip]
```

출력물에서는 vm SCSI.sys 드라이버가 TDL3 루트킷에 의해 감염되었다고 보여지지 않는다. 비록 모든 IRP 들이 vm SCSI.sys 를 다시 가리키지만 정확한 우회 루트킷 감지 툴의 목적으로써 TDL3 에 의해 그 영역의 특정 부분을 가르키는 것이다. 확장 정보를 얻기 위해서는 --verbose 를 사용하면 된다.

Volatility Command 2.1 번역 문서

```
$ python vol.py -f tdl3.vmem driverirp -r vm SCSI --verbose
Volatile Systems Volatility Framework 2.1_alpha
-----
DriverName: vm SCSI
DriverStart: 0xf9db8000
DriverSize: 0x2c00
DriverStartIo: 0xf97ea40e
  0 IRP_MJ_CREATE                                0xf9db9cbd vm SCSI.sys

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

  1 IRP_MJ_CREATE_NAMED_PIPE                        0xf9db9cbd vm SCSI.sys
0xf9db9cbd a10803dfff      MOV EAX, [0xffdf0308]
0xf9db9cc2 ffa0fc000000    JMP DWORD [EAX+0xfc]
0xf9db9cc8 0000          ADD [EAX], AL
0xf9db9cca 0000          ADD [EAX], AL

[snip]
```

이제 TDL3 가 vm SCSI.sys 드라이버안의 특정 부분으로 모든 IRP 들이 재 연결 되는 것을 볼 수 있다. 이 코드는 KUSER_SHARED_DATA 영역의 장소로 0xffdf0308 로 가리켜지는 어떠한 주소로든지 넘어간다.

10. 그외

10.1. strings

주어진 이미지와 <십진수_오프셋>:<문자열> 형식의 문자열들인 파일을 위해, 특정 문자열을 찾을 수 있는 프로세스와 가상 주소들을 출력한다. 입력을 위한 도구는 Microsoft Sysinternals'Strings utility 또는 오프셋:문자열 형태의 비슷한 출력을 지원하는 다른 도구들의 출력이다. 입력 오프셋은 파일이나 이미지의 시작지점의 물리 오프셋이다.

Sysinternals Strings 는 Wine 을 사용하는 Linux/Max 에서 사용가능하다. 출력은 Volatility strings 플러그인에 사용하기 위해 파일의 형태로 리다이렉션되어야 한다. 만약 GNU strings 명령어를 사용한다면, 십진 오프셋을 출력하기 위해 -td 옵션을 사용한다.(GNU strings 는 십육진 오프셋을 지원하지 않는다.) 사용법의 몇몇 예시는 아래와 같다:

Windows

```
C:\W> strings.exe -q -o -accepteula win7.dd > win7_strings.txt
```

Linux/Mac

```
$ wine strings.exe -q -o -accepteula win7.dd > win7_strings.txt
```

Sysinternals strings 프로그램이 끝나는 데는 시간이 좀 걸릴 수 있다. -q 옵션은 헤더의 출력을 생략하고, -o 옵션은 각 줄의 오프셋을 출력하기 때문에 꼭 사용해야 한다. 결과는 이미지로 부터 얻은 오프셋과 문자열로 이루어진 텍스트 파일이다.

[예시]

```
16392: @@@
17409:
17441: !!!
17473: ""
17505: ###
17537: $$$
17569: %%%
17601: &&&
17633: ""
17665: (((
17697: )))
17729: ***
```

EnCase Keyword Export

EnCase 를 사용하여 출력된 키워드와 오프셋은 약간의 수정을 거쳐 사용할 수 있다. EnCase 는 문자를 바이트오더마크를 U+FEFF 을 사용하는 UTF-16 형태이기 때문에 strings 플러그인을 사용하는데 문제가 있다. 출력된 키워드 파일의 예시는 아래와 같다:

Volatility Command 2.1 번역 문서

File Offset Hit Text

```
114923 DHCP
114967 DHCP
115892 DHCP
115922 DHCP
115952 DHCP
116319 DHCP
```

[snip]

헤더를 제거하고 탭을 : 로 바꾼다.

```
114923:DHCP
114967:DHCP
115892:DHCP
115922:DHCP
115952:DHCP
116319:DHCP
```

[snip]

헥스에디터를 사용하면, 바이트오더마크를 U+FEFF 을 사용하는 UTF-16 형태라는 것을 볼 수 있다.

```
$ file export.txt
export.txt: Little-endian UTF-16 Unicode text, with CRLF, CR line terminators

$ xxd export.txt |less

00000000: fffe 3100 3100 3400 3900 3200 3300 3a00  ..1.1.4.9.2.3..
```

[snip]

이를 ANSI 또는 UTF-8 형태로 변환해야한다. 윈도우에서는 텍스트 파일을 열어서 "다른이름으로 저장" 를 이용하여 ANSI(아래의 "인코딩" 메뉴에서 선택)로 저장할 수 있다. 리눅스에서는 iconv 를 사용할 수 있다:

```
$ iconv -f UTF-16 -t UTF-8 export.txt > export1.txt
```

주의 : "strings" 파일에서 마지막에 빈 줄이 있어서는 안된다.

이제 수정된 파일과 원본 파일의 결과가 다른것을 볼 수 있다:

```
$ ./vol.py -f Bob.vmem --profile=WinXPSP2x86 strings -s export.txt
Volatile Systems Volatility Framework 2.1_alpha
ERROR : volatility.plugins.strings: String file format invalid.

$ ./vol.py -f Bob.vmem --profile=WinXPSP2x86 strings -s export1.txt
Volatile Systems Volatility Framework 2.1_alpha
0001c0eb [kernel:2147598571] DHCP
0001c117 [kernel:2147598615] DHCP
0001c4b4 [kernel:2147599540] DHCP
0001c4d2 [kernel:2147599570] DHCP
0001c4f0 [kernel:2147599600] DHCP
0001c65f [kernel:2147599967] DHCP
0001c686 [kernel:2147600006] DHCP
```

[snip]

Volatility Command 2.1 번역 문서

주의 : Volatility strings 의 출력은 매우 장황하기때문에 리다이렉션 또는 파일로 저장하는 것이 좋다. 아래 명령어에서 출력을 저장하는 것은 -output-file 옵션과 "win7_vol_strings.txt" 파일이름이다.

```
$ python vol.py --profile=Win7SP0x86 strings -f win7.dd -s win7_strings.txt --output-file=win7_vol_strings.txt
```

기본적으로 strings 는 PsActiveProcessHead 가 가리키는 이중 연결리스트를 순회하며 프로세스들을 찾아서 커널주소를 포함하여 출력한다. strings 는 -S 옵션을 사용하여 숨겨진 프로세스들을 출력한다.

```
$ python vol.py --profile=Win7SP0x86 strings -f win7.dd -s win7_strings.txt --output-file=win7_vol_strings.txt -S
```

또한 EPROCESS 오프셋도 제공한다.

```
$ python vol.py --profile=Win7SP0x86 strings -f win7.dd -s win7_strings.txt --output-file=win7_vol_strings.txt -o 0x04a291a8
```

strings 플러그인이 완료되는데에는 시간이 좀 걸린다. 완료되면, 아래의 형태와 같은 출력 파일이 생성된다.

```
physical_address [kernel_or_pid:virtual_address] string
```

PID 와 커널 참조주소를 볼수있는 출력의 예시이다:

```
$ less win7_vol_strings.txt

000003c1 [kernel:4184445889] '<'@
00000636 [kernel:4184446518] 8,t
000006c1 [kernel:4184446657] w#r
000006d8 [kernel:4184446680] sQOtN2
000006fc [kernel:4184446716] t+a`j
00000719 [kernel:4184446745] aas
0000072c [kernel:4184446764] Invalid partition ta
00000748 [kernel:4184446792] r loading operating system
00000763 [kernel:4184446819] Missing operating system
000007b5 [kernel:4184446901] ,Dc
0000400b [kernel:2147500043 kernel:4184461323] 3TYk
00004056 [kernel:2147500118 kernel:4184461398] #:s
000040b0 [kernel:2147500208 kernel:4184461488] COO
000040e9 [kernel:2147500265 kernel:4184461545] BrvWo
000040f0 [kernel:2147500272 kernel:4184461552] %Sz
000040fc [kernel:2147500284 kernel:4184461564] A0?0=
00004106 [kernel:2147500294 kernel:4184461574] 7http://crl.microsoft.com/pki/crl/products/WinIntPCA.crl0U

[snip]

00369f14 [1648:1975394068] Ph$I
00369f2e [1648:1975394094] 9}$
00376044 [1372:20422724] Ju0w
0037616d [1372:20423021] msxml6.dll
003761e8 [1372:20423144] U'H
003762e3 [1372:20423395] }e_
0037632e [1372:20423470] xnA

[snip]
```

Volatility Command 2.1 번역 문서

```
03678031 [360:2089816113 596:2089816113 620:2089816113 672:2089816113 684:2089816113
844:2089816113 932:2089816113 1064:2089816113 1164:2089816113 1264:2089816113 1516:2089816113
1648
:2089816113 1896:2089816113 1904:2089816113 1756:2089816113 512:2089816113 1372:2089816113
560:2089816113] A$9B
```

한번 strings 출력을 함으로써, 메모리에서 수상한 문자열을 가진 프로세스를 볼 수 있고 시야를 좁힐 수 있다. 또한 grep 을 사용하여 특정 문자열이나 정규표현식에 적용되는 문자열들을 추출할 수 있다. 아래는 특정 명령어의 예시이다:

```
$ grep [command or pattern] win7_vol_strings.txt > strings_of_interest.txt
```

모든 IP 찾기:

```
$ cat win7_vol_strings.txt | \
perl -e 'while(<>){if(/(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)/){print $_;}}' > IPs.txt
```

모든 URL 찾기:

```
$ cat win7_vol_strings.txt | \
perl -e 'while(<>){ if(/(http|https|ftp|mail):[\/\w.]+/){print $_;}}' > URLs.txt
```

문맥에 따라 검색내용이 다양하게 바뀔 수 있다.

10.2. volshell

메모리 이미지를 대화형식으로 분석하려면 volshell 명령어를 사용한다. 이는 메모리 덤프에서 WinDbg 랑 유사한 인터페이스를 제공한다. 예를 들어 아래와 같은 것들을 할수 있다:

- 프로세스들의 목록화
- 프로세스들의 문맥 교환
- 구조체와 객체들의 형식 출력
- 주어진 주소에 타입 덮어씌우기
- 링크드 리스트 순회
- 주어진 주소의 코드 디스어셈

주의 : IPython 이 설치되어있다면, tab-completion 과 저장된 명령어 히스토리를 사용 할 수 있다.

volshell 을 살펴보자:

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp volshell
Volatile Systems Volatility Framework 2.0
Current context: process System, pid=4, ppid=0 DTB=0x185000
Welcome to volshell! Current memory image is:
file:///Users/M/Desktop/win7.dmp
```

Volatility Command 2.1 번역 문서

```
To get help, type 'hh()'
>>> hh()
ps() : Print a process listing.
cc(offset=None, pid=None, name=None) : Change current shell context.
dd(address, length=128, space=None) : Print dwords at address.
db(address, length=128, width=16, space=None) : Print bytes as canonical hexdump.
hh(cmd=None) : Get help on a command.
dt(objct, address=None, address_space=None) : Describe an object or show type info.
list_entry(head, objname, offset=-1, fieldname=None, forward=True) : Traverse a _LIST_ENTRY.
dis(address, length=128, space=None) : Disassemble code at a given address.

For help on a specific command, type 'hh(<command>)'
>>>
```

explorer.exe 의 메모리의 0x779f0000 주소를 보려면 먼저, 프로세스들을 출력하여 PID 또는 Explorer 의 ERPROCESS 를 습득한다.(주의: 커널메모리의 데이터를 보려면, 문맥교환이 먼저다.)

```
>>> ps()
Name      PID  PPID  Offset
System    4    0     0x83dad960
smss.exe  252   4     0x84e47840
csrss.exe 348   340    0x8d5ffd40
wininit.exe 384   340    0x84e6e3d8
csrss.exe 396   376    0x8d580530
winlogon.exe 424   376    0x8d598530
services.exe 492   384    0x8d4cc030
lsass.exe 500   384    0x8d6064a0
lsme.exe  508   384    0x8d6075d8
svchost.exe 616   492    0x8d653030
svchost.exe 680   492    0x8d673b88
svchost.exe 728   492    0x8d64fb38
taskhost.exe 1156  492    0x8d7ee030
dwm.exe   956   848    0x8d52bd40
explorer.exe 1880  1720   0x8d66c1a8
wuaucit.exe 1896  876    0x83ec3238
VMwareTray.exe 2144  1880   0x83f028d8
VMwareUser.exe 2156  1880   0x8d7893b0
[snip]
```

Explorer 의 문맥을 교환하고 db(표준 헥스값으로 출력) 또는 dd(더블워드 형태로 출력)로 데이터를 출력한다:

```
>>> dd(0x779f0000)
779f0000 00905a4d 00000003 00000004 0000ffff
779f0010 000000b8 00000000 00000040 00000000
779f0020 00000000 00000000 00000000 00000000
779f0030 00000000 00000000 00000000 000000e0
779f0040 0eba1f0e cd09b400 4c01b821 685421cd
779f0050 70207369 72676f72 63206d61 6f6e6e61
779f0060 65622074 6e757220 206e6920 20534f44
779f0070 65646f6d 0a0d0d2e 00000024 00000000
>>> db(0x779f0000)
779f0000 4d 5a 90 00 03 00 00 04 00 00 00 ff ff 00 00 MZ.....
779f0010 b8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
779f0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
779f0030 00 00 00 00 00 00 00 00 00 00 00 00 e0 00 00 .....
779f0040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 .....!..L.!Th
779f0050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
779f0060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS
779f0070 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00 mode....$......
```

explorer.exe 의 0x779f0000 에 PE 가 위치한다. PE 의 RVA 0x2506 부분부터 디스어셈을 하려면 아래와 같이 하면 된다:

Volatility Command 2.1 번역 문서

```
>>> dis(0x779f0000 + 0x2506)
0x779f2506 8d0c48          LEA ECX, [EAX+ECX*2]
0x779f2509 8b4508          MOV EAX, [EBP+0x8]
0x779f250c 8b4c4802        MOV ECX, [EAX+ECX*2+0x2]
0x779f2510 8d0448          LEA EAX, [EAX+ECX*2]
0x779f2513 e9c07f0300      JMP 0x77a2a4d8
0x779f2518 85f6           TEST ESI, ESI
0x779f251a 0f85c12c0700   JNZ 0x77a651e1
0x779f2520 8b4310          MOV EAX, [EBX+0x10]
0x779f2523 8b407c          MOV EAX, [EAX+0x7c]
0x779f2526 8b4b18          MOV ECX, [EBX+0x18]
0x779f2529 0fb7444102      MOVZX EAX, [ECX+EAX*2+0x2]
0x779f252e 894520          MOV [EBP+0x20], EAX
[snip]
```

주어진 운영체제의 EPROCESS 객체의 멤버들을 보려면 아래와 같이 하면 된다:

```
>>> dt("_EPROCESS")
'_EPROCESS' (704 bytes)
0x0  : Pcb                ['_KPROCESS']
0x98 : ProcessLock        ['_EX_PUSH_LOCK']
0xa0 : CreateTime         ['_LARGE_INTEGER']
0xa8 : ExitTime           ['_LARGE_INTEGER']
0xb0 : RundownProtect     ['_EX_RUNDOWN_REF']
0xb4 : UniqueProcessId    ['pointer', ['void']]
0xb8 : ActiveProcessLinks ['_LIST_ENTRY']
0xc0 : ProcessQuotaUsage  ['array', 2, ['unsigned long']]
0xc8 : ProcessQuotaPeak   ['array', 2, ['unsigned long']]
0xd0 : CommitCharge       ['unsigned long']
0xd4 : QuotaBlock         ['pointer', ['_EPROCESS_QUOTA_BLOCK']]
[snip]
```

explorer.exe 의 특정 오프셋에 EPROCESS 를 덮어쓰려면 아래와 같이 하면 된다:

```
>>> dt("_EPROCESS", 0x8d66c1a8)
[_EPROCESS _EPROCESS] @ 0x8D66C1A8
0x0  : Pcb                2372321704
0x98 : ProcessLock        2372321856
0xa0 : CreateTime         2010-07-06 22:38:07
0xa8 : ExitTime           1970-01-01 00:00:00
0xb0 : RundownProtect     2372321880
0xb4 : UniqueProcessId    1880
0xb8 : ActiveProcessLinks 2372321888
0xc0 : ProcessQuotaUsage  -
0xc8 : ProcessQuotaPeak   -
0xd0 : CommitCharge       4489
0xd4 : QuotaBlock         2372351104
[snip]
```

db, dd, dt 그리고 dis 명령어들은 모두 특정 주소 공간을 인자로 쓰기위한 추가적인 공간을 허용합니다. 사용하는 주소공간에 따라 다른 데이터를 볼수있다. Volshell 은 아래와 같이 몇가지 중요한 초기설정과 규칙들이 있다:

- 주소공간과 프로세스에서 cc 와 문맥교환이 없다면, 기본 커널공간(시스템 프로세스)을 사용한다.
- 주소공간과 cc 와 프로세스에서 문맥교환이 있다면, 현재/활성화된 프로세스의 공간을 사용한다.
- 명확한 주소공간을 지원한다면, 그 공간을 사용한다.

Volatility Command 2.1 번역 문서

scan 명령어들(psscan, connscan, etc.)을 사용할때, 오탐이 발생할 수 있다.scan 명령어들은 물리 오프셋(메모리 덤프파일의 오프셋)을 출력한다. 특정 구조체의 멤버의 정상여부를 위해 잠재적인 오탐 근처의 데이터를 스스로 조사하기 원한다면, hex viewer 로 메모리 덤프를 열고 물리 오프셋으로 이동하여 raw 바이트들을 봐야한다. 하지만, volshell 을 이용하여 추측한 물리 오프셋의 구조체를 덮어쓰는게 더 좋다. 이는 그들의 의도된 형태(DWORD, string, short, etc.)로 번역된 필드를 보게 해준다.

예시는 다음과 같다. 우선 물리 주소 공간을 예시한다:

```
>>> physical_space = utils.load_as(self._config, astype = 'physical')
```

오탐된 EPROCESS 의 추측 주소가 0x433308 이라면 아래와 같이 한다:

```
>>> dt("_EPROCESS", 0x433308, physical_space)
[_EPROCESS _EPROCESS] @ 0x00433308
0x0 : Pcb 4403976
0x6c : ProcessLock 4404084
0x70 : CreateTime 1970-01-01 00:00:00
0x78 : ExitTime 1970-01-01 00:00:00
...
```

volshell 을 사용하는 다른 기술은 비대화형태이다. 예를 들면, 커널 메모리의 주소를 그에 해당하는 물리 오프셋으로 변환하는 것이다.

```
$ echo "hex(self.addrspace.vtop(0x823c8830))" | python vol.py -f stuxnet.vmem volshell
Volatile Systems Volatility Framework 2.1_alpha
Current context: process System, pid=4, ppid=0 DTB=0x319000
Welcome to volshell! Current memory image is:
file:///mem/stuxnet.vmem
To get help, type 'hh()'
>>> '0x25c8830'
```

위와같이 커널 주소 0x823c8830 은 메모리 덤프의 물리오프셋 0x25c8830 으로 변환된다.

```
$ echo "cc(pid=4); dd(0x10000)" | [...]
```

아래와 같이 여러개의 명령어를 순차적으로 실행할 수 있다.

10.3. bioskbd

메모리의 BIOS 영역에서 키보드 입력을 읽기위해, bioskbd 명령어를 사용한다. 이는 HP, Intel, 그리고 Lenovo BIOS 와 SafeBoot, TrueCrypt, 그리고 BitLocker 에 입력한 비밀번호를 볼 수 있다. 메모리 덤프 툴에 따라서 필요한 BIOS 영역을 포함할 수도 있고 아닐 수도 있다. 더 많은 정보를 보려면, [Andreas Schuster 의 키보드버퍼로부터 비밀번호 읽기](#), [David Sharpe 의](#)

활성 윈도우 시스템에서 Volatility Bioskbd 명령어 함수를 복사하기, 그리고 [Jonathan Brossard](#)의 BIOS 키보드 버퍼를 이용하여 부팅 전의 인증 비밀번호 우회하기를 참고하라.

10.4. patcher

patcher 플러그인은 아래의 XML 파일과 같이 '-x'의 한개의 인자만 받는다. 그 XML 파일은 아래의 예제와 같이 요구되는 패치들을 특정한다.

```
<patchfile>
  <patchinfo method="pagescan" name="Some Descriptive Name">
    <constraints>
      <match offset="0x123">554433221100</match>
    </constraints>
    <patches>
      <setbytes offset="0x234">001122334455</setbytes>
    </patches>
  </patchinfo>
  <patchinfo>
    ...
  </patchinfo>
</patchfile>
```

XML 최상위 요소는 늘 patchfile 이고, 여러개의 patchinfo 를 포함한다. patchfile 이 실행될 때, 각 patchinfo 마다 한번씩 메모리를 조사하고, method 속성에 명시된 method 를 이용해 조사를 시도한다. 현재 지원하는 유일한 method 는 pagescan 이며 이는 각 patchinfo 요소를 명확하게 선언한다.

각 pagescan 은 단일 constraints 요소와 단일 patches 요소를 포함한 patchinfo 요소를 분류한다. 모든 constraints 가 정확한지, pathes 요소에 명시된 명령이 잘 수행되었는지 메모리의 각 페이지를 계속해서 조사한다.

constraints 요소는 일치하는 특정 오프셋 속성(페이지가 일치하는 위치를 명시한)을 가지고 있는 요소들의 특정 번호와 그와 일치하는 16 진 문자열을 포함한다.

patches 요소는 특정 숫자의 특정 오프셋 속성(데이터가 수정된 페이지의 위치를 명시한)을 가지고 있는 setbytes 요소를 포함하고 페이지에 적힐 16 진 문자열을 포함한다.

주의 : patcher 플러그인을 실행할때, 명령줄에 쓰기 옵션(-w) 지정하지 않는 한 수정이 안된다.

10.5. pagecheck

pagecheck 플러그인은 커널 DTB (System/Idle 프로세스로부터)을 사용하고 페이지들(pages)이 위치하고 있는 메모리(AddressSpace.get_available_pages 메소드를 사용)를 추정한다. 각 페이지별로 그 페이지 데이터에 접근하는 것을 시도하고 자세하게 결과를 출력한다. 만약 그 시도한 것이 실패되면 PDE, PTE 주소 정보들도 나타낸다. 주소 공간의 문제 되는 "공간들"에서 도움이 될 수 있는 진단보고적인 플러그인이다.

이 플러그인은 기본적으로 지원되지 않고, contrib 디렉터리안에 위치하고 있으며 non-PAE x86 주소 공간들에서 작동이 된다.

```
$ python vol.py --plugins=contrib/plugins/ -f pat-2009-11-16.mddramimage pagecheck
Volatile Systems Volatility Framework 2.1_rc1
(V): 0x06a5a000 [PDE] 0x038c3067 [PTE] 0x1fe5e047 (P): 0x1fe5e000 Size: 0x00001000
(V): 0x06c5f000 [PDE] 0x14d62067 [PTE] 0x1fe52047 (P): 0x1fe52000 Size: 0x00001000
(V): 0x06cd5000 [PDE] 0x14d62067 [PTE] 0x1fe6f047 (P): 0x1fe6f000 Size: 0x00001000
(V): 0x06d57000 [PDE] 0x14d62067 [PTE] 0x1fe5c047 (P): 0x1fe5c000 Size: 0x00001000
(V): 0x06e10000 [PDE] 0x14d62067 [PTE] 0x1fe62047 (P): 0x1fe62000 Size: 0x00001000
(V): 0x070e4000 [PDE] 0x1cac7067 [PTE] 0x1fe1e047 (P): 0x1fe1e000 Size: 0x00001000
(V): 0x077a8000 [PDE] 0x1350a067 [PTE] 0x1fe06047 (P): 0x1fe06000 Size: 0x00001000
(V): 0x07a41000 [PDE] 0x05103067 [PTE] 0x1fe05047 (P): 0x1fe05000 Size: 0x00001000
(V): 0x07c05000 [PDE] 0x103f7067 [PTE] 0x1fe30047 (P): 0x1fe30000 Size: 0x00001000
...
```

11. 끝맺음

1 차로 Volatility 2.1 버전에 포함되어 있는 Command 문서를 번역하였다. 현재 2.2 가 릴리즈 되었으며 설치 단계부터 응용된 블로그 내용까지 번역을 할 계획이다. 이 문서는 큰 카테고리가 번역이 완료되는 대로 **버전업을 하여 배포**하도록 하겠다.

www.boanproject.com