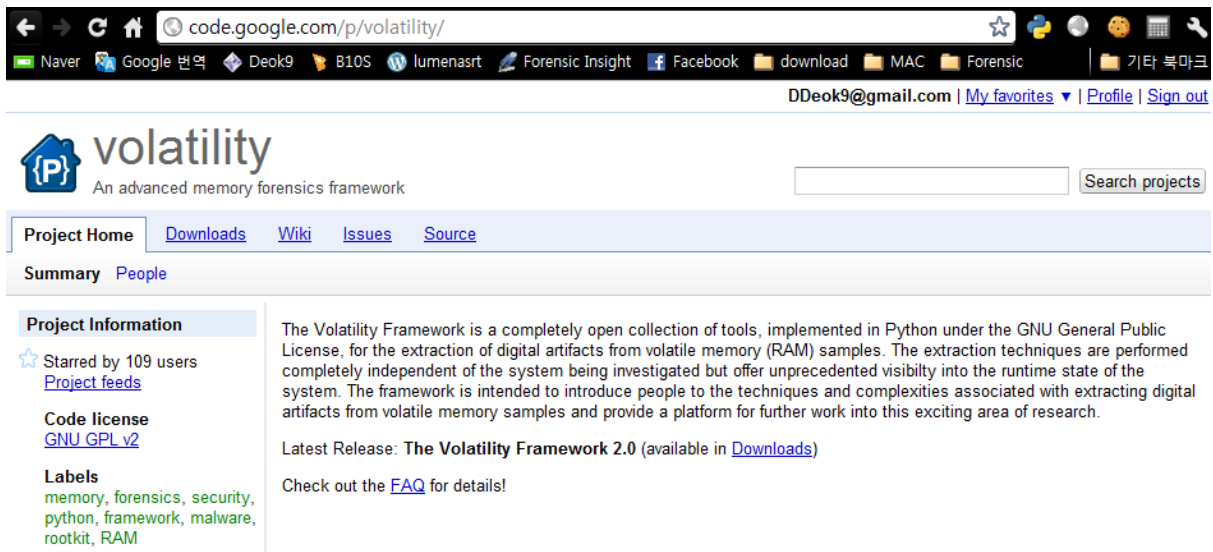


루멘소프트 보안기술연구팀

Windows Memory Dump Analysis

Volatility Plugin 을 이용한 Windows Memory Dump 분석

Volatility



code.google.com/p/volatility/

Naver Google 번역 Deok9 B10S lumenasrt Forensic Insight Facebook download MAC Forensic 기타 북마크

DDeok9@gmail.com | My favorites | Profile | Sign out

volatility

An advanced memory forensics framework

Project Home Downloads Wiki Issues Source

Summary People

Project Information

★ Starred by 109 users
[Project feeds](#)

Code license
[GNU GPL v2](#)

Labels
memory, forensics, security, python, framework, malware, rootkit, RAM

The Volatility Framework is a completely open collection of tools, implemented in Python under the GNU General Public License, for the extraction of digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer unprecedented visibility into the runtime state of the system. The framework is intended to introduce people to the techniques and complexities associated with extracting digital artifacts from volatile memory samples and provide a platform for further work into this exciting area of research.

Latest Release: **The Volatility Framework 2.0** (available in [Downloads](#))

Check out the [FAQ](#) for details!

[code.google.com/p/Volatility]

Open source 이며, Plugin 형태로 다양한 기능들을 제공하고 있는 Python 기반 Windows Memory Forensic Tool 이다. Tool에서 제공하고 있는 Option을 통해서 Windows Memory로 부터 어떠한 정보들을 추출할 수 있는지에 대해 파악 할 수 있을 정도로 완성도가 높다.

현재 Version은 2.0 이며, SVN을 통해 Down 받으면 2.1 Version을 받을 수 있다. 1.3 Version까지는 Windows XP까지만 지원이 되며, 2.0 Version은 모든 Windows Version을 지원한다.

Volatility에서 지원하는 Plugin들은 아래와 같다.

Plugins

Image Identification :

imageinfo, kdbgscan, kprcscan

Processes and DLLs :

pslist, pstree, psscan, dlllist, dlldump, handles, getsids, verinfo, enumfunc

Process Memory :

memmap, memdump, procmemdump, procexedump, vadwalk, vadtrees, vadinfo, vaddump

Kernel Memory and Objects :

modules, modscan, moddump, ssdt, driverscan, filesan, mutantscan, symlinksan, thrdsan

Networking :

connections, connscan, Sockets, sockscan, netscan

Registry :

hivescan, hivelist, printkey, hivedump, hashdump, lsadump, userassist

Crash Dumps, Hibernation, and Conversion :

crashinfo, , hibinfo, imagecopy

Malware and Rootkits :

malfind, , svcsan, ldrmodules, impscan, apihooks, idt, gdt, threads, callbacks, driverirp, devicetree, psxview, ssdt_ex, timers

Miscellaneous :

strings, volshell, bioskbd

Image Identification

imageinfo

```
$ python vol.py -f win7.dmp imageinfo
Volatile Systems Volatility Framework 2.0
Determining profile based on KDBG search...
Suggested Profile : Win7SP1x86, Win7SP0x86
AS Layer1 : JKIA32PagedMemory (Kernel AS)
AS Layer2 : FileAddressSpace (/Users/M/Desktop/win7.dmp)
PAE type : No PAE
DTB : 0x185000
KDBG : 0x8296cbe8
KPCR : 0x8296dc00
KUSER_SHARED_DATA : 0xffdf0000
Image date and time : 2010-07-06 22:40:28
Image local date and time : 2010-07-06 22:40:28
Number of Processors : 2
Image Type :
```

Image의 System 정보를 모를 시에 사용하는 명령어이며 Suggested Profile Field 결과값은 Type Field의 결과값을 통해 조금 더 정확한 Service Pack 정보까지 알 수 있다. 추후 --profile의 인자로 전달하여 Image 분석 시에 사용하게 된다.

KDBG, KPCR(KDDEBUGGER_DATA64) 주소 값도 보여주기 때문에 --kpcr, --kdbg를 통해 다른 Volatility 명령어와 사용할 정보를 제공해 준다.

kdbgscan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp kdbgscan
Volatile Systems Volatility Framework 2.0
Potential KDBG structure addresses (P = Physical, V = Virtual):
_KDBG: V 0x80544ce0 (WinXPSP3x86)
_KDBG: P 0x00544ce0 (WinXPSP3x86)
_KDBG: V 0x80544ce0 (WinXPSP2x86)
_KDBG: P 0x00544ce0 (WinXPSP2x86)
```

가능성 있는 KDBG 구조체의 주소를 보여주는 명령어이며 자세한 KDBG 구조는 아래 두 주소에서 확인할 수 있다.

<http://moyix.blogspot.com/2008/04/finding-kernel-global-variables-in.html>

<http://gleeda.blogspot.com/2010/12/identifying-memory-images.html>

kpcrscan

```
$ python vol.py --profile=win7SP0x86 -f mem.dmp kpcrscan
Volatile Systems Volatility Framework 2.1.alpha
Potential KPCR structure virtual addresses:
_KPCR: 0x807c3000
_KPCR: 0x8296fc00
_KPCR: 0x8cd00000
_KPCR: 0x8cd36000
```

가능성 있는 KPCR 구조체의 주소를 보여주는 명령어이며, Multi-core System에서는 각각의 Processor가 자신의 KPCR을 가지고 있으므로 Image에서 하나 이상의 KPCR주소를 확인할 수 있을 것이다.

Processor의 KPCR을 참조로 하는 idt와 timers와 같은 Plugin은 우선 kpcrscan을 수행하고, 해당 결과값을 --kpcr의 인자로 전달해 주어야 한다.

KPCR 구조는 아래의 주소에서 확인할 수 있다.

<http://blog.schatzforensic.com.au/2010/07/finding-object-roots-in-vista-kpcr/>

[+] KPCR

Kernel Processor Control Region 이라는 뜻으로, Windows Kernel 에서 각각의 Processor 에 대한 정보를 저장하는 Data 구조체 이다.

해당 구조체는 Windows XP, 2003, Vista 에서 Virtual address 0xffdfxxx 에 위치하고 있으며, Opcode(Nickname)가 XP 의 Reserved2 Field 가 2000 에서 kdVersionBlock 으로 변경된 것을 확인 하였다. 그 후, 연구를 통해 이 필드가 _DBGKD_GET_VERSION64 구조체에서 가리켜 지는 것을 확인 하였고, 이는 _KDDEBUGGER_DATA64 구조체의 Linked list 에 포함된 것으로 확인되었다.

[+] _KDDEBUGGER_DATA64

Kernel Debugger 가 해당 OS 의 상태를 쉽게 찾기 위해 사용하는 구조체 이다. 어떠한 Process 가 활성화 상태인지, 어떠한 Handle 이 열려 있는지, Kernel 변수의 Memory 주소 등의 정보를 가지고 있다

[+] Debugger Data Block

```
typedef struct _DBGKD_DEBUG_DATA_HEADER64 {  
    LIST_ENTRY64 List;  
    ULONG          OwnerTag;  
    ULONG          Size;  
} DBGKD_DEBUG_DATA_HEADER64, *PDBGKD_DEBUG_DATA_HEADER64;
```

OwnerTag : KDBG

Size : 각 OS 마다 다름

| | |
|---------|-------|
| - XP | 0x290 |
| - 2000 | 0x208 |
| - W2K3 | 0x318 |
| - Vista | 0x328 |
| - W2K8 | 0x330 |
| - 7 | 0x340 |

[+] "KDBG"

Debugger data block 의 Header 에 포함된 Signature 이며, 이를 통해 OS 정보를 식별 가능하다. 참고로 32bit 의 경우 8Byte 의 0x00 이 존재하고, 64bit 의 경우 0xfffff800 값을 가진 후에 KDBG Signature 가 존재한다.

Processes and DLLs

pslist

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp pslist
Volatile Systems Volatility Framework 2.0
Offset(V)  Name          PID  PPID  Thds  Hnds  Time
-----
0x84133a30 System        4     0    88   486  2010-06-16 15:24:58
0x852e7020 smss.exe     252   4     2    29  2010-06-16 15:24:58
0x859f3d40 csrss.exe    352  316     9   406  2010-06-16 15:25:12
0x85a5a530 wininit.exe  392  316     3    75  2010-06-16 15:25:15
0x85a5f530 csrss.exe    400  384    10   361  2010-06-16 15:25:15
0x859f5bc0 winlogon.exe 464  384     3   112  2010-06-16 15:25:18
0x85b0b318 services.exe 508  392     6   185  2010-06-16 15:25:18
0x85d393f8 lsass.exe    516  392     6   584  2010-06-16 15:25:18
0x841d1750 lsm.exe      524  392    10   143  2010-06-16 15:25:18
0x85d5b8f8 svchost.exe 628  508     9   361  2010-06-16 15:25:19
0x850c67e0 svchost.exe 688  508     7   268  2010-06-16 15:25:20
[snip]
```

System에서 사용하고 있는 Process list를 보여주는 명령어이며, PsActiveProcessHead를 통해 doubly-linked list 형태로 탐색을 한다. 해당 명령어는 hidden/unlinked Process는 감지를 못하는 단점이 있고, Time은 Process의 시작 시간을 의미한다.

또한 만약 결과물에 Thds와 Hnds가 0이면 해당 Process는 활성화 상태가 아님을 뜻한다. 자세한 정보는 아래의 주소에서 확인할 수 있다.

<http://mnin.blogspot.com/2011/03/mis-leading-active-in.html>

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp pslist -P
Volatile Systems Volatility Framework 2.0
Offset(P)  Name          PID  PPID  Thds  Hnds  Time
-----
0x3fff3a30 System        4     0    88   486  2010-06-16 15:24:58
0x3ece7020 smss.exe     252   4     2    29  2010-06-16 15:24:58
0x3e7f3d40 csrss.exe    352  316     9   406  2010-06-16 15:25:12
0x3e45a530 wininit.exe  392  316     3    75  2010-06-16 15:25:15
0x3e45f530 csrss.exe    400  384    10   361  2010-06-16 15:25:15
0x3e7f5bc0 winlogon.exe 464  384     3   112  2010-06-16 15:25:18
0x3e50b318 services.exe 508  392     6   185  2010-06-16 15:25:18
0x3e393f8  lsass.exe    516  392     6   584  2010-06-16 15:25:18
0x3eff1750 lsm.exe      524  392    10   143  2010-06-16 15:25:18
0x3e35b8f8 svchost.exe 628  508     9   361  2010-06-16 15:25:19
0x3eec67e0 svchost.exe 688  508     7   268  2010-06-16 15:25:20
[snip]
```

Offset은 Virtual address를 기본으로 보여주고 있기 때문에, Physical 주소를 보고 싶다면 -P Option을 주면 가능하다.

pstree

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp pstree
Volatile Systems Volatility Framework 2.0
Name          Pid  PPid  Thds  Hnds  Time
-----
0x84E6E3D8:wininit.exe      384   340     3    73  2010-07-06 22:28:53
.. 0x8D4CC030:services.exe   492   384    12   216  2010-07-06 22:28:54
.. 0x84E19030:svchost.exe    1920  492     8   115  2010-07-06 22:33:17
.. 0x8D4E5BB0:schtasks.exe   2512  492     2    60  2010-07-06 22:39:09
.. 0x8D7E9030:wsqmqns.exe    2576  492     1     3  2010-07-06 22:39:11
.. 0x8D5B18A8:dllhost.exe    1944  492    16   187  2010-07-06 22:31:21
.. 0x8D7EE030:taskhost.exe   1156  492    10   155  2010-07-06 22:37:54
.. 0x84D79D40:msdtc.exe       284   492    15   152  2010-07-06 22:31:24
.. 0x8D6781D8:svchost.exe    1056  492    16   589  2010-07-06 22:29:31
.. 0x8D777D40:taskhost.exe   2520  492    11   224  2010-07-06 22:39:10
.. 0x8D759470:sdclt.exe      2504  492     1     4  2010-07-06 22:39:09
.. 0x8D5574D8:rundll32.exe   2484  492     1     5  2010-07-06 22:39:08
.. 0x84D82C08:SearchIndexer. 1464  492    18   624  2010-07-06 22:33:20
... 0x8D759760:SearchFilterHo 1724 1464     6    82  2010-07-06 22:37:36
... 0x8D55E678:SearchProtocol 2680 1464     8   231  2010-07-06 22:39:27
.. 0x8D5CC030:svchost.exe    1140  492    17   375  2010-07-06 22:29:51
[snip]
```

pslist와 비슷한 기능을 하지만 hidden/unlinked Process 까지도 확인할 수 있다는 장점이 있으며, 그림을 통해 볼 수 있듯이 자식 Process 는 "." 을 통해 Tree 형태로 보여준다.

psscan

```
$ 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  SearchProtocolHost. 2448   1168    0x3ecf15c0 2010-06-16 23:30:52
0x3eb10030  SearchFilterHost.    1812   1168    0x3ecf1480 2010-06-16 23:31:02
[snip]
```

pool tag scanning 을 통해 Process 들을 보여주며, 이미 종료된(interactive) Process 와 Rootkit 에 의한 hidden/unlinked Process 들도 확인할 수 있다.

Process 가 이미 종료 되었다면 Time exited Field 에 시간이 남게 된다.

dlllist

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlllist
[snip]
*****
services.exe pid: 492
Command line : C:\Windows\system32\services.exe

Base      Size      Path
-----
0x00a50000 0x041000 C:\Windows\system32\services.exe
0x778a0000 0x13c000 C:\Windows\SYSTEM32\ntdll.dll
0x779f0000 0x0d4000 C:\Windows\system32\kernel32.dll
0x75ca0000 0x04a000 C:\Windows\system32\KERNELBASE.dll
0x75e40000 0x0ac000 C:\Windows\system32\msvcrt.dll
0x76650000 0x0a1000 C:\Windows\system32\RPCRT4.dll
0x758d0000 0x01a000 C:\Windows\system32\SspiCli.dll
0x759f0000 0x00b000 C:\Windows\system32\profapi.dll
0x75d80000 0x019000 C:\Windows\SYSTEM32\sechost.dll
0x75940000 0x00c000 C:\Windows\system32\CRYPTBASE.dll
0x758c0000 0x00f000 C:\Windows\system32\sxcext.dll
0x764a0000 0x0c9000 C:\Windows\system32\USER32.dll
0x765b0000 0x04e000 C:\Windows\system32\GDI32.dll
0x76330000 0x00a000 C:\Windows\system32\LPK.dll
[snip]
```

Image에서 Dll list를 보여주는 명령어이며, PEB의 InLoadOrderModuleList가 가리키는 LDR_DATA_TABLE_ENTRY를 통해 doubly-linked list 형태로 탐색을 한다. Dll들은 Process 가 LoadLibrary(또는 LdrLoadDll) 함수를 호출하면 자동으로 추가되며 FreeLibrary 함수가 호출되거나 reference count가 0이 되기 전까지는 삭제되지 않는다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlllist --pid=492
```

특정 Process에 대한 Dll list를 보려면 -p 또는 --pid Option을 사용하여 확인 가능하다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlllist --offset=0x04a291a8
```

Rootkit에 의한 hidden/unlinked Process의 Dll을 보려면 psscan 후 얻게 되는 physical 주소를 --offset Option과 함께 사용하면 확인 가능하다.

dlldump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlldump -r kernel32 -D out
Cannot dump TrustedInstall@kernel32.dll at 779f0000
Cannot dump WmiPrvSE.exe@kernel32.dll at 779f0000
Dumping kernel32.dll, Process: SearchFilterHo, Base: 779f0000 output: module.623.da1d760.779f0000.dll
Dumping kernel32.dll, Process: taskhost.exe, Base: 779f0000 output: module.484.546d030.779f0000.dll
Cannot dump dwm.exe@kernel32.dll at 779f0000
Dumping kernel32.dll, Process: explorer.exe, Base: 779f0000 output: module.758.4a291a8.779f0000.dll
Cannot dump wuauclt.exe@kernel32.dll at 779f0000
Dumping kernel32.dll, Process: VMwareTray.exe, Base: 779f0000 output: module.860.fe828d8.779f0000.dll
[snip]
```

Memory의 Process로 부터 DLL 추출 및 분석을 위해 dump를 수행하는 명령어 이다. 모든 PE File dump는 --base Option과 함께 사용하며, 이는 hidden DLL 탐지에 유용하다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlldump --pid=492 -D out --base=0x00680000
```

특정 Process에 해당하는 DLL dump는 -p 또는 --pid Option을 사용하여 확인 가능하다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp dlldump -o 0x3e3f64e8 -D out --base=0x00680000
```

hidden/unlinked Process에 해당하는 DLL dump는 -o 또는 --offset Option을 사용하여 확인 가능하다.

handles

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp handles
Volatile Systems Volatility Framework 1.4_rc1
Offset(V)  Pid  Type  Details
0x823c8818 4    Process System(4)
0x823c7008 4    Thread TID 12 PID 4
0xe13f8fa0 4    Key    MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMET
0xe100f418 4    Key    MACHINE\SYSTEM\WPA\MEDIACENTER
0xe1406418 4    Key    MACHINE\SYSTEM\WPA\KEY-4F3B2RFXKC9C637882MBM
0xe1404b20 4    Key    MACHINE\SYSTEM\WPA\PNP
0xe1000480 4    Key    MACHINE\SYSTEM\WPA\SIGNINGHASH-V44KQMCFXKQCTQ
0xe1013b80 4    Key    MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0xe142cc28 4    Key    MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG
0x823c1778 4    Event  'TRKWS_EVENT'
0x81ebb938 4    File   '\\Documents and Settings\\NetworkService\\NTUSER.DAT'
0xe14dc268 4    Key    MACHINE\HARDWARE\DEVICEMAP\SCSI
[snip]
```

Image로 부터 열려 있는 handle을 보여주는 명령어 이며, CreateFile과 같은 함수를 통한 File handle을 포함할 수 있다. CloseHandle과 같은 함수가 호출 되기 전까지는 계속 유효한 것들이며, registry keys, mutexex, named pipes, events, windows stations, desktops, threads 등의 모든 object들을 보여준다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp handles -p 600 -t Process
Volatile Systems Volatility Framework 1.4_rc1
Offset(V)  Pid  Type  Details
0x81da5638 600  Process winlogon.exe(624)
0x82073008 600  Process services.exe(668)
0x81e6b648 600  Process VMwareUser.exe(1356)
0x81e70008 600  Process lsass.exe(680)
0x823315c0 600  Process vmacthlp.exe(844)
0x81db8d88 600  Process svchost.exe(856)
[snip]
```

특정 Process에 대한 handle만을 출력하고 싶다면, -p 또는 --pid 또는 --physical-offset Option을 사용한다.

특정 type에 대한 handle만을 출력하고 싶다면, -t 또는 --object-type Option을 사용한다.

결과에 object 이름이 없어 공백으로 나오는 경우를 없애고 싶다면 --silent Option을 사용하면 좀 더 의미 있는 결과들만 확인할 수 있다.

getsids

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp getsids
Volatile Systems Volatility Framework 2.0
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 (252): S-1-5-18 (Local System)
smss.exe (252): S-1-5-32-544 (Administrators)
smss.exe (252): S-1-1-0 (Everyone)
smss.exe (252): S-1-5-11 (Authenticated Users)
smss.exe (252): S-1-16-16384 (System Mandatory Level)
[snip]
```

Process 와 관련된 Security ID 를 보여주는 명령어 이며, 권한 상승과 같은 의심 가는 부분을 확인할 수 있다.

verinfo

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp verinfo
[snip]

C:\Windows\system32\CRYPTBASE.dll
C:\Windows\system32\winlogon.exe
File version : 6.1.7600.16447
Product version : 6.1.7600.16447
Flags :
OS : Windows NT
File Type : Application
File Date :
CompanyName : Microsoft Corporation
FileDescription : Windows Logon Application
FileVersion : 6.1.7600.16447 (win7_gdr.091027-1503)
InternalName : winlogon
LegalCopyright : \xa9 Microsoft Corporation. All rights reserved.
OriginalFilename : WINLOGON.EXE
ProductName : Microsoft\xae Windows\xae Operating System
ProductVersion : 6.1.7600.16447

[snip]

C:\Windows\System32\ntlanman.dll
File version : 6.1.7600.16385
Product version : 6.1.7600.16385
Flags :
OS : Windows NT
File Type : Dynamic Link Library
File Date :
CompanyName : Microsoft Corporation
FileDescription : Microsoft\xae Lan Manager
FileVersion : 6.1.7600.16385 (win7_rtm.090713-1255)
InternalName : ntlanman.dll
LegalCopyright : \xa9 Microsoft Corporation. All rights reserved.
OriginalFilename : ntlanman.dll
ProductName : Microsoft\xae Windows\xae Operating System
ProductVersion : 6.1.7600.16385

[snip]
```

PE File 에 저장된 정보들을 보여주는 명령어 이며, 모든 PE File 들이 version 정보를 가지지는 않지만, 식별 및 범위 축소에 도움을 주는 도구라 할 수 있다.
해당 명령어는 Process ID, 정규 표현 식, EPROCESS offset Option 을 지원하고 있다.

enumfunc

```
$ python vol.py --plugins=contrib/plugins --profile=Win7SP0x86 -f win7.dmp enumfunc -P -E
Process      Type      Module      Ordinal      Address      Name
winlogon.exe Export    ntdll.dll    18            0x00000000778b657b A_SHAFinal
winlogon.exe Export    ntdll.dll    19            0x00000000778b6392 A_SHAInit
winlogon.exe Export    ntdll.dll    20            0x00000000778b63e8 A_SHAUpdate
winlogon.exe Export    ntdll.dll    21            0x00000000779185fe AlpcAdjustCompletionListConcurrencyCount
winlogon.exe Export    ntdll.dll    22            0x0000000077918b8e AlpcFreeCompletionListMessage
winlogon.exe Export    ntdll.dll    23            0x000000007794858d AlpcGetCompletionListLastMessageInformat
winlogon.exe Export    ntdll.dll    24            0x0000000077948559 AlpcGetCompletionListMessageAttributes
winlogon.exe Export    ntdll.dll    25            0x00000000778fb0ff AlpcGetHeaderSize
[snip]
```

Process 들에서 import/export 된 함수를 추출하여 보여주는 명령어 이다. 결과가 매우 장황하기 때문에 Option 을 추가하여 사용하여야 한다.

Processes Memory

memmap

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 memmap
explorer.exe pid: 1880
Virtual Physical Size
0x0000010000 0x00075cb000 0x000000001000
0x0000021000 0x0009c2c000 0x000000001000
0x0000030000 0x0002adf000 0x000000001000
0x0000031000 0x0000d99000 0x000000001000
0x0000032000 0x000583a000 0x000000001000
0x0000040000 0x000a25b000 0x000000001000
0x0000041000 0x00044d6000 0x000000001000
0x0000050000 0x00099ee000 0x000000001000
0x0000060000 0x000b155000 0x000000001000
[snip]
```

해당 Image에서 할당된 Memory 주소 Map을 보여주는 명령어 이다.

memdump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 memdump -D memory/
Volatile Systems Volatility Framework 2.0
*****
Writing explorer.exe [ 1880] to 1880.dmp

$ ls -alh memory/1880.dmp
-rw-r--r-- 1 User staff 140M Feb 8 15:13 memory/1880.dmp
```

Process의 다양한 Memory Segment에서 모든 Data를 추출하여 dump하는 명령어 이다.

procmemdump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 procmemdump -D memory/
Volatile Systems Volatility Framework 2.0
*****
Dumping explorer.exe, pid: 1880 output: executable.1880.exe

$ file memory/executable.1880.exe
memory/executable.1880.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
```

실행 가능한 Process(Slack space 포함)을 dump 하는 명령어 이다. -u 또는 --unsafe를 사용하면 PE header 검증을 하지 않은 채로 dump가 가능하며, 이는 악성코드가 의도적으로 PE header의 size 필드를 위조 하기 때문에 header를 검증하는 memory dump tool 이 실패하는 경우를 고려하여 만들어 진 것이다.

조금 더 상세한 정보는 아래의 주소에서 확인 가능하다.

<http://computer.forensikblog.de/en/2006/04/reconstructing-a-binary-1.html#more>

추가적으로 import 함수의 rebuild 의 경우 impscan Plugin 을 통하여 도움 받을 수 있다.

procexedump

위의 procmemdump 에서 Slack space 를 포함하지 않고 실행 가능한 Process 를 dump 하는 명령어 이다.

vadwalk

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 vadwalk
Volatile Systems Volatility Framework 2.0
*****
Pid: 1880
Address Parent Left Right Start End Tag Flags
8d5487b8 00000000 8d6a4a20 8d7d5ef8 6ce80000 6ceeefff Vad
8d6a4a20 00000000 8d57ed70 84e4e7f8 02e30000 02e31fff VadS
8d57ed70 8d6a4a20 8d6cfff8 8d7c5c20 01c90000 01e8ffff Vadm
8d6cfff8 00000000 8d760e58 8d457268 00a20000 00ca0fff Vadm
8d760e58 00000000 84d529c0 84d2a1b8 00090000 000cffff VadS
84d529c0 00000000 84e689a0 8d782428 00040000 00041fff Vad
84e689a0 84d529c0 84d52708 83efff78 00020000 00021fff Vad
84d52708 84e689a0 00000000 00000000 00010000 0001ffff Vad
83efff78 84e689a0 00000000 00000000 00030000 00033fff Vad
[snip]
```

VAD(Virtual address Descriptor) nodes 들을 보여주는 명령어 이며, 조금 더 자세한 정보는 아래의 주소를 통해 확인 가능하다.

<http://www.dfrws.org/2007/proceedings/p62-dolan-gavitt.pdf>

vadtree

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 vadtree
Volatile Systems Volatility Framework 2.0
*****
Pid: 1880
6ce80000 - 6ceeefff
02e30000 - 02e31fff
01c90000 - 01e8ffff
00a20000 - 00ca0fff
00090000 - 000cffff
00040000 - 00041fff
00020000 - 00021fff
00010000 - 0001ffff
```

VAD nodes 들을 Tree 형태로 보여주는 명령어 이며, 추가적으로 Graphviz format 으로 출력하여 조금 더 가시적으로 보고 싶다면 --output 을 dot 으로 지정해 주면 된다.

vadinfo

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp -p 1880 vadinfo

VAD node @8d570798 Start 05330000 End 0536ffff Tag VadS
Flags: PrivateMemory
Commit Charge: 18 Protection: 4

VAD node @8d6d78a0 Start 05850000 End 0588ffff Tag VadS
Flags: PrivateMemory
Commit Charge: 16 Protection: 4

VAD node @84f40530 Start 6c750000 End 6c756fff Tag Vad
Flags: UserPhysicalPages
Commit Charge: 2 Protection: 7
ControlArea @84d501e8 Segment 99e28118
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences: 0 NumberOfPfnReferences: 1
NumberOfMappedViews: 1 NumberOfUserReferences: 1
WaitingForDeletion Event: 00000000
Flags: File, Image
FileObject @84e8e910 FileBuffer @ 9c52c7b0 Name: \Windows\System32\msiltsfcg.dll
First prototype PTE: 99e28144 Last contiguous PTE: ffffffff
Flags2: Inherit
File offset: 00000000

[snip]
```

Virtual address 의 시작/끝, Tag, memory map File 이름(존재 시), Kernel Memory 의 MMVAD 구조체 주소, Memory Protection(only original protection)을 보여주는 명령어 이다.

예를 들어 PAGE_NOACCESS 로 Memory 할당 시에 보호를 하였더라도, 후에 VirtualAlloc 이 PAGE_READWRITE 로 MEM_COMMIT 를 시킬 수도 있기 때문에 명령어 수행 결과에 PAGE_NOACCESS 가 나온다고 하더라도, 이 것이 현재 Memory 상태를 나타내는 것이 아닐 수도 있다.

vaddump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp vaddump -D vads
Volatile Systems Volatility Framework 2.0
Pid: 4
*****
Pid: 252
*****
Pid: 348
*****
Pid: 384
*****
Pid: 396
*****
Pid: 424
[snip]

$ ls -alh vads/
-rw-r--r-- 1 User staff 128K Feb 8 15:29 System.a2d960.00120000-0013ffff.dmp
-rw-r--r-- 1 User staff 128K Feb 8 15:29 System.a2d960.00140000-0015ffff.dmp
-rw-r--r-- 1 User staff 128K Feb 8 15:29 System.a2d960.00160000-0017ffff.dmp
-rw-r--r-- 1 User staff 128K Feb 8 15:29 System.a2d960.00180000-0019ffff.dmp
-rw-r--r-- 1 User staff 1.2M Feb 8 15:29 System.a2d960.778a0000-779dbfff.dmp
-rw-r--r-- 1 User staff 1.0M Feb 8 15:29 csrss.exe.3164d40.00000000-000fffff.dmp
-rw-r--r-- 1 User staff 412K Feb 8 15:29 csrss.exe.3164d40.00100000-00166fff.dmp
-rw-r--r-- 1 User staff 4.0K Feb 8 15:29 csrss.exe.3164d40.00170000-00170fff.dmp
-rw-r--r-- 1 User staff 8.0K Feb 8 15:29 csrss.exe.3164d40.00180000-00181fff.dmp
-rw-r--r-- 1 User staff 4.0K Feb 8 15:29 csrss.exe.3164d40.00190000-00190fff.dmp
[snip]
```

memdump 와 유사한 명령어로서, 각각의 VAD Segment 에서 포함된 Data 를 dump 하는 명령어 이다.

memdump 와는 달리 dump 되는 dmp File 의 이름이 Memory 주소 영역으로 되어 있다.

해당 File 명은 ProcessName.PhysicalOffset.StartingVPN.EndingVPN.dmp 와 같은 포맷으로 되어 있다.

PhysicalOffset 이 File 명에 들어가는 이유는 같은 이름을 가진 2 개 이상의 Process 들을 구분하기 위함이다.

Processes and DLLs

modules

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp modules
Volatile Systems Volatility Framework 2.0
Offset(V)  File                                                    Base          Size      Name
0x84131c98 \SystemRoot\system32\ntkrnlpa.exe                     0x008283c000 0x410000   ntoskrnl.exe
0x84131c20 \SystemRoot\system32\halmacpi.dll                     0x0082805000 0x037000   hal.dll
0x84131ba0 \SystemRoot\system32\kdcom.dll                         0x0080b99000 0x008000   kdcom.dll
0x84131b20 \SystemRoot\system32\mcupdate_GenuineIntel.dll      0x0082e24000 0x078000   mcupdate.dll
0x84131aa0 \SystemRoot\system32\PSHED.dll                         0x0082e9c000 0x011000   PSHED.dll
0x84131a20 \SystemRoot\system32\BOOTVID.dll                       0x0082ead000 0x008000   BOOTVID.dll
0x841319a8 \SystemRoot\system32\CLFS.SYS                          0x0082eb5000 0x042000   CLFS.SYS
0x84131930 \SystemRoot\system32\CI.dll                          0x0082ef7000 0x0ab000   CI.dll
0x841318b0 \SystemRoot\system32\drivers\Wdf01000.sys             0x0086a1b000 0x071000   Wdf01000.sys
0x84131830 \SystemRoot\system32\drivers\WDFLDR.SYS              0x0086a8c000 0x00e000   WDFLDR.SYS
0x841317b8 \SystemRoot\system32\DRIVERS\ACPI.sys                  0x0086a9a000 0x048000   ACPI.sys
0x84131738 \SystemRoot\system32\DRIVERS\WMILIB.SYS              0x0086ae2000 0x009000   WMILIB.SYS
0x841316b8 \SystemRoot\system32\DRIVERS\msisadrv.sys       0x0086aeb000 0x008000   msisadrv.sys
0x8412be78 \SystemRoot\system32\DRIVERS\pci.sys            0x0086af3000 0x02a000   pci.sys
[snip]
```

System에 load된 kernel driver들을 보여주는 명령어 이다. PSLoadedModuleList가 가리키는 LDR_DATA_TABLE_ENTRY를 통해 doubly-linked 형태로 동작하며 hidden/unlinked kernel driver는 출력해 주지 않는다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp modules -P
Volatile Systems Volatility Framework 2.0
Offset(P)  File                                                    Base          Size      Name
0x3fff1c98 \SystemRoot\system32\ntkrnlpa.exe                     0x008283c000 0x410000   ntoskrnl.exe
0x3fff1c20 \SystemRoot\system32\halmacpi.dll                     0x0082805000 0x037000   hal.dll
0x3fff1ba0 \SystemRoot\system32\kdcom.dll                         0x0080b99000 0x008000   kdcom.dll
0x3fff1b20 \SystemRoot\system32\mcupdate_GenuineIntel.dll      0x0082e24000 0x078000   mcupdate.dll
0x3fff1aa0 \SystemRoot\system32\PSHED.dll                         0x0082e9c000 0x011000   PSHED.dll
0x3fff1a20 \SystemRoot\system32\BOOTVID.dll                       0x0082ead000 0x008000   BOOTVID.dll
0x3fff19a8 \SystemRoot\system32\CLFS.SYS                          0x0082eb5000 0x042000   CLFS.SYS
0x3fff1930 \SystemRoot\system32\CI.dll                          0x0082ef7000 0x0ab000   CI.dll
0x3fff18b0 \SystemRoot\system32\drivers\Wdf01000.sys             0x0086a1b000 0x071000   Wdf01000.sys
0x3fff1830 \SystemRoot\system32\drivers\WDFLDR.SYS              0x0086a8c000 0x00e000   WDFLDR.SYS
0x3fff17b8 \SystemRoot\system32\DRIVERS\ACPI.sys                  0x0086a9a000 0x048000   ACPI.sys
0x3fff1738 \SystemRoot\system32\DRIVERS\WMILIB.SYS              0x0086ae2000 0x009000   WMILIB.SYS
0x3fff16b8 \SystemRoot\system32\DRIVERS\msisadrv.sys       0x0086aeb000 0x008000   msisadrv.sys
0x3ffebe78 \SystemRoot\system32\DRIVERS\pci.sys            0x0086af3000 0x02a000   pci.sys
[snip]
```

기본적으로 Virtual address 로 출력하며, Physical address 를 보고 싶다면 -P Option 을 추가하면 된다.

modscan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp modscan
Volatile Systems Volatility Framework 2.0
Offset      File                                                    Base          Size      Name
0x3e011200 '\\SystemRoot\\system32\\DRIVERS\\lltdio.sys'      0x008e7b6000 0x010000   'lltdio.sys'
0x3e020a50 '\\SystemRoot\\system32\\DRIVERS\\rspndr.sys'       0x008e7c6000 0x013000   'rspndr.sys'
0x3e03bec0 '\\SystemRoot\\system32\\DRIVERS\\asynccmac.sys'      0x0090b19000 0x009000   'asynccmac.sys'
0x3e067370 '\\?\\c:\\Users\\user\\Desktop\\win32dd.sys'         0x0090a00000 0x00c000   'win32dd.sys'
0x3e0dc7a0 '\\SystemRoot\\system32\\DRIVERS\\srv.sys'              0x0090a5e000 0x051000   'srv.sys'
0x3e364908 '\\SystemRoot\\system32\\drivers\\luafv.sys'             0x008e781000 0x01b000   'luafv.sys'
0x3e366608 '\\SystemRoot\\system32\\drivers\\WudfPF.sys'         0x008e79c000 0x01a000   'WudfPF.sys'
0x3e36b850 '\\SystemRoot\\system32\\drivers\\HTTP.sys'              0x008dc04000 0x085000   'HTTP.sys'
0x3e422648 '\\SystemRoot\\System32\\TSDDD.dll'                       0x008fda0000 0x009000   'TSDDD.dll'
0x3e4291e0 '\\SystemRoot\\system32\\DRIVERS\\mouhid.sys'             0x008e76b000 0x00b000   'mouhid.sys'
0x3e429bd8 '\\SystemRoot\\system32\\DRIVERS\\HIDPARSE.SYS'          0x008e764000 0x007000   'HIDPARSE.SYS'
0x3e46a4d0 '\\SystemRoot\\System32\\cdd.dll'                         0x008fdd0000 0x01e000   'cdd.dll'
0x3e46aed0 '\\SystemRoot\\System32\\drivers\\mpsdrv.sys'             0x008dca2000 0x012000   'mpsdrv.sys'
0x3e474a38 '\\SystemRoot\\system32\\DRIVERS\\bowser.sys'            0x008dc89000 0x019000   'bowser.sys'
0x3e482870 '\\SystemRoot\\system32\\DRIVERS\\mrxsmb20.sys'        0x008dd12000 0x01b000   'mrxsmb20.sys'
0x3e482da0 '\\SystemRoot\\system32\\DRIVERS\\mrxsmb10.sys'      0x008dcd7000 0x03b000   'mrxsmb10.sys'
[snip]
```

Kernel module 을 Physical address 로 보여주는 명령어 이며, 이전에 load 되었던 driver, Rootkit 에 의해 hidden/unlinked 된 Driver 도 출력해 준다.

moddump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp moddump -D mods/
Volatile Systems Volatility Framework 2.0
Dumping ntoskrnl.exe, Base: 8284c000 output: driver.8284c000.sys
Dumping hal.dll, Base: 82815000 output: driver.82815000.sys
Dumping fwpkclnt.sys, Base: 86550000 output: driver.86550000.sys
Dumping kdcom.dll, Base: 80bcc000 output: driver.80bcc000.sys
Dumping NDIS.sys, Base: 8c7ec000 output: driver.8c7ec000.sys
Dumping CLFS.SYS, Base: 85cbe000 output: driver.85cbe000.sys
Dumping luafv.sys, Base: 8840c000 output: driver.8840c000.sys
Dumping peauth.sys, Base: 8857c000 output: driver.8857c000.sys
[snip]
```

Kernel driver 를 File 로 dump 해 주는 명령어 이다. 이는 정규 표현 식과, Physical Offset 을 이용한 Filter 를 지원해 준다. 모든 driver 들을 dump 하고 싶다면 아무런 Filter 를 적용하지 않으면 된다.

아래 주소를 통해 조금 더 상세한 설명을 확인할 수 있다.

<http://moyix.blogspot.com/2008/10/plugin-post-moddump.html>

ssdt

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp ssdt
Volatile Systems Volatility Framework 2.0
Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 828a8634 with 401 entries
Entry 0x0000: 0x82a8530e (NtAcceptConnectPort) owned by ntoskrnl.exe
Entry 0x0001: 0x828d6774 (NtAccessCheck) owned by ntoskrnl.exe
Entry 0x0002: 0x82abd460 (NtAccessCheckAndAuditAlarm) owned by ntoskrnl.exe
Entry 0x0003: 0x82901dea (NtAccessCheckByType) owned by ntoskrnl.exe
Entry 0x0004: 0x82a9f99a (NtAccessCheckByTypeAndAuditAlarm) owned by ntoskrnl.exe
Entry 0x0005: 0x8294145a (NtAccessCheckByTypeResultList) owned by ntoskrnl.exe
[snip]
SSDT[1] at 977a5000 with 825 entries
Entry 0x1000: 0x9772eb34 (NtGdiAbortDoc) owned by win32k.sys
Entry 0x1001: 0x9774752e (NtGdiAbortPath) owned by win32k.sys
Entry 0x1002: 0x975adc1a (NtGdiAddFontResourceW) owned by win32k.sys
Entry 0x1003: 0x9773e5ae (NtGdiAddRemoteFontToDC) owned by win32k.sys
Entry 0x1004: 0x97748c89 (NtGdiAddFontMemResourceEx) owned by win32k.sys
Entry 0x1005: 0x9772f351 (NtGdiRemoveMergeFont) owned by win32k.sys
Entry 0x1006: 0x9772f3e5 (NtGdiAddRemoteMMInstanceToDC) owned by win32k.sys
Entry 0x1007: 0x976545cc (NtGdiAlphaBlend) owned by win32k.sys
[snip]
```

Native/GUI SSDT 의 함수들을 list 시켜주는 명령어 이다. 이는 index, function name, SSDT 안 각 entry 의 driver owner 를 출력해 준다.

Windows 는 4 개의 SSDT 를 가지고 있으나 NT module 의 Native Function, win32k.sys module 의 GUI function 만을 사용하고 있다.

Volatility 는 ETHREAD 객체를 Scan 한 후, 모든 유일한 ETHREAD.Tcb.ServiceTable Pointer 들을 모아서 확인하기 때문에 Rootkit 에 의해 덮어 쓰거나 복사된 SSDT 미탐을 우회할 수 있다. 또한, OS version 에 따라 SSDT 안의 순서, 개수 등이 다르기 때문에 Volatility 는 각 OS 에 대한 정보들을 모두 가지고 있다.

아래 주소를 통해 조금 더 상세한 설명을 확인할 수 있다.

<http://moyix.blogspot.com/2008/08/auditing-system-call-table.html>

추가적으로 System 의 NT module 이 ntkrnlpa.exe 또는 ntkrnlmp.exe 가 될 수 있기 때문에 이를 유의하기 바란다.

driverscan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp driverscan
Volatile Systems Volatility Framework 2.0
Phys.Addr. Obj Type #Ptr #Hnd Start Size Service key Name
0x007f1300 0x0000001a 67 0 0x85e2a000 294912 'ACPI' '\\Driver\\ACPI'
0x007f1b30 0x0000001a 3 0 0x85dab000 462848 'wdf01000' '\\Driver\\wdf01000'
0x00c630d8 0x0000001a 3 0 0x88427000 65536 'lltdio' '\\Driver\\lltdio'
0x00cb0108 0x0000001a 3 0 0x88437000 77824 'rspndr' '\\Driver\\rspndr'
0x00dedd38 0x0000001a 4 0 0x8844a000 544768 'HTTP' '\\Driver\\HTTP'
0x05533b88 0x0000001a 3 0 0x88573000 28672 'Parvdm' '\\Driver\\Parvdm'
0x0ad99af8 0x0000001a 5 0 0x8c400000 94208 'usbccgp' '\\Driver\\usbccgp'
0x0bd46650 0x0000001a 3 0 0x8857a000 7680 'VMMEMCTL' '\\Driver\\VMMEMCTL'
0x0c8b5400 0x0000001a 2 0 0x8840c000 110592 'luafl' '\\FileSystem\\luafl'
0x0d747c18 0x0000001a 3 0 0x884cf000 102400 'bowser' '\\FileSystem\\bowser'
0x0d98cc60 0x0000001a 3 0 0x8864b000 323584 'srv2' '\\FileSystem\\srv2'
[snip]
```

Memory의 DRIVER_OBJECT를 Scan하는 명령어이다. 이는 Kernel module을 찾을 수 있는 또 다른 방법이며, 모든 kernel module이 DRIVER_OBJECT와 관련된 것은 아니다.

DRIVER_OBJECT는 28 IRP(Major Function) Table을 가지고 있기 때문에 driverirp 명령어가 기초가 되었다고 한다.

아래의 주소를 통해 조금 더 상세한 설명을 확인할 수 있다.

<http://computer.forensikblog.de/en/2009/04/scanning-for-drivers.html>

filescan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp filescan
Volatile Systems Volatility Framework 2.0
Phys.Addr. Obj Type #Ptr #Hnd Access Name
0x007b1020 0x0000001c 17 0 RW-rwd '\\$Directory'
0x007b1280 0x0000001c 8 1 R--r-d '\\Windows\\System32\\en-US\\gpsvc.dll.mui'
0x00921bb8 0x0000001c 2 0 R--r-d '\\Windows\\System32\\msftedit.dll'
0x00be5950 0x0000001c 1 1 R--rw- '\\Windows\\System32'
0x00be76b8 0x0000001c 7 0 R--r-- '\\Windows\\Fonts\\marlett.ttf'
0x00bf2370 0x0000001c 8 0 R--r-d '\\Windows\\System32\\sscore.dll'
0x00bf2520 0x0000001c 9 1 R--r-d '\\Windows\\System32\\en-US\\sysmain.dll.mui'
0x00c23a68 0x0000001c 3 0 R--r-d '\\Windows\\System32\\wkssvc.dll'
0x00c23db0 0x0000001c 1 1 ----- '\\srvsvc'
0x00c5a910 0x0000001c 17 0 RW-rwd '\\$Directory'
0x00c5ab10 0x0000001c 5 0 R--r-d '\\Windows\\System32\\w32time.dll'
0x00c64228 0x0000001c 16 1 RW-r-- '\\Windows\\System32\\winevt\\Logs\\Microsoft-Windows-GroupPolicy%4Operational.evtx'
0x00c64610 0x0000001c 8 0 R--r-d '\\Windows\\System32\\RpcRtRemote.dll'
0x00c648a0 0x0000001c 6 0 R--r-d '\\Windows\\System32\\ntlanman.dll'
0x00c70c70 0x0000001c 1 1 ----- '\\wkssvc'
0x00ca3530 0x0000001c 4 0 ----- '\\Windows\\System32\\locale.nls'
0x00ca3ea8 0x0000001c 3 0 R--r-d '\\Windows\\System32\\wiapc.dll'
0x00ca4330 0x0000001c 3 0 R--r-d '\\Windows\\System32\\Sens.dll'
0x00ca4b48 0x0000001c 2 0 R--r-d '\\Windows\\System32\\ktmw32.dll'
0x00ca4c00 0x0000001c 6 0 R--r-d '\\Windows\\System32\\schedsvc.dll'
0x00cad020 0x0000001c 1 1 R--r-- '\\Windows\\Registration\\R00000000000006.c1b'
0x00cadc28 0x0000001c 1 1 ----- '\\wkssvc'
0x00cade78 0x0000001c 1 1 ----- '\\wkssvc'
[snip]
```

Memory의 FILE_OBJECT를 Scan하는 명령어이다. 이는 Rootkit이 디스크에 File을 hiding 하였거나, 실제 System의 open handle을 hooking 하였더라도 open File을 찾을 수 있다. FILE_OBJECT의 Physical offset, File name, number of pointers to the object, number of handles to the object, effective permissions granted to the object를 결과로 출력해 준다.

아래의 주소를 통해 조금 더 상세한 설명을 확인할 수 있다.

<http://computer.forensikblog.de/en/2009/04/scanning-for-file-objects.html>

<http://computer.forensikblog.de/en/2009/04/linking-file-objects-to-processes.html>

mutantscan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp mutantscan -s
Volatile Systems Volatility Framework 2.0
Phys.Addr. Obj Type #Ptr #Hnd Signal Thread CID Name
0x07f955a0 0x0000000e 2 1 1 0x00000000 'TapiSrv_Perf_Library_Lock_PID_5d0'
0x081b5ce8 0x0000000e 2 1 1 0x00000000 'WininetProxyRegistryMutex'
0x08ffcc8 0x0000000e 3 2 1 0x00000000 'ZoneAttributeCacheCounterMutex'
0x099f21f8 0x0000000e 2 1 1 0x00000000 '!MSFTHISTORY!'
0x099f2e58 0x0000000e 2 1 1 0x00000000 'ZonesCacheCounterMutex'
0x09aa83a0 0x0000000e 2 1 1 0x00000000 'ZonesLockedCacheCounterMutex'
0x09aa8bd0 0x0000000e 2 1 1 0x00000000 'ZonesCacheCounterMutex'
0x09d06c18 0x0000000e 2 1 1 0x00000000 'VMwareGuestDnDDataMutex'
0x0afe45d8 0x0000000e 2 1 1 0x00000000 '...?_c:_programdata_microsoft_rac_temp_sql4c79.tmp:x'
0x0b6ea040 0x0000000e 2 1 0 0x83ecd030 2520:2616 'F659A567-8ACB-4E4A-92A7-5C2DD1884F72'
0x0be081e8 0x0000000e 2 1 1 0x00000000 'BITS_Perf_Library_Lock_PID_5d0'
[snip]
```

Memory 에서 KMUTANT 객체를 scan 하는 명령어 이다. 기본적으로 모든 객체들을 보여주며 -s 또는 --silent Option 을 통하여 이름이 있는 mutexes 를 볼 수 있다.

CID column 은 Process ID 와 Thread ID 를 보여준다.

Volatility 는 mutexes 들이 random 한 이름을 가지기 때문에 의심 가는 mutexes 를 판단 하는 것이 어렵다는 것을 알고, sqlite3 database 를 통해 해당 Memory dump 에 이미 DB 에 존재하는 이름이 있으면 강조해주는 기능을 가지고 있다.

symlinkscan

```
$ python vol.py -f win7.dd --profile=win7SP0x86 symlinkscan
Volatile Systems Volatility Framework 2.1_alpha
Offset(P) #Ptr #Hnd CreateTime From To
0x00be60d8 1 0 2010-06-16 15:24:28 Global '\\GLOBAL??'
0x00beabd0 1 0 2010-06-16 15:24:28 DosDevices '\\??'
0x04675030 1 0 2010-06-16 15:27:40 {E2F8A220-AF88-446C-9A55-453E58DD3A33} '\\Device\\NDMP13'
0x05e02fe8 1 0 2010-06-16 19:46:13 PROCEXP113 '\\Device\\PROCEXP113'
0x09700bb0 1 0 2010-06-16 15:25:11 Root#MS_PPTPMINIORT#0000#{cac88484-7515-4c03-82e6-71a87abac361} '\\Device\\00'
0x09700dd8 1 0 2010-06-16 15:25:11 Root#MS_NDISWANIPV6#0000#{cac88484-7515-4c03-82e6-71a87abac361} '\\Device\\00'
0x0a040d28 1 0 2010-06-16 15:25:11 Root#USB#0000#{65a9a6cf-64cd-480b-843e-32c86e1ba19f} '\\Device\\000000043'
0x0e725450 1 0 2010-06-16 15:26:58 Global '\\Global??'
0x0eae33f0 1 0 2010-06-16 15:26:57 Global '\\Global??'
0x15780150 1 0 2010-06-16 15:25:11 Root#SYSTEM#0000#{4747b320-62ce-11cf-a5d6-28db04c10000} '\\Device\\000000042'
0x157806b8 1 0 2010-06-16 15:25:11 Root#SYSTEM#0000#{53172480-4791-11d0-a5d6-28db04c10000} '\\Device\\000000042'
0x15780a90 1 0 2010-06-16 15:25:11 Root#SYSTEM#0000#{cf1dda2c-9743-11d0-a3ee-00a0c9223196} '\\Device\\000000042'
0x15780e50 1 0 2010-06-16 15:25:11 Root#MS_SSTPMINIORT#0000#{cac88484-7515-4c03-82e6-71a87abac361} '\\Device\\00'
0x15ec5820 1 0 2010-06-16 15:25:29 $VDMLPT1 '\\Device\\ParallelVdm0'
0x160c4a38 1 0 2010-06-16 15:25:29 vmememct1 '\\Device\\vmememct1'
0x1655c890 1 0 2010-06-16 15:25:29 MpsDevice '\\Device\\MPS'
[snip]
```

Symbolic link 객체들을 scan 하여 해당 정보를 출력해주는 명령어 이다.

thrdsan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp thrdsan
Volatile Systems Volatility Framework 2.0
Offset PID TID Create Time Exit Time StartAddr
0x0637f030 1344 140 0x778e64d8
0x06389d48 848 852 0x778e64d8
0x064470f8 508 780 0x778e64d8
0x064474b8 508 776 2010-07-06 22:30:03 0x778e64d8
0x06447c10 728 772 0x999f10ac8
0x0658fd48 424 2004 0x778e64d8
0x06a87d48 2520 2552 2010-07-06 22:39:15 0x778e64d8
0x06a902d8 1140 704 0x778e64d8
0x06a908d0 1488 580 0x778e64d8
0x06d3c820 1880 1220 0x778e64d8
0x06e98d48 2312 2316 0x778e64d8
0x06fa7d48 1156 1784 2010-07-06 22:38:01 0x778e64d8
[snip]
```

Memory 의 ETHREAD 객체를 scan 하여 보여주는 명령어 이다. ETHREAD 는 부모 process 를 식별할 수 있는 정보를 가지고 있기 때문에 이는 hidden Process 를 찾는 데 도움을 주기도 한다.

Networking

connections

```
$ python vol.py -f Bob.vmem connections
Volatile Systems Volatility Framework 2.0
Offset(V) Local Address Remote Address Pid
-----
0x81c6a9f0 192.168.0.176:1176 212.150.164.203:80 888
0x82123008 192.168.0.176:1184 193.104.22.71:80 880
0x81cd4270 192.168.0.176:2869 192.168.0.1:30379 1244
0x81cd4270 127.0.0.1:1168 127.0.0.1:1169 888
0x81e41108 127.0.0.1:1169 127.0.0.1:1168 888
0x82108890 192.168.0.176:1178 212.150.164.203:80 1752
0x82210440 192.168.0.176:1185 193.104.22.71:80 880
0x8207ac58 192.168.0.176:1171 66.249.90.104:80 888
0x81cef808 192.168.0.176:2869 192.168.0.1:30380 4
0x81cc57c0 192.168.0.176:1189 192.168.0.1:9393 1244
0x8205a448 192.168.0.176:1172 66.249.91.104:80 888
```

활성화 상태의 Network 연결을 보여주는 명령어 이다. tcpip.sys module 의 non-exported symbol 이 가리키는 connection 구조체를 통해 singly-linked 로 동작하고 있다. 이 명령어는 Windows XP 와 Windows 2003 Server Image 에서만 사용이 가능하다.

```
$ python vol.py -f Bob.vmem connections -P
Volatile Systems Volatility Framework 2.0
Offset(P) Local Address Remote Address Pid
-----
0x01e6a9f0 192.168.0.176:1176 212.150.164.203:80 888
0x02323008 192.168.0.176:1184 193.104.22.71:80 880
0x01ed4270 192.168.0.176:2869 192.168.0.1:30379 1244
0x01ed4270 127.0.0.1:1168 127.0.0.1:1169 888
0x02041108 127.0.0.1:1169 127.0.0.1:1168 888
0x02308890 192.168.0.176:1178 212.150.164.203:80 1752
0x02410440 192.168.0.176:1185 193.104.22.71:80 880
0x0227ac58 192.168.0.176:1171 66.249.90.104:80 888
0x01eef808 192.168.0.176:2869 192.168.0.1:30380 4
0x01ec57c0 192.168.0.176:1189 192.168.0.1:9393 1244
0x0225a448 192.168.0.176:1172 66.249.91.104:80 888
```

기본적으로 출력은 _TCPT_OBJECT 가상 주소를 포함하고 있으며, -P switch 를 이용하여 물리적 주소를 확인 가능하다.

connscan

```
$ python vol.py -f Bob.vmem connscan
Volatile Systems Volatility Framework 2.0
Offset Local Address Remote Address Pid
-----
0x01e6a9f0 192.168.0.176:1176 212.150.164.203:80 888
0x01ec57c0 192.168.0.176:1189 192.168.0.1:9393 1244
0x01ed4270 192.168.0.176:2869 192.168.0.1:30379 1244
0x01eef808 192.168.0.176:2869 192.168.0.1:30380 4
0x01ffa7f8 0.0.0.0:0 80.206.204.129:0 0
0x02041108 127.0.0.1:1168 127.0.0.1:1169 888
0x0225a448 192.168.0.176:1172 66.249.91.104:80 888
0x0226ac58 127.0.0.1:1169 127.0.0.1:1168 888
0x0227ac58 192.168.0.176:1171 66.249.90.104:80 888
0x02308890 192.168.0.176:1178 212.150.164.203:80 1752
0x02323008 192.168.0.176:1184 193.104.22.71:80 880
0x02410440 192.168.0.176:1185 193.104.22.71:80 880
```

Pool tag scanning 을 통하여 connection 구조체를 찾는 명령어 이다. 이미 종료된 연결 정보도 찾을 수 있는 장점이 있다.

그러나, 위 결과에서 볼 수 있듯이 몇몇 Field 에서 부분적으로 덮어쓰기 된 것과 같은 오탐(False Positive)이 존재한다.

이 명령어도 Windows XP 와 Windows 2003 Server Image 에서만 사용이 가능하다.

sockets

```
$ python vol.py -f silentbanker.vmem --profile=WinXPSP3x86 sockets
Volatile Systems Volatility Framework 2.0
Offset(V) PID Port Proto Address Create Time
-----
0x80fd1008 4 0 47 0.0.0.0 2010-08-11 06:08:00
0xff362d18 1088 1066 17 0.0.0.0 2010-08-15 18:54:13
0xff258008 688 500 17 0.0.0.0 2010-08-11 06:06:35
0xff367008 4 445 6 0.0.0.0 2010-08-11 06:06:17
0x80ffc128 936 135 6 0.0.0.0 2010-08-11 06:06:24
0xff225b70 688 0 255 0.0.0.0 2010-08-11 06:06:35
0xff225b70 1028 123 17 127.0.0.1 2010-08-15 19:01:51
0x80fce930 1088 1025 17 0.0.0.0 2010-08-11 06:06:38
0xff127d28 216 1026 6 127.0.0.1 2010-08-11 06:06:39
0xff2608c0 1088 1053 17 0.0.0.0 2010-08-15 18:54:09
0x80fdc708 1884 1051 17 127.0.0.1 2010-08-15 18:54:07
0x80fdc708 1148 1900 17 127.0.0.1 2010-08-15 19:01:51
[snip]
```

TCP, UDP, RAW 등의 protocol 에서 listening Socket 들을 찾아내는 명령어 이다.
tcpip.sys module 의 non-exported symbol 이 가리키는 Socket 구조체를 통해 singly-linked 로 동작하고 있다.
이 명령어는 Windows XP 와 Windows 2003 Server Image 에서만 사용이 가능하다.

```
$ python vol.py -f silentbanker.vmem --profile=WinXPSP3x86 sockets -P
Volatile Systems Volatility Framework 2.0
Offset(P) PID Port Proto Address Create Time
-----
0x01134008 4 0 47 0.0.0.0 2010-08-11 06:08:00
0x04c2dd18 1088 1066 17 0.0.0.0 2010-08-15 18:54:13
0x05f44008 688 500 17 0.0.0.0 2010-08-11 06:06:35
0x04be7008 4 445 6 0.0.0.0 2010-08-11 06:06:17
0x0115f128 936 135 6 0.0.0.0 2010-08-11 06:06:24
0x06237b70 688 0 255 0.0.0.0 2010-08-11 06:06:35
0x06237b70 1028 123 17 127.0.0.1 2010-08-15 19:01:51
0x01131930 1088 1025 17 0.0.0.0 2010-08-11 06:06:38
0x02daad28 216 1026 6 127.0.0.1 2010-08-11 06:06:39
0x05e3c8c0 1088 1053 17 0.0.0.0 2010-08-15 18:54:09
0x0113f708 1884 1051 17 127.0.0.1 2010-08-15 18:54:07
0x0113f708 1148 1900 17 127.0.0.1 2010-08-15 19:01:51
[snip]
```

기본적으로 출력은 _ADDRESS_OBJECT 가상 주소를 포함하고 있으며, -P switch 를 이용하여 물리적 주소를 확인 가능하다.

sockscan

```
$ python vol.py -f silentbanker.vmem --profile=WinXPSP3x86 sockscan
Volatile Systems Volatility Framework 2.0
Offset PID Port Proto Address Create Time
-----
0x00096e08 1884 1069 6 0.0.0.0 2010-08-15 18:54:13
0x0089bab8 1148 1900 17 127.0.0.1 2010-08-15 18:53:56
0x01073910 1884 1077 6 0.0.0.0 2010-08-15 18:54:14
0x01073e98 1884 1082 6 0.0.0.0 2010-08-15 18:54:15
0x0107aba8 1884 1057 6 0.0.0.0 2010-08-15 18:54:10
0x0107c500 1884 1073 6 0.0.0.0 2010-08-15 18:54:13
0x0107db70 1884 1072 6 0.0.0.0 2010-08-15 18:54:13
0x010f1e98 1884 1079 6 0.0.0.0 2010-08-15 18:54:15
0x01120c40 4 445 17 0.0.0.0 2010-08-11 06:06:17
0x0112ee30 1884 1071 6 0.0.0.0 2010-08-15 18:54:13
0x01131930 1088 1025 17 0.0.0.0 2010-08-11 06:06:38
0x01134008 4 0 47 0.0.0.0 2010-08-11 06:08:00
0x01139e98 1884 1087 6 0.0.0.0 2010-08-15 18:54:17
0x0113b7d0 1884 1076 6 0.0.0.0 2010-08-15 18:54:14
0x011568a8 4 137 17 172.16.176.143 2010-08-15 18:53:56
0x0115f128 936 135 6 0.0.0.0 2010-08-11 06:06:24
[snip]
```

Pool tag scanning 을 통하여 Socket 구조체를 찾는 명령어 이다. 잔여 Data 와 이전 Socket 의 정보를 확인 가능하며, 이 명령어도 Windows XP 와 Windows 2003 Server Image 에서만 사용이 가능하다.

netscan

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp netscan
Volatile Systems Volatility Framework 2.0
Offset Proto Local Address Foreign Address State Pid Owner Created
0xca3008 TCPv4 192.168.181.133:139 0.0.0.0:0 LISTENING 4 System 1970-01-01
0x3027008 TCPv4 0.0.0.0:49155 0.0.0.0:0 LISTENING 876 svchost.exe 1970-01-01
0x3027008 TCPv6 :::49155 :::0 LISTENING 876 svchost.exe 1970-01-01
0x5ac5c80 TCPv4 0.0.0.0:49153 0.0.0.0:0 LISTENING 728 svchost.exe 1970-01-01
0x5ac5c80 TCPv6 :::49153 :::0 LISTENING 728 svchost.exe 1970-01-01
0xbfe1208 TCPv4 0.0.0.0:49152 0.0.0.0:0 LISTENING 384 wininit.exe 1970-01-01
0xbfe1208 TCPv6 :::49152 :::0 LISTENING 384 wininit.exe 1970-01-01
0xbfe1648 TCPv4 0.0.0.0:49152 0.0.0.0:0 LISTENING 384 wininit.exe 1970-01-01
0xc1fad48 TCPv4 0.0.0.0:49153 0.0.0.0:0 LISTENING 728 svchost.exe 1970-01-01
0xc5ae148 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 680 svchost.exe 1970-01-01
0xc6f5bb0 TCPv4 0.0.0.0:135 0.0.0.0:0 LISTENING 680 svchost.exe 1970-01-01
0xc6f5bb0 TCPv6 :::135 :::0 LISTENING 680 svchost.exe 1970-01-01
0xd816270 TCPv4 0.0.0.0:445 0.0.0.0:0 LISTENING 4 System 1970-01-01
0xd816270 TCPv6 :::445 :::0 LISTENING 4 System 1970-01-01
0xdc5a368 TCPv4 0.0.0.0:49156 0.0.0.0:0 LISTENING 492 services.exe 1970-01-01
0xde59008 TCPv4 0.0.0.0:49156 0.0.0.0:0 LISTENING 492 services.exe 1970-01-01
0xde59008 TCPv6 :::49156 :::0 LISTENING 492 services.exe 1970-01-01
0xed29808 TCPv4 0.0.0.0:49155 0.0.0.0:0 LISTENING 876 svchost.exe 1970-01-01
0xee49450 TCPv4 0.0.0.0:49154 0.0.0.0:0 LISTENING 500 lsass.exe 1970-01-01
0xee52d98 TCPv4 0.0.0.0:49154 0.0.0.0:0 LISTENING 500 lsass.exe 1970-01-01
0xee52d98 TCPv6 :::49154 :::0 LISTENING 500 lsass.exe 1970-01-01
0x4b5c008 TCPv4 0.0.0.0:49170 65.54.89.134:80 CLOSED 876 svchost.exe 1970-01-01
0x9b3ca30 TCPv4 192.168.181.133:49167 192.168.181.2:80 CLOSED 876 svchost.exe 1970-01-01
0xee8e0c8 TCPv4 0.0.0.0:49159 65.54.89.134:80 CLOSED 876 svchost.exe 1970-01-01
0xf78d468 TCPv4 0.0.0.0:49164 65.54.89.135:80 CLOSED 876 svchost.exe 1970-01-01
0x3165c8 UDPv4 0.0.0.0:0 *:*: 1056 svchost.exe 2010-07-06
0x3165c8 UDPv6 :::0 *:*: 1056 svchost.exe 2010-07-06
0xea9868 UDPv4 127.0.0.1:1900 *:*: 1920 svchost.exe 2010-07-06
0x12d6738 UDPv4 0.0.0.0:0 *:*: 1140 svchost.exe 2010-07-06
0x12d6738 UDPv6 :::0 *:*: 1140 svchost.exe 2010-07-06
[snip]
```

Windows Vista, 2008 Server, 7 Image 에서 network artifacts 를 찾아주는 명령어 이다.

TCP endpoints, listeners, UDP endpoints, listeners 를 찾아주며, IPv4 와 IPv6, local/remote IP, port(적용 가능 시) 구분을 해준다. Socket bound, Connection established, TCP 일 경우 현재 상태도 출력해준다.

아래의 주소를 통해 조금 더 자세한 설명을 확인할 수 있다.

<http://mnin.blogspot.com/2011/03/volatilitys-new-netscan-module.html>

Registry

Volatility는 Memory Forensic Framework로, 단지 Registry data를 추출하는데 도움을 준다. 조금 더 자세한 정보는 아래의 주소를 통해 확인 가능하다.

<http://moyix.blogspot.com/2009/01/memory-registry-tools.html>

<http://moyix.blogspot.com/2009/01/registry-code-updates.html>

hivescan

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp hivescan
Volatile Systems Volatility Framework 2.0
Offset      (hex)
1493000     0x0016c808
37018064    0x0234d9d0
66253488    0x03f2f2b0
73746896    0x046549d0
86327304    0x05254008
148837272   0x08df1398
148838864   0x08df19d0
153573840   0x092759d0
[snip]
```

Memory Dump 로 부터 CMHIVES(Registry hives)의 Physical address 를 찾아주는 명령어이다. 조금 더 자세한 정보는 아래의 주소를 통해 확인 가능하다.

<http://moyix.blogspot.com/2008/02/enumerating-registry-hives.html>

hivelist

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp hivelist
Volatile Systems Volatility Framework 2.0
Virtual      Physical      Name
0x99f0d008   0x0da7d008   \??\C:\windows\ServiceProfiles\LocalService\NTUSER.DAT
0x9c1692b0   0x03f2f2b0   \??\C:\Users\admin\ntuser.dat
0x9c7dc5c8   0x0d46d5c8   \??\C:\Users\admin\AppData\Local\Microsoft\Windows\UsrClass.dat
0x9cc839d0   0x046549d0   \??\C:\windows\System32\SMI\Store\Machine\SCHEMA.DAT
0x82baa140   0x02baa140   [no name]
0x8780c008   0x0ac97008   [no name]
0x878197b8   0x0ab5e7b8   \REGISTRY\MACHINE\SYSTEM
0x878419d0   0x0a7489d0   \REGISTRY\MACHINE\HARDWARE
0x8c089398   0x08df1398   \SystemRoot\System32\Config\SOFTWARE
0x8c0899d0   0x08df19d0   \SystemRoot\System32\Config\SECURITY
0x8c18f9d0   0x092759d0   \SystemRoot\System32\Config\DEFAULT
0x8e23e008   0x05254008   \SystemRoot\System32\Config\SAM
0x980e75e0   0x0dccc5e0   \??\C:\windows\System32\config\COMPONENTS
0x984709d0   0x0234d9d0   \Device\HarddiskVolume1\Boot\BCD
0x99e22808   0x0016c808   \??\C:\System Volume Information\Syscache.hve
[snip]
```

Memory Dump 로 부터 Registry hives 의 Virtual address 와 Disk 상의 절대 경로를 알려주는 명령어 이다.

printkey

```
$ python vol.py --profile=Win7SP0x86 -f win7.dmp printkey -K "Microsoft\Security Center\Svc"
Volatile Systems Volatility Framework 2.0
Legend: (S) = Stable (V) = Volatile

-----
Registry: \SystemRoot\System32\Config\SOFTWARE
Key name: Svc (S)
Last updated: 2010-07-06 22:33:20

Subkeys:
(V) Vol

Values:
REG_QWORD      VistaSp1      : (S) 128920209537502489
REG_DWORD      AntiVirusOverride : (S) 0
REG_DWORD      AntiSpywareOverride : (S) 0
REG_DWORD      FirewallOverride : (S) 0
```

특정 Registry key 의 subkeys, values, data, data type 를 보여주는 명령어 이다. 기본적으로 printkey 는 모든 hives 로 부터 검색을 하여 찾아진 key information 을 출력하므로, -K Option 을 통하여 포함 문자열을 함께 적어 주는 것이 좋다.

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp printkey -o 0x8c089398
Volatile Systems Volatility Framework 2.0
Legend: (S) = Stable (V) = Volatile

-----
Registry: User Specified
Key name: CMI-CreateHive{3D971F19-49AB-4000-8D39-A6D9C673D809} (S)
Last updated: 2010-03-10 22:48:38

Subkeys:
(S) AccessData
(S) BreakPoint
(S) Classes
(S) Clients
(S) Foxit Software
(S) Intel
(S) Microsoft
(S) Mozilla
(S) mozilla.org
[snip]
```

printkey 는 Virtual address 정보를 입력 받아서 해당 하는 Key 를 출력하는 기능도 지원하고 있으므로, 위의 그림과 같이 -o Option 을 이용해서 사용해도 된다.

hivedump

```
$ python vol.py --profile=win7SP0x86 -f win7.dmp hivedump -o 0x8e23e008
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Administrators
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Backup Operato
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Cryptographic
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Distributed CO
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Event Log Read
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Guests
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\IIS_IUSRS
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Network Config
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Performance Lo
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Performance Mo
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Power Users
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Remote Desktop
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Replicator
2010-03-09 19:50:19 \CMI-CreateHive{899121E8-11D8-4486-ACEB-301713D5ED8C} \SAM\Domains\BuiltinAliases\Names\Users
[snip]
```

hive 로 부터 모든 subkey 를 list 형태로 보여주는 명령어 이다. -o Option 을 통해 원하는 hive 의 Virtual address 를 지정해 주면 해당 hive 에 대한 subkey 를 list 형태로 출력 가능하다.

hashdump

```
$ python vol.py hashdump -f image.dd -y 0xe1035b60 -s 0xe165cb60
Administrator:500:08f3a52bdd35f179c81667e9d738c5d9:ed88ccbc08d1c18bcded317112555f4:::
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:::
```

주로 SAM File 을 추출할 때 쓰이는 명령어 이다. SYSTEM hive 의 Virtual address 를 -y Option 을 통해 넣어주고 SAM hive 의 Virtual address 를 -s Option 을 통해 넣어주면 위와 같이 추출이 가능하며, 이를 John the Ripper, rainbow tables 와 같은 Tool 을 이용해 Crack 가능하다.

자세한 설명은 아래의 주소를 통해 확인할 수 있다.

<http://moyix.blogspot.com/2008/02/cached-domain-credentials.html>

<http://www.slideshare.net/mooyix/sans-forensics-2009-memory-forensics-and-registry-analysis>

Memory Dump 로 부터 registry key 를 읽을 수 없는 경우는 "ERROR : volatility.plugins.registry.lsadump: Unable to read hashes from registry"와 같은 Error 를 확인할 수 있으며 이는 아래와 같은 방법을 통해 해결한다.


```
$ ./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
```

만약 SYSTEM 의 "CurrentControlSet\Control\lsa"와 SAM 의 "Domains\Account"에 올바른 key 가 존재하는 것을 확인 가능하다면 위처럼 volshell 을 이용해 "CurrentControlSet"을 받아야 한다.

```
$ ./vol.py -f XPSP3.vmem --profile=WinXPSP3x86 printkey -K "ControlSet001\Control\lsa" --no-cache
$ ./vol.py -f XPSP3.vmem --profile=WinXPSP3x86 printkey -K "SAM\Domains\Account" --no-cache
```

그 후, printkey 명령어를 사용하여 해당 data 와 keys 가 존재하는지 확인 가능하다. 만약 key 가 없다면 "The requested key could not be found in the hive(s) searched"라는 Error 를 확인할 수 있다.

lsadump

```
$ python vol.py -f laqma.vmem lsadump -y 0xe1035b60 -s 0xe16a6b60
Volatile Systems Volatility Framework 2.0
LSRMTIMEBOMB_1320153D-8DA3-4e8e-B27B-0D888223A588

0000 00 92 8D 60 01 FF C8 01 ...`....
_SC_Dnscache
LSHYDRAENCKEY_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 38 BA 41 C8 53 D7 CE C6 ...|P...;A.S...
0030 C8 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 B8 F0 74 E8 ..b^...!y...!..t.
0050 85 66 F4 C4 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 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..81.
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.1.....6...I.
0140 83 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 .....
0160 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 .....

DPAPI_SYSTEM

0000 01 00 00 00 24 04 D6 B0 DA D1 3C 40 B8 EE EC 89 ....$. ....<@....
0010 B4 B8 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 .....

```

Registry 로 부터 LSA dump 를 수행하는 명령어 이며, 이는 Default password, RDP public key, DPAPI credentials 등의 정보를 출력 가능하다.

SYSTEM hive 의 Virtual address 를 -y Option 을 통해 넣어주고 SECURITY hive 의 Virtual address 를 -s Option 을 통해 넣어주면 위와 같이 추출이 가능하다.

조금 더 자세한 정보는 아래 주소를 통해 확인 가능하다.

<http://moyix.blogspot.com/2008/02/decrypting-lsa-secrets.html>

userassist

```
$ ./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 08 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]
```

Userassist Keys 를 추출하는 명령어 이다.

조금 더 자세한 설명은 아래 주소를 통해 확인 가능하다.

<http://gleeda.blogspot.com/2011/04/volatility-14-userassist-plugin.html>

Crash Dumps, Hibernation, and Conversion

crashinfo

```
$ python vol.py crashinfo -f win7.dmp --profile=Win7SP0x86
Volatile Systems Volatility Framework 2.0
DUMP_HEADER32:
Majorversion:      0x0000000f (15)
Minorversion:      0x00001db0 (7600)
KdSecondaryVersion 0x00000041
DirectoryTableBase 0x00185000
PfnDataBase         0x83a00000
PsLoadedModuleList  0x82984810
PsActiveProcessHead 0x8297ce98
MachineImageType    0x0000014c
NumberProcessors    0x00000001
BugCheckCode        0x5454414d
PaeEnabled           0x00000001
KdDebuggerDataBlock 0x82964be8
ProductType         0x45474150
SuiteMask           0x45474150
WriterStatus        0x45474150

Physical Memory Description:
Number of runs: 3
FileOffset  Start Address  Length
00001000    00001000    0009e000
0009f000    00100000    3fdf0000
3fe8f000    3ff00000    00100000
3ff8e000    3ffff000
```

Crash dump header 정보를 출력해주는 명령어이며, Microsoft 사의 dumpcheck Utility 를 통해서도 해당 정보를 확인 가능하다.

hibinfo

```
$ python vol.py -f hiberfil.sys hibinfo
Volatile Systems Volatility Framework 2.0
IMAGE_HIBER_HEADER:
Signature: hibr
SystemTime: 2009-10-03 15:33:26

Control registers flags
CR0: 80010031
CR0[PAGING]: 1
CR3: 1a300060
CR4: 000006f9
CR4[PSE]: 1
CR4[PAE]: 1

Windows Version is 5.1 (2600)
```

Hibernation file 로 부터 Control Register 상태, file 생성시간, 상태, Hibernation 직전의 해당 Windows Version 과 같은 정보를 제공해주는 명령어 이다.

imagecopy

```
$ python vol.py imagecopy -f win7.dmp --profile=Win7SP0x86 -O win7.raw
Volatile Systems Volatility Framework 2.0
Writing data (5.00 MB chunks): |.....|
```

Crash dump, Hibernation file, live Firewire session 등을 Raw Memory Image 로 변환 시키는 명령어 이다. Windows XP SP2 가 아니라면 --profile Option 을 통하여 Profile 을 지정해줘야 하며, -O Option 을 통하여 output file 을 지정할 수 있다.

Malware and Rootkits

malfind

```
$ python vol.py -f zeus.vmem malfind -p 1724 -D hidden_dumps/
Volatile Systems Volatility Framework 2.0
Name      Pid      Start      End      Tag      Hits Protect
explorer.exe 1724 0x01600000 0x01600FFF VadS      0      6 (MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.01600000-01600fff.dmp
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 0.0{..U.....u..
0x01600020 55 8b ec e9 95 76 bc 75 8b ff 55 8b ec e9 be 53 U....v..u...S
0x01600030 bd 75 8b ff 55 8b ec e9 d6 18 c1 75 8b ff 55 8b .u...U.....u..U.
0x01600040 ec e9 14 95 bc 75 8b ff 55 8b ec e9 4f 7e bf 75 .....u...U...0~.u
0x01600050 8b ff 55 8b ec e9 0a 32 bd 75 8b ff 55 8b ec e9 ..U...2..u..U...
0x01600060 7d 61 bc 75 6a 2c 68 b8 8d 1c 77 e9 01 8c bc 75 }a.uj,h...w...u
0x01600070 8b ff 55 8b ec e9 c4 95 4b 70 8b ff 55 8b ec e9 ..U....Kp..U...

Disassembly:
01600000: b835000000          MOV EAX, 0x35
01600005: e9cdd7307b          JMP 0x7c90d7d7
0160000a: b891000000          MOV EAX, 0x91
0160000f: e94fdf307b          JMP 0x7c90df63
01600014: 8bff               MOV EDI, EDI
01600016: 55                PUSH EBP
01600017: 8bec               MOV EBP, ESP
01600019: e9ef17c175          JMP 0x7721180d
0160001e: 8bff               MOV EDI, EDI
01600020: 55                PUSH EBP

explorer.exe 1724 0x015D0000 0x015F5FFF VadS      0      6 (MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.015d0000-015f5fff.dmp
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 00 00 00 00 .....
0x015d0040 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68 .....!.!.!Th
0x015d0050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f is program canno
0x015d0060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20 t be run in DOS
0x015d0070 6d 6f 64 65 2e 0d 0a 24 00 00 00 00 00 00 00 00 mode...$......
```

malfind 명령어는 다양한 목적으로 사용 가능하다.

1. User-mode Memory 에서 숨겨지거나 삽입된 Code/Dll 들을 VAD tag 나 Page Permission 에 기반하여 찾을 때 사용할 수 있다.
2. 둘째로 User-mode 혹은 Kernel-mode Memory 에서 bytes, 정규 표현 식, ANSI 문자열, Unicode 문자열들의 위치를 찾는데 사용할 수 있다.

참고적으로 malfind 명령어의 목적은 일반적인 Method/Tool 로는 볼 수 없는 Dll 들을 찾는 것이기 때문에 CreateRemoteThread-> LoadLibrary 를 이용하여 삽입된 Dll 은 hidden 상태가 아니므로 dllist 명령어를 사용해서 확인 해야 한다.

```
$ python vol.py -f zeus.vmem malfind -D hidden_dumps --yara-rules=rules.yar
```

Process Memory 에서 Pattern 검색을 원한다면 YARA rules file 을 --yara-rules Option 을 통하여 생성 가능하다. 위 그림에서는 rules.yar File 안에 정의된 signature 들을 모든 Process 에서 찾는 명령을 설명하고 있다.

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

위 그림은 간단한 명령어를 모든 Process 에서 찾는 명령을 설명하고 있다.

Kernel Memory 에서 byte pattern 을 찾을 경우에는 Kernel Memory 전체를 탐색하는 것이 아니라, kernel driver 들에 의해 Load 되어 있는 Memory 만 탐색한다.

```
$ python vol.py -f tdl3.dmp malfind --yara-rules=rules.yar -D hidden_dumps --kernel
Volatile Systems Volatility Framework 2.0
Name      Pid      Start      End      Tag      Hits Protect
vmcs.sys  -        0xF9DB8000 0xF9DB8000 -        1 -      (Unknown)
Hit: 8b00813854444c33755a
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...
0xf9dba503 00 00 8b 80 00 00 df ff ff b0 00 01 00 00 b8 08 .....
0xf9dba513 03 00 00 8b 80 00 00 df ff 8b 00 05 84 03 00 00 .....
0xf9dba523 ff d0 eb 10 8b 45 e0 8b 40 0c 89 45 e0 e9 66 fe .....E..@..E..f.
0xf9dba533 ff ff 33 c0 c9 c2 08 00 5e f8 a8 f2 fe 63 ec d0 ...3.....^....c..
```

TDL3 는 disk 상의 atapi.sys File .rsrc section 에 shell code 를 삽입하고 addressOfEntryPoint 가 shell code 를 가리키게 하며, cmp dword ptr [eax], '3LDT' 와 같은 문자와 같은 특정한 code 가 존재한다.

위 그림은 이를 통해 rule 을 생성한 후 탐지를 실행한 결과이다.

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

위 그림은 해당 byte pattern 을 특정 Process 영역에서 찾는 명령을 수행한 것이다..

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

위 그림은 정규 표현 식을 통해 특정 Process 영역에서 찾는 명령을 수행한 것이다.

svcsan

```
$ python vol.py svcsan -f lagma.vmem
Volatile Systems Volatility Framework 2.0
Record      Order      Pid      Name      DisplayName      Type      Sta
[snip]
0x6ea738     0xf5       1148     WebClient WebClient        SERVICE_WIN32_SHARE_PROCESS SER
0x6ea7c8     0xf6       1028     winmgmt  Windows Management Instrumentation SERVICE_WIN32_SHARE_PROCESS SER
0x6ea858     0xf7       ----- WmdmPmSN Portable Media Serial Number Service SERVICE_WIN32_SHARE_PROCESS SER
0x6ea8e8     0xf8       ----- Wmi      Windows Management Instrumentation Driver Extensions SERVICE_WIN32_SHARE_PR
0x6ea970     0xf9       ----- WmiApSrv WMI Performance Adapter        SERVICE_WIN32_OWN_PROCESS SER
0x6eaa00     0xfa       ----- WS2IFSL  Windows Socket 2.0 Non-IFS Service Provider Support Environment SERVICE_KER
0x6eaa90     0xfb       1028     wscsvc   Security Center        SERVICE_WIN32_SHARE_PROCESS SER
0x6eab20     0xfc       1028     wuauerv  Automatic Updates      SERVICE_WIN32_SHARE_PROCESS SER
0x6eabb0     0xfd       1028     WZCSVC   Wireless Zero Configuration SERVICE_WIN32_SHARE_PROCESS SER
0x6eac40     0xfe       ----- xmlprov Network Provisioning Service SERVICE_WIN32_SHARE_PROCESS SER
0x6eacd0     0xff       ----- lanmandrv lanmandrv              SERVICE_KERNEL_DRIVER SER
```

Memory Image 에 등록된 service 목록을 확인할 수 있는 명령어 이며, 각 service 의 Process ID, Service name, Service display name, Service type, Current status, Binary path 을 확인할 수 있다.

Binary path 의 경우 User-mode service 뿐만 아니라,Kernel-mode 에서 사용된 service 의 Driver 이름까지도 출력 해 준다.

위 그림에서는 lanmandrv 라는 의심스러운 Kernel driver 를 설치하고 있는 것을 확인 가능하다.

ldrmodules

```
$ python vol.py -f laqma.vmem ldrmodules
Volatile Systems Volatility Framework 2.0
Pid Process Base InLoad InInit InMem Path
608 csrss.exe 0x010E0000 0 0 0 \WINDOWS\Fonts\vgasys.fon
608 csrss.exe 0x75B60000 0 0 0 \WINDOWS\system32\winsrv.dll
608 csrss.exe 0x77D40000 0 0 0 \WINDOWS\system32\user32.dll
632 winlogon.exe 0x01000000 1 0 1 \WINDOWS\system32\winlogon.exe
632 winlogon.exe 0x77DD0000 1 1 1 \WINDOWS\system32\advapi32.dll
632 winlogon.exe 0x77D40000 1 1 1 \WINDOWS\system32\user32.dll
676 services.exe 0x01000000 1 0 1 \WINDOWS\system32\services.exe
676 services.exe 0x758E0000 1 1 1 \WINDOWS\system32\scserv.dll
688 lsass.exe 0x01000000 1 0 1 \WINDOWS\system32\lsass.exe
936 svchost.exe 0x01000000 1 0 1 \WINDOWS\system32\svchost.exe
1028 svchost.exe 0x01000000 1 0 1 \WINDOWS\system32\svchost.exe
1028 svchost.exe 0x20000000 1 1 1 \WINDOWS\system32\xpsp2res.dll
1028 svchost.exe 0x76D30000 1 1 1 \WINDOWS\system32\wm1.dll
1028 svchost.exe 0x77F60000 1 1 1 \WINDOWS\system32\shlwapi.dll
[snip]
```

PEB 안의 linked list 에서 Unlinking 을 통해 DLL 을 숨기는 기법의 경우 VAD(Virtual address Descriptor)에 정보(Base address, Full path)가 남아 있기 때문에 이를 탐지 가능하다. 이 명령어는 Unlinked 된 명령어를 출력해 준다.

해당 명령어의 결과에서 Memory mapping 된 PE File 이 PEB List 에 존재하면 1, 하지 않으면 0 을 출력해 준다.

```
$ python vol.py -f laqma.vmem ldrmodules -v
Volatile Systems Volatility Framework 2.0
Pid Process Base InLoad InInit InMem Path
[snip]
1028 svchost.exe 0x77C10000 1 1 1 \WINDOWS\system32\msvcrt.dll
Load Path: C:\WINDOWS\system32\msvcrt.dll : msvcrt.dll
Init Path: C:\WINDOWS\system32\msvcrt.dll : msvcrt.dll
Mem Path: C:\WINDOWS\system32\msvcrt.dll : msvcrt.dll
1028 svchost.exe 0x76D10000 1 1 1 \WINDOWS\system32\clusapi.dll
Load Path: C:\WINDOWS\System32\CLUSAPI.DLL : CLUSAPI.DLL
Init Path: C:\WINDOWS\System32\CLUSAPI.DLL : CLUSAPI.DLL
Mem Path: C:\WINDOWS\System32\CLUSAPI.DLL : CLUSAPI.DLL
1028 svchost.exe 0x76E80000 1 1 1 \WINDOWS\system32\rtutils.dll
Load Path: c:\windows\system32\rtutils.dll : rtutils.dll
Init Path: c:\windows\system32\rtutils.dll : rtutils.dll
Mem Path: c:\windows\system32\rtutils.dll : rtutils.dll
1028 svchost.exe 0x714A0000 1 1 1 \WINDOWS\system32\ws2help.dll
Load Path: c:\windows\system32\WS2HELP.dll : WS2HELP.dll
Init Path: c:\windows\system32\WS2HELP.dll : WS2HELP.dll
Mem Path: c:\windows\system32\WS2HELP.dll : WS2HELP.dll
[snip]
```

경로를 덮어쓰기 함으로써 악성코드가 DLL 을 숨길 수 있으며, 이는 모든 Entry 의 Full path 를 보는 -v 또는 --verbose Option 을 사용하여 확인할 수 있다.

impscan

Image 로 부터 Reversing 을 위해 완벽한 PE File 을 획득하기 위해서는 어떤 함수를 Import 하는지 알아야 할 것이다. 즉, 어떠한 API 함수가 호출하는 지를 알아야 한다. dlldump, moddump, precexedump/procmemdump 를 통해 Binary dump 를 수행하면 IAT(Import address Table)은 rebuild 되지 않는다. 이는 Memory page 가 Disk 에 paged 되기까지 많은 시간이 요구 하기 때문이다.

그러나 impscan, Impscan 는 PE File 의 IAT 를 사용하지 않고 API 를 호출을 탐지한다. 심지어 악성코드가 Kernel driver 에서 사용되고, PE header 를 완전히 지웠더라도 이는 동작한다.

만약 IDA Pro가 설치되어 있고, 환경변수에 등록되어 있다면 impscan은 자동으로 IDB 파일을 생성하므로, 해당 File을 Reversing할 수 있을 것이다.

```
$ python vol.py -f conficker.bin -p 3108 malfind -D out/
Volatile Systems Volatility Framework 2.0
Name Pid Start End Tag Hits Protect
notepad.exe 3108 0x00A10000 0x00A2BFFF VadS 0 24 (MM_EXECUTE_UNKNOWN)
Dumped to: out/services.exe.20c8558.00a10000-00a2bfff.dmp
0x00a10000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00a10070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

위의 그림은 target Process 에서 Load 될 때 PE Header 를 삭제하는 악성코드를 예로 든 것이며, malfind 명령으로 Base address 를 찾은 그림이다.

```
$ python vol.py -f conficker.bin -p 3108 impscan -a 0x00A10000 -D out
Volatile Systems Volatility Framework 2.0
a11204 RPCRT4.dll NdrClientCall2 77ef44d0
a11208 RPCRT4.dll RpcStringBindingComposeA 77e9a8e4
a110ac kernel32.dll Process32First 7c864f55
a1120c RPCRT4.dll RpcBindingFromStringBindingA 77e9a898
a11210 RPCRT4.dll RpcBindingFree 77e7b3d8
a11058 kernel32.dll FreeLibrary 7c80ac7e
a11220 VERSION.dll VerQueryValueA 77c018aa
a110b0 kernel32.dll Process32Next 7c8650c8
a11224 VERSION.dll GetFileVersionInfoA 77c01a40
a11228 VERSION.dll GetFileVersionInfoSizeA 77c019ef
a11230 WININET.dll InternetOpenA 771c5796
a11034 kernel32.dll WriteFile 7c810e27
a11238 WININET.dll InternetReadFile 771c82f2
a1123c WININET.dll InternetCloseHandle 771c4d94
a11240 WININET.dll InternetGetConnectedState 771d5c8e
a11260 WS2_32.dll sendto 71ab2f51
a11244 WININET.dll InternetOpenUrlA 771c5a62
[snip]
```

해당 악성코드의 Base address 를 -a Option 과 함께 지정해 주면, 해당 Binary 의 import 함수들을 확인할 수 있다. 만약 Base address 를 지정해 주지 않으면, Process 의 main module 끝까지 탐색 할 것이다.

impscan 은 Kernel driver 의 Base address 도 지정 가능하다.

```
$ python vol.py -f laqma.vmem modules | grep lanman
Volatile Systems Volatility Framework 2.0
\\?\C:\WINDOWS\System32\lanmandrv.sys 0x00f8c52000 0x002000 lanmandrv.sys
```

Laqma 는 lanmandrv.sys 를 호출하며, 이를 moddump 로 Dump 를 수행하면 IAT 가 손상되어 있을 것이고, impscan 을 통하여 이를 Rebuild 해야 한다.

```
$ python vol.py -f laqma.vmem impscan -a 0x00f8c52000 -D out/
Volatile Systems Volatility Framework 2.0
f8c53080 ntoskrnl.exe IoCompleteRequest 804ee07a
f8c530a0 ntoskrnl.exe ZwOpenKey 804fdd8c
f8c530a4 ntoskrnl.exe _except_handler3 80535230
f8c53084 ntoskrnl.exe IoDeleteDevice 804f0776
f8c53088 ntoskrnl.exe IoDeleteSymbolicLink 8056833a
f8c5309c ntoskrnl.exe wcsncpy 80536fe3
f8c533ac ntoskrnl.exe NtQueryDirectoryFile 8056e1c2
f8c5308c ntoskrnl.exe IoCreateSymbolicLink 80567fc6
f8c53090 ntoskrnl.exe MmGetSystemRoutineAddress 805a23dc
f8c53094 ntoskrnl.exe IoCreateDevice 80569c5e
f8c53098 ntoskrnl.exe ExAllocatePoolWithTag 80544280
f8c533b4 ntoskrnl.exe NtQuerySystemInformation 806065e4
f8c533bc ntoskrnl.exe NtOpenProcess 805bfe1e
```

modules 명령으로 확인한 Base address 를 통하여 이를 복구도 가능하다.

apihooks

User-mode 또는 Kernel-mode 에서의 API Hooking 을 탐지하기 위해서 사용하는 명령어로, 이는 IAT, EAT, Inline Hooking 을 찾아낼 수 있다.

Inline Hooking 을 탐지하기 위해서 CALL, JMP Operand 가 직접 또는 간접적으로 위치를 참조하는 것, PUSH/RET 명령어를 탐지하며, ntdll.dll 의 syscall 이나 Kernel memory 의 unknown code page 를 호출하는 것을 탐지 한다.

```
$ python vol.py -f coreflood.vmem -p 2044 apihooks
Volatile Systems Volatility Framework 2.0
Name                                     Type      Target                                     Value
IEXPLORE.EXE[2044]@winspool.drv         iat       KERNEL32.dll!GetProcAddress             0x0 0x7ff82360 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv         iat       KERNEL32.dll!LoadLibraryW              0x0 0x7ff82ac0 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv         iat       KERNEL32.dll!CreateFileW              0x0 0x7ff82240 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv         iat       KERNEL32.dll!LoadLibraryA             0x0 0x7ff82a50 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv         iat       ADVAPI32.dll!RegSetValueExW            0x0 0x7ff82080 (UNKNOWN)
[snip]
```

IAT hooking 을 탐지하는 예를 살펴보면 가장 오른쪽 Field 에 UNKNOWN 이 보일 것이다. 이는 Rootkit Code 가 존재하는 Memory 와 연결된 Module 이 없기 때문이며, 만약 hook 을 포함하는 Code 를 추출하고 싶다면 아래와 같은 방법을 따르면 된다.

1. malfind 명령어가 자동으로 찾고 추출 가능하다면 이를 사용
2. volshell 의 dd/db 명령어를 통해 MZ header 를 찾은 후, dlldump 를 --base 값과 함께 사용하여 추출
3. vaddump 명령어를 통해 모든 Code segment 들을 추출 한 후(File 명은 주소 범위) 그 후 해당 범위에 해당하는 Dump File 을 찾음

아래 화면은 Inline Hooking 을 탐지한 화면이다.

```
$ python vol.py -f silentbanker.vmem -p 1884 apihooks
Volatile Systems Volatility Framework 2.0
Name                                     Type      Target                                     Value
IEXPLORE.EXE[1884]                       inline    ws2_32.dll!connect                     0x71ab406a JMP 0xe90000 (UNKNOWN)
IEXPLORE.EXE[1884]                       inline    ws2_32.dll!send                        0x71ab428a JMP 0xe70000 (UNKNOWN)
IEXPLORE.EXE[1884]                       inline    user32.dll!DispatchMessageA           0x77d4bcbd JMP 0x10e0000 (UNKNOWN)
IEXPLORE.EXE[1884]                       inline    user32.dll!DispatchMessageW           0x77d489d9 JMP 0x1100000 (UNKNOWN)
IEXPLORE.EXE[1884]                       inline    user32.dll!GetClipboardData           0x77d6fcb2 JMP 0x10c0000 (UNKNOWN)
[snip]
```

```
$ python vol.py -f laqma.vmem -p 1624 apihooks
Volatile Systems Volatility Framework 2.0
Name                                     Type      Target                                     Value
explorer.exe[1624]                       inline    user32.dll!MessageBoxA                 0x7e45058a PUSH 0xac10aa; RET (D11.d11)
explorer.exe[1624]                       inline    crypt32.dll!CertSerializeCRLStoreElement 0x77aa28df PUSH 0xac1104; RET (D11.d11)
explorer.exe[1624]                       inline    crypt32.dll!CertSerializeCTLStoreElement 0x77aa28df PUSH 0xac1104; RET (D11.d11)
explorer.exe[1624]                       inline    crypt32.dll!CertSerializeCertificateStoreElement 0x77aa28df PUSH 0xac1104; RET (D11.d11)
explorer.exe[1624]                       inline    crypt32.dll!PFXImportCertStore         0x77aef748 PUSH 0xac12a8; RET (D11.d11)
explorer.exe[1624]                       inline    wininet.dll!HttpOpenRequestA           0x771c36dd PUSH 0xac148c; RET (D11.d11)
explorer.exe[1624]                       inline    wininet.dll!HttpSendRequestA           0x771c6129 PUSH 0xac162c; RET (D11.d11)
[snip]
```

```
$ python vol.py -f carberp.vmem apihooks
Volatile Systems Volatility Framework 2.0
Name                                     Type      Target                                     Value
explorer.exe[1004]                       inline    ntdll.dll!NtCreateThread                0x7c90d190 JMP 0x15a3fa7 (UNKNOWN)
explorer.exe[1004]                       inline    ntdll.dll!ZwCreateThread                0x7c90d190 JMP 0x15a3fa7 (UNKNOWN)
explorer.exe[1004]                       syscall  ntdll.dll!NtQueryDirectoryFile          0x1dadd84 MOV EDX, 0x1dadd84 (UNKNOWN)
explorer.exe[1004]                       syscall  ntdll.dll!NtResumeThread                0x1dadd78 MOV EDX, 0x1dadd78 (UNKNOWN)
explorer.exe[1004]                       syscall  ntdll.dll!ZwQueryDirectoryFile          0x1dadd84 MOV EDX, 0x1dadd84 (UNKNOWN)
explorer.exe[1004]                       syscall  ntdll.dll!ZwResumeThread                0x1dadd78 MOV EDX, 0x1dadd78 (UNKNOWN)
explorer.exe[1004]                       inline    ws2_32.dll!WSASend                      0x71ab68fa JMP 0x15a97f7 (UNKNOWN)
explorer.exe[1004]                       inline    ws2_32.dll!closesocket                  0x71ab3e2b JMP 0x15a979e (UNKNOWN)
[snip]
```

```
$ python vol.py apihooks -K -f rustock.vmem
Name      Type      Function      Value
-         -         -             -
inlinek   ntoskrnl.exe!IoFCallDriver 0x804ee130 jmp [0x8054c280] ==> 0xb17a189d (\Driver\pe386)
```

위 그림은 Kernel-mode 함수의 Inline Hooking 함수를 탐지한 것이다.

```
$ python vol.py -f rustock-2.vmem apihooks -K
Volatile Systems Volatility Framework 2.0
Name      Type      Target      Value
-         -         -             -
ucpcall   tcpip.sys 0xf7be2514 CALL [0x81ecd0c0]
ucpcall   tcpip.sys 0xf7be28ad CALL [0x81e9da60]
ucpcall   tcpip.sys 0xf7be2c61 CALL [0x81f8a058]
ucpcall   tcpip.sys 0xf7bfa0c0 CALL [0x82009dd0]
```

Kernel driver 의 unknown code page 를 CALL 하는 부분도 탐지할 수 있으며, 해당 그림에서는 tcpip.sys 에 의심스러운 redirection 들이 보인다.

idt

```
$ python vol.py idt -f rustock.vmem
Index      Selector Function      Value      Details
[snip]
2A         8      KiGetTickCount 0x8053cbae ntoskrnl.exe .text
2B         8      KiCallbackReturn 0x8053ccb0 ntoskrnl.exe .text
2C         8      KiSetLowWaitHighThread 0x8053ce50 ntoskrnl.exe .text
2D         8      KiDebugService 0x8053d790 ntoskrnl.exe .text
2E         8      KiSystemService 0x806b973c ntoskrnl.exe .rsrc ==> JMP 0xf6ec0e45
[snip]
```

System 의 IDT(Interrupt Descriptor Table) 현재 주소, Module, Interrupt 목적 등을 보여주는 명령어 이다. 위 그림에서 볼 수 있듯이 Inline Hooking 을 탐지하기 위해 IDT Entry 들도 Check 한다.

gdt

```
$ python vol.py -f alipop.vmem gdt
Volatile Systems Volatility Framework 2.0
Sel      Base      Limit      Type      DPL      Gr      Pr
0x0      0xffdf0a 0xdbbb    TSS16 Busy 2      By      P
0x8      0x0      0xffffffff Code RE Ac 0      Pg      P
0x10     0x0      0xffffffff Data RW Ac 0      Pg      P
0x18     0x0      0xffffffff Code RE Ac 3      Pg      P
0x20     0x0      0xffffffff Data RW Ac 3      Pg      P
0x28     0x80042000 0x20ab    TSS32 Busy 0      By      P
0x30     0xffdf000 0x1fff    Data RW Ac 0      Pg      P
0x38     0x0      0xffff    Data RW Ac 3      By      P
[...]
0x3c8    0x8003    0xf3d0    <Reserved> 0      By      Np
0x3d0    0x8003    0xf3d8    <Reserved> 0      By      Np
0x3d8    0x8003    0xf3e0    <Reserved> 0      By      Np
0x3e0    0x8003f000 -          CallGate32 3      -      P
8003f000: bdb0adfff MOV EBX, 0xffdf0adb
8003f005: c3 RET
0x3e8    0x0      0xffffffff Code RE Ac 0      Pg      P
0x3f0    0x8003    0xf3f8    <Reserved> 0      By      Np
0x3f8    0x0      0x0       <Reserved> 0      By      Np
```

System 의 GDT(Global Descriptor Table) 를 탐지하는 명령어 이며, 이는 Call gate 를 설치하는 악성코드 탐지에 매우 유용하다.

위 그림에서 볼 수 있듯이 0x3e0 Selector 는 32-bit Call gate 를 위한 목적으로 감염된 것을 확인 가능하며, Hook Address 는 0x8003F000 임을 볼 수 있다. 해당 주소를 Disassembly 화면으로 보면 EBX 에 0xFFDF0ADB 의 값을 넣는 것을 확인 가능하다.


```

$ python vol.py -f alipop.vmem volshell
Volatile Systems Volatility Framework 2.0
Current context: process System, pid=4, ppid=0 DTB=0x320000
Welcome to volshell! Current memory image is:
file:///Users/M/Desktop/alipop.vmem
To get help, type 'hh()'

>>> dis(0xffff0adb, length=32)
0xffff0adb c8000000 ENTER 0x0, 0x0
0xffff0adf 31c0 XOR EAX, EAX
0xffff0ae1 60 PUSHA
0xffff0ae2 8b5508 MOV EDX, [EBP+0x8]
0xffff0ae5 bb00704d80 MOV EBX, 0x804d7000
0xffff0aea 8b4b3c MOV ECX, [EBX+0x3c]
0xffff0aed 8b6c0b78 MOV EBP, [EBX+ECX+0x78]

>>> db(0xffff0adb + 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 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 .@.....
ffdf0c56 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...

```

조금 더 많은 조사를 원한다면 위처럼 volshell 을 사용하면 된다.

threads

```

$ python vol.py -f test.vmem threads -L
Volatile Systems Volatility Framework 2.0
Tag Description
-----
DkmExit 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

```

각 Thread 에 속한 Register 정보, Thread 시작 주소의 Disassemble Code 등 조사에 관련된 다양한 정보를 제공해주는 명령어 이다. Default 로 모든 Thread 에 대한 정보를 제공해 주기 때문에 정렬에 어려움이 있을 것이며, 이 때는 -F Option 을 인자에 ","를 이용하여 여러 개의 Filter 를 적용 가능하다.

```

$ python vol.py -f XPSP3.vmem threads -F AttachedProcess
Volatile Systems Volatility Framework 2.0
-----
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

```

ETHREAD 객체의 가상 주소, pid, tid, Thread 와 관련된 모든 tag(SystemThread, AttachedProcess, HookedSSDT), 생성/종료 시간, 상태, 순서, 시작 주소 등을 확인 가능하며 SSDT base 주소와 각 Service Table, Table 안의 Hook 된 함수도 출력해 준다. 위 그림의 마지막 부분을 통해 Thread 처음 지점의 Disassembly Code 도 확인 가능하다. 자세한 설명은 아래의 주소를 통해 확인할 수 있다.

<http://mnin.blogspot.com/2011/04/investigating-windows-threads-with.html>

callbacks

Kernel callback 은 Rootkit, Anti-virus, Dynamic Analysis, Windows Monitoring Tool 들에 사용되며 Volatility 는 아래의 Callback 탐지를 지원한다.

- 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)

```
$ python vol.py -f be2.vmem callbacks
Volatile Systems Volatility Framework 2.0
Type          Callback  Owner
PsSetCreateThreadNotifyRoutine 0xff0d2ea7 00004A2A
PsSetCreateProcessNotifyRoutine 0xfc58e194 vmci.sys
KeBugCheckCallbackListHead      0xfc1e85ed NDIS.sys (Ndis miniport)
KeBugCheckReasonCallback        0x806d57ca hal.dll (ACPI 1.0 - APIC platform UP)
KeRegisterBugCheckReasonCallback 0xfc967ac0 mssmbios.sys (SMBiosData)
KeRegisterBugCheckReasonCallback 0xfc967a78 mssmbios.sys (SMBiosRegistry)
[snip]
```

Owner : 00004A2A 을 통해 BackEnergy2 Malware 임을 확인

```
$ python vol.py -f rustock.vmem callbacks
Volatile Systems Volatility Framework 2.0
Type          Callback  Owner
PsSetCreateProcessNotifyRoutine 0xf88bd194 vmci.sys
PsSetCreateProcessNotifyRoutine 0xb17a27ed '\\Driver\\pe386'
KeBugCheckCallbackListHead      0xf83e65ef NDIS.sys (Ndis miniport)
KeBugCheckReasonCallback        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]
```

Owner : '\\Driver\\pe386'을 통해 Rustock Rootkit 임을 확인


```
$ python vol.py -f ascesso.vmem callbacks
Volatile Systems Volatility Framework 2.0
Type Callback Owner
IoRegisterShutdownNotification 0xf853c2be ftdisk.sys (\Driver\Ftdisk)
IoRegisterShutdownNotification 0x805f5d66 ntoskrnl.exe (\Driver\ntoskrnl)
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
```

위 그림의 경우 CmRegisterCallback 이 0x8216628F 주소를 가리키고 Owner 가 UNKNOWN 으로 나타나며, 그 아래 같은 주소에 GenericKernelCallback Type 이 2 개 존재한다. 이와 같은 결과가 나오는 이유는 탐색을 Pool tag Scanning 과 같이 하기 때문이며, 탐색이 실패 하더라도 Pool tag Scanning 을 수행을 통해 정보를 출력 가능한 것을 의미한다. 위의 GenericKernel 이라는 표현은 Windows Kernel 에서 같은 type 의 Pool tag 를 다양한 callback 에 사용하기 때문이다.

driverirp

Driver 의 IRP(Major Function) Table 을 보기 위해서 사용하는 명령어 이며, 해당 명령어는 driverscan 에 속해 있기 때문에 해당 DRIVER_OBJECT 들을 찾을 수 있으며, 이 후 함수 Table 을 통해 Cycle 을 순회하면서 각 함수들의 목적, 주소, 주소의 소유 Module 들을 출력해 준다.

많은 Driver 들이 IRP 함수들을 정상적인 목적으로 가져 올 수도 있기 때문에 포함하는 Module 을 이용한 Hooking 된 IRP 함수 탐지 방법은 좋은 방법이 아니므로, 모든 것을 출력해준 후 각각 판단을 할 수 있게 도와줄 뿐이다.

해당 명령어는 IRP 함수의 Inline Hooking 탐지를 지원하며 -v 또는 --verbose Option 으로 해당 IRP 주소의 명령어들을 Disassemble 하여 제공 한다.

해당 명령어는 특정 정규 표현식을 사용하지 않는 경우 Default 로 모든 Driver 에 대해 출력해 준다

```
$ python vol.py -f tdl3.vmem driverirp -r vm SCSI
Volatile Systems Volatility Framework 2.0
DriverStart Name IRP IrpAddr IrpOwner HookAddr HookOwner
0xf9db8000 'vm SCSI' IRP_MJ_CREATE 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_CREATE_NAMED_PIPE 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_CLOSE 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_READ 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_WRITE 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_QUERY_INFORMATION 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_SET_INFORMATION 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_QUERY_EA 0xf9db9cbd vm SCSI.sys - -
0xf9db8000 'vm SCSI' IRP_MJ_SET_EA 0xf9db9cbd vm SCSI.sys - -
[snip]
```

위 그림에서는 vm SCSI.sys Driver 가 TDL3 Rootkit 에 감염되었다는 것을 분명히 보여주지는 않는다. 모든 IRP 가 vm SCSI.sys 를 다시 가리키지만 Rootkit 탐지 도구 우회일 수 있으므로 --verbose Option 을 통해 조금 더 정확히 확인해 보도록 한다.

```

$ python vol.py -f tdl3.vmem driverirp -r vmcsi --verbose
Volatile Systems Volatility Framework 2.0
DriverStart Name IRP IrpAddr IrpOwner HookAddr HookOwner
0xf9db8000 'vmcsi' IRP_MJ_CREATE 0xf9db9cbd vmcsi.sys - -
f9db9cbd: a10803dfff MOV EAX, [0xffdf0308]
f9db9cc2: ffa0fc000000 JMP DWORD [EAX+0xfc]
f9db9cc8: 0000 ADD [EAX], AL
f9db9cca: 0000 ADD [EAX], AL
f9db9ccc: 0000 ADD [EAX], AL

0xf9db8000 'vmcsi' IRP_MJ_CREATE_NAMED_PIPE 0xf9db9cbd vmcsi.sys - -
f9db9cbd: a10803dfff MOV EAX, [0xffdf0308]
f9db9cc2: ffa0fc000000 JMP DWORD [EAX+0xfc]
f9db9cc8: 0000 ADD [EAX], AL
f9db9cca: 0000 ADD [EAX], AL
f9db9ccc: 0000 ADD [EAX], AL
[snip]

```

위 그림을 통해 TDL3 가 모든 IRP 를 Redirect 하는 것을 확인 할 수 있다. 해당 Code 는 0xFFDF0308 을 가리키고 있으며 이는 KUSER_SHARED_DATA 영역 이다.

devicetree

Windows 는 계층화된 Driver Architecture 또는 Driver chain 을 사용하므로, 여러 개의 Driver 를 검사 또는 IRP 에 응답할 수 있다. Rootkit 은 Driver 나 Device 를 이러한 Chain 에 검사 우회의 목적으로 삽입한다.

devicetree Plugin 은 `_DRIVER_OBJECT.DeviceObject.NextDevice` 을 통해 해당 Driver 또는 Device 들의 관계를 보여주며 `_DRIVER_OBJECT.DeviceObject.AttachedDevice` 를 통해 Attach 된 Device 들을 보여준다.

```

$ python vol.py -f stuxnet.vmem devicetree
Volatile Systems Volatility Framework 1.4_rc1
[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]

```

해당 그림은 Stuxnet 이 `\\FileSystem\\Ntfs` 을 unnamed Device 를 통해 감염 시킨 것을 확인할 수 있으며 비록 Device 가 unnamed 라고 표시되어 있지만 해당 Device 객체는 `\\Driver\\MRxNet` 에서 확인 가능하다.

devicetree Plugin 은 Driver 를 "DRV"로, Device 를 "DEV"로 표현하며, Attached Device 를 "ATT"로 표현한다.

psxview

Process list 의 여러 다른 출처에 의해 보고되는 것과 PsActiveProcessHead 에 포함된 것을 비교하여 Hidden Process 를 탐지하는 명령어 이며, 해당 명령어가 비교하는 것은 아래와 같다.

- PsActiveProcessHead linked list
- EPROCESS pool scanning
- ETHREAD pool scanning(EPROCESS 를 참조)

- PspCidTable
- Csrss.exe handle table(Vista,7 에서는 불가, 몇몇 XP image 에서는 불가)
- Csrss.exe internal linked list(Vista,7 에서는 불가, 몇몇 XP image 에서는 불가)

```
$ python vol.py -f prolaco.vmem psxview
Offset      Name      Pid      pslist      psscan      thrdproc      pspcid      csr_hnds      csr_list
0xff1b8b28  vmtoolsd.exe  1668      1            1            1            1            1            0
0x80ff88d8  svchost.exe   856       1            1            1            1            1            0
0xff1d7da0  spoolsv.exe   1432      1            1            1            1            1            0
0x810b1660  System       4         1            1            1            1            0            0
0x80fbf910  svchost.exe   1028      1            1            1            1            1            0
0xff2ab020  smss.exe      544       1            1            1            1            0            0
0xff3667e8  VMwareTray.exe 432       1            1            1            1            1            0
0xff247020  services.exe  676       1            1            1            1            1            0
0xff217560  svchost.exe   936       1            1            1            1            1            0
0xff143b28  TPAutoConnSvc.e 1968      1            1            1            1            1            0
0x80fdc648  1_doc_RCData_61 1336      0            1            1            1            1            0
0xff255020  lsass.exe     688       1            1            1            1            1            0
0xff3865d0  explorer.exe  1724      1            1            1            1            1            0
0xff22d558  svchost.exe   1088      1            1            1            1            1            0
0xff374980  VMwareUser.exe 452       1            1            1            1            1            0
0xff1fdc88  VMUpgradeHelper 1788      1            1            1            1            1            0
0xff218230  vmacthlp.exe  844       1            1            1            1            1            0
0xff364310  wscntfy.exe   888       1            1            1            1            1            0
0x80f94588  wuauc1t.exe   468       1            1            1            1            1            0
0xff25a7e0  alg.exe       216       1            1            1            1            1            0
0xff1ecd40  csrss.exe     608       1            1            1            1            0            0
0xff38b5f8  TPAutoConnect.e 1084      1            1            1            1            1            0
0xff37a4b0  ImmunityDebugge 1136      1            1            1            1            1            0
0xff1ec978  winlogon.exe  632       1            1            1            1            1            0
0xff203b80  svchost.exe   1148      1            1            1            1            1            0
```

해당 열에 0 으로 나타난 것은 Process 가 손실 된 것을 나타내며, 해당 그림에서는 "1_doc_RCData_61"이 pslist 에서 제외된 것을 통해 수상한 것을 확인할 수 있다.

ssdt_ex

Rootkit 에 의해 설치된 SSDT hooking 에 대한 흔적을 검색 해주는 명령어 이다. 자동으로 어떤 SSDT 함수가 Hooking 된지 확인해 주며, Hooking kernel driver 를 디스크로 추출, IDA Script File 인 IDC File 을 Rootkit 의 함수 Label 을 포함하여 생성해 준다. 그 후, idag.exe(Windows), idal(Linux/OS X)가 \$PATH 에 있으면 IDB 파일을 Kernel driver 로 부터 생성하여 IDC Script 를 실행 할 수 있다.

```
$ python vol.py -f laqma.vmem ssdt_ex -D outdir/
Volatile Systems Volatility Framework 2.0
Entry 0x0049: 0xf8c52884 (NtEnumerateValueKey) owned by lanmandrv.sys
Entry 0x007a: 0xf8c5253e (NtOpenProcess) owned by lanmandrv.sys
Entry 0x0091: 0xf8c52654 (NtQueryDirectoryFile) owned by lanmandrv.sys
Entry 0x00ad: 0xf8c52544 (NtQuerySystemInformation) owned by lanmandrv.sys
Dumping IDC file to /Users/M/Desktop/Volatility-2.0/outdir/driver.f8c52000.sys.idc
[snip]
```

위 그림을 통해 outdir 를 살펴보면 추출된 Kernel Driver(driver.f8c52000.sys), IDC Script(drdiver.f8c52000.sys.idc), IDA Database(driver.f8c52000.idb)를 확인 할 수 있으며, IDC Script 를 보면 아래와 같다.

```
#include <idc.idc>
static main(void) {
    MakeFunction(0xf8c52a4c, BADADDR);
    MakeFunction(0xf8c52e7c, BADADDR);
    MakeName(0xf8c52544, "HookNtQuerySystemInformation");
    MakeFunction(0xf8c52544, BADADDR);
    MakeName(0xf8c52654, "HookNtQueryDirectoryFile");
    MakeFunction(0xf8c52654, BADADDR);
    MakeName(0xf8c52884, "HookNtEnumerateValueKey");
    MakeFunction(0xf8c52884, BADADDR);
    MakeName(0xf8c5253e, "HookNtOpenProcess");
    MakeFunction(0xf8c5253e, BADADDR);
    Exit(0);
}
```

IDB File 을 열어 "Hook" 단어가 앞에 붙은 함수를 유심히 살펴보면 된다.

timers

Installed 된 Kernel times(KTIMER)와 관련된 DPC(Deferred Procedure Calls)를 출력해 주는 명령어 이다. Zero Access, Rustock, Stuxnet 의 경우 DPC 를 이용하여 timer 를 등록한다. DPC 주소와 KTIMES 를 통해 악성코드가 다양한 방법으로 Kernel 공간에 숨어 있는 것을 빠르게 찾을 수 있다.

```
$ python vol.py timers -f rustock-c.vmem
Offset      DueTime      Period(ms)  Signaled  Routine      Module
0xf730a790  0x00000000:0x6db0f0b4 0             -          0xf72fb385   srv.sys
0x80558a40  0x00000000:0x68f10168 1000         Yes        0x80523026   ntoskrnl.exe
0x80559160  0x00000000:0x695c4b3a 0             -          0x80526bac   ntoskrnl.exe
0x820822e4  0x00000000:0xa2a56bb0 150000      Yes        0x81c1642f   UNKNOWN
0xf842f150  0x00000000:0xb5cb4e80 0             -          0xf841473e   Ntfs.sys
...
```

위 그림에서 UNKNOWN 을 통해 Rootkit 이 숨어 있는 것(DPC 가 Kernel Memory 영역의 UNKNOWN 부분을 가리키고 있음)을 확인할 수 있다.

```
$ python vol.py -f mem.dmp --profile=Win7SP0x86 kpcrscan
Volatile Systems Volatility Framework 2.1_alpha
Potential KPCR structure virtual addresses:
_KPCR: 0x807c3000
_KPCR: 0x8296fc00
_KPCR: 0x8cd00000
_KPCR: 0x8cd36000

$ python vol.py -h
Volatile Systems Volatility Framework 2.1_alpha
Usage: Volatility - A memory forensics analysis platform.

Options:
[...]
-k KPCR, --kpcr=KPCR Specify a specific KPCR address
[...]

$ python vol.py -f mem.dmp timers --profile=Win7SP0x86 --kpcr=0x807c3000
Volatile Systems Volatility Framework 2.1_alpha
Offset      DueTime      Period(ms)  Signaled  Routine      Module
0x869ed930  0x0000217a:0x47a09f1a 0             -          0x99a78005   srv.sys
0x85f451d8  0x0000217a:0x5f8703b2 10            -          0x8b0a78b9   tcpip.sys
0x807c65a8  0x0000217a:0x47e7b336 15000        Yes        0x8287b4d3   ntoskrnl.exe

$ python vol.py -f mem.dmp timers --profile=Win7SP0x86 --kpcr=0x8296fc00
Volatile Systems Volatility Framework 2.1_alpha
Offset      DueTime      Period(ms)  Signaled  Routine      Module
0x91593180  0x0000217a:0x4f4a5c00 0             -          0x9158e240   luafv.sys
0x829810a0  0x0000217a:0x4f4a5c00 5000         Yes        0x8290b2d0   ntoskrnl.exe
0x86135d70  0x0000217a:0x5b361e00 30000        Yes        0x8f6e0298   afd.sys
0x96243900  0x0000217a:0x4f5de140 0             -          0x9621b3b2   HTTP.sys
0x829ac4f0  0x0000217a:0x62940216 60000        Yes        0x82878d8f   ntoskrnl.exe
0x862d9150  0x0000217a:0x5448bae0 0             -          0x8aedda4f   ndis.sys
0x99a878c0  0x0000217a:0x51eb2594 0             -          0x99a780b6   srv.sys
0x8605dd20  0x0000217a:0x4dc62f22 0             -          0x82fbbba4   storport.sys
[...]
```

Windows XP, 2003, 2008, VISTA 의 경우 times 를 전역 변수에 저장하지만, Windows 7 의 경우 KPCR(Kernel Processor Control Region)에 저장하고 있다. 그렇기 때문에 Windows7 에서 모든 timers 들을 확인 하려면 위 그림처럼 kpcrscan 을 통해 KPCR 주소들을 --kpcr Option 을 통해 인자로 넣어 주어야 한다.

위 그림에서 볼 수 있듯이 KPCR 에 따라 다른 timers 의 목록을 볼 수 있으며, 이는 각 Processor 가 자신만의 timers 의 set 을 가지고 있기 때문이다.

자세한 설명은 아래의 주소를 통해 확인할 수 있다.

<http://mnin.blogspot.com/2011/10/aint-nuthin-but-ktimer-thing-baby.html>

Miscellaneous

strings

<decimal_offset>:<string> line 형식으로 된 주어진 Image 와 File 에서 해당 문자열을 찾을 수 있는 Process 와 Virtual Address 를 출력해 주는 명령어 이다. 해당 명령의 입력은 Sysinternals 의 Strings Utility 또는 유사한 Tool 들의 출력(offset:string)이며, 입력 Offset 은 File/Image 의 시작 지점의 물리 Offset 이다.

Sysinternals 의 Strings 는 Linux/MAC 에서 Wine 을 이용하여 사용 가능하며, 이에 대한 출력은 File 형식으로 Volatility 에게 Redirect 되어야 한다. GNU 의 Strings 명령어를 사용한다면 -td Option 을 통해 Offset 을 10 진수로 출력 가능하다.

Windows

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

Linux/Mac

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

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

File/Image 를 Sysinternals 의 Strings Utility 를 수행하면 많은 시간이 걸리며, -q 와 -o 는 Header 출력의 생략(-q)과 각 줄의 Offset(-o)을 구하기 위해 필수적인 Option 이다.

```
File Offset Hit Text
```

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

```
[snip]
```

```
$ file export.txt
```

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

```
$ xxd export.txt |less
```

```
0000000: fffe 3100 3100 3400 3900 3200 3300 3a00  ..1.1.4.9.2.3.:
```

```
[snip]
```

EnCase 에서 Export 한 Keyword 와 Offset 의 경우 UTF-16 with a BOM of (U+FEFF)로 되어 있기 때문에 약간의 변형을 수행한 후 Strings Plugin 을 사용할 수 있다.

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

Windows 의 경우 Notepad 에서 ANSI 로 저장하면 되지만, Linux 에서는 iconv 명령을 이용하여 Encoding 을 변환한다. 이 때 마지막에 빈 줄이 포함되어서는 안 된다.

```
$ ./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]
```

위 그림을 통해 Encoding 변환 전 후의 결과가 다른 것을 확인 가능하며, --output-file Option 을 통해 File 로 저장이 가능하다.

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

Default 로 PsActiveProcessHead 를 이용해 Double-linked 형태로 Process 들을 출력하며, -S Option 을 통해 Hidden Process 들도 출력 가능하다.

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

EPROCESS Offset 도 지원해 준다.

```
$ 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] C00
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://cr1.microsoft.com/pki/cr1/products/WinIntPCA.cr10U

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

[snip]
03678031 [360:2089816113 596:2089816113 620:2089816113 672:2089816113 684:2089816113 844:2089816113 932:2089816113 1064:208
:2089816113 1896:2089816113 1904:2089816113 1756:2089816113 512:2089816113 1372:2089816113 560:2089816113] A$9B
```

위 그림은 Strings 를 통해 확인한 PIDs/kernel Reference 들 이다.

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

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

Strings 의 출력을 통해 어떠한 Process 가 Memory 에서 수상한 문자열을 가지고 있는지를 확인하여 시야를 좁힐 수 있다. 또한, 정규 표현식, Script 등을 통하여 특정 Pattern 에 대응하는 문자열들을 추출할 수 있다.

volshell

WinDbg 와 비슷한 Interface 를 제공해 주며, Memory Image 를 대화형식으로 조사 할 수 있다. volshell 에서 제공하는 기능은 아래와 같으며, IPython 과 연동이 된다.

- List processes
- Switch into a process's context
- Display types of structures/objects
- Overlay a type over a given address
- Walk linked lists
- Disassemble code at a given address

volshell 을 실행하면 위 그림과 같은 화면이 출력되며, 아래는 volshell 을 이용하여 Explorer.exe 에 대해 살펴보는 예이다.

```
>>> 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
wuauclt.exe         1896       876    0x83ec3238
VMwareTray.exe      2144      1880    0x83f028d8
VMwareUser.exe      2156      1880    0x8d7893b0
[snip]

>>> 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]

>>> 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 00 04 00 00 ff ff 00 00  MZ.....
779f0010 b8 00 00 00 00 00 00 00 40 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 00 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....$......
```

```
>>> 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]

>>> 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, dis 명령어는 특정 주소 공간을 인자로 전달 가능하며, 사용하는 주소 공간에 따라 다른 결과를 출력하는 것을 확인 가능하다.

psscan, connsnscan 등과 같은 Scan 명령어를 사용할 경우 오탐을 발견할 수 있는데, 이를 물리적 Offset 으로 출력해 준다. 이를 확인 하려면 Hex Viewer 로 Image 를 연 후에 직접 Raw Byte 를 보아야 한다. 그러나 Volshell 을 아래와 같이 확인 가능하다.

우선 Physical Address 공간을 Instantiate 한다.

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

오탐된 EPROCESS 가 Physical Offset 0x433308 지점에 있다고 추측되면 아래와 같은 명령어를 입력한다.

```
>>> 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
...
```

다른 예로 Kernel Memory 를 Memory Dump 내의 Physical Offset 로 출력할 수 있다.

```
$ echo "hex(self.addr_space.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'
```

자세한 설명은 아래의 주소를 통해 확인할 수 있다.

<http://moyix.blogspot.com/2008/08/introducing-volshell.html>

bioskbd

Memory 의 BIOS 영역에서 Key 입력을 읽을 때 사용하는 명령어 이며, HP, Intel, Lenovo 의 BIOS 와 SafeBoot, TrueCrypt, BitLocker 에 입력한 비밀 번호를 확인 가능하다. 참고로 Memory Dump Tool 에 따라 BIOS 영역을 포함할 수도 하지 않을 수도 있다.

자세한 설명은 아래의 주소를 통해 확인할 수 있다.

<http://computer.forensikblog.de/en/2009/04/reading-passwords-from-the-keyboard-buffer.html#more>

<http://blog.sharpesecurity.com/2011/05/09/duplicating-volatility-bioskbd-command-function-on-live-windows-systems/>

http://www.ivizsecurity.com/research/preboot/preboot_whitepaper.pdf

Write-up Using Volatility

1. Nuit du hack 2011 Forensic 100

“ On a dumper la RAM d'une machine sur laquelle tournait un serveur VNC.
Le but est de recuperer le mot de passe de ce serveur.
* * *

We have dumped the RAM of a Machine on which was running a VNC server.
The goal is to get the password of that VNC server.

해당 문제를 다운 받으면 dump.raw File 이 주어지며 우선 해당 Memory Dump 를 추출할 당시에 어떠한 Process 가 수행되고 있었는지를 확인하기 위해 psscan 을 수행하였다.

```
c:\volatility>vol.py -f c:\Users\Deok9\Desktop\dump.raw psscan
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump (ImportError: No module
named Crypto.Hash)
Offset      Name                PID    PPID    PDB          Time created      T
ime exited
-----
0x01fb0020  ctfmon.exe          1664   1580  0x06f10140  2011-03-10 13:02:35
0x01fce938  lsass.exe            696    632  0x06f100a0  2011-03-10 13:02:30
0x01fd1500  svchost.exe          928    684  0x06f100e0  2011-03-10 13:02:31
0x01fe8020  wscntfy.exe          532   1020  0x06f10200  2011-03-10 13:02:59
0x01ff4020  svchost.exe          1020    684  0x06f10100  2011-03-10 13:02:31
0x0201d7e8  spoolsv.exe          1472    684  0x06f10180  2011-03-10 13:02:34
0x02192020  alg.exe              500    684  0x06f101e0  2011-03-10 13:02:58
0x021ea980  winvnc4.exe          1696    684  0x06f10240  2011-03-10 13:09:47
```

문제에서 언급했듯이, "winvnc4.exe"가 동작되고 있었고 vnc server 의 키를 찾기 위해 registry 를 확인하였다. registry 외에도 File 이나 Memory 상에 남아 있을 수 있지만, 이러한 접속 정보의 경우 registry 에 설정 정보로 남아 있는 경우도 많으므로 제일 먼저 수행하였다.

```

c:\volatility>vol.py -f c:\Users\Deok9\Desktop\dump.raw hivelist
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump (ImportError: No module
named Crypto.Hash)
Virtual      Physical      Name
0x8066e904   0x0066e904   [no name]
0xe1809008   0x08bfd008   WDevice\HarddiskVolume1\Documents and Settings\eleve\Loc
al Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1986008   0x09f7e008   WDevice\HarddiskVolume1\Documents and Settings\eleve\NTU
SER.DAT
0xe17a9768   0x08a48768   WDevice\HarddiskVolume1\Documents and Settings\LocalServ
ice\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe179b758   0x08a40758   WDevice\HarddiskVolume1\Documents and Settings\LocalServ
ice\NTUSER.DAT
0xe1770008   0x085d6008   WDevice\HarddiskVolume1\Documents and Settings\NetworkSe
rvice\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe175fb60   0x08410b60   WDevice\HarddiskVolume1\Documents and Settings\NetworkSe
rvice\NTUSER.DAT

```

HKLM\Software 의 Hive File 인 \Windows\system32\config\software File 의 주소를 확인할 수 있었으며, 어떠한 Software 들의 설정 정보들이 들어 있는지 확인하여 보았다.

```

c:\volatility>vol.py -f c:\Users\Deok9\Desktop\dump.raw --hive-offset 0xe13ffb60
printkey
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump (ImportError: No module
named Crypto.Hash)
Legend: (S) = Stable (U) = Volatile

-----
Registry: User Specified
Key name: $$$PROTO.HIU (S)
Last updated: 2011-03-10 13:09:47

Subkeys:
(S) C07ft5Y
(S) Classes
(S) Clients
(S) Gemplus
(S) Microsoft
(S) ODBC
(S) Policies
(S) Program Groups
(S) RealUNC
(S) Schlumberger
(S) Secure
(S) Windows 3.1 Migration Status

```

Hive File 에 저장된 Subkey 들 중 RealVNC 를 발견할 수 있었고, 해당 Subkey 에서 WinVNC4 의 Key 들을 출력하여 보면 아래와 같이 Password, SecurityTypes 등이 나오게 되었다.

```

c:\volatility>vol.py -f c:\Users\Deok9\Desktop\dump.raw printkey --hive-offset 0xe13ffb60 --key "RealUNC\WinUNC4"
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump (ImportError: No module named Crypto.Hash)
Legend: (S) = Stable (U) = Volatile

-----
Registry: User Specified
Key name: WinUNC4 (S)
Last updated: 2011-03-10 13:10:51

Subkeys:

Values:
REG_BINARY Password : (S)
0000 DA 6E 31 84 95 77 AD 6B .n1..w.k

REG_SZ SecurityTypes : (S) UncAuth
REG_SZ ReverseSecurityTypes : (S) None
REG_DWORD QueryConnect : (S) 0

```

해당 Password 의 경우 Triple-DES 로 암호화 되어 저장되기 때문에 Decrypt 를 수행 하여야 올바른 Key 값이 나오게 된다.

[+] 해당 문제에서 사용된 Volatility Plugin

Psscan : vol.py -f dump.raw psscan 을 통해 Memory dump로부터 Process 목록 확인

Hivelist : vol.py -f dump.raw hivelist 를 통해 Memory dump로부터 hive file 의 주소 및 경로 확인

Printkey : vol.py -f dump.raw printkey --hive-offset 0xe13ffb60(HKLM\Software vaddr)를 통해 HKLM\Software Registry key 의 subkeys 확인

-K/--key option : Printkey 에서 특정 문자열을 포함하는 Key 의 값만 출력

2. Nuit du hack 2011 Forensic 300

해당 문제를 다운 받으면 DumpRAM_CTF.vmem File 이 주어지며 Virtual Machine 에서 Memory File 을 준 것을 확인 가능하다. 우선 해당 Memory Dump 의 OS Profile 을 확인해 보기 위해 imageinfo 명령을 수행하였다.

```
c:\volatility>vol.py -f c:\Users\Deok9\Desktop\DumpRAM_CTF.vmem imageinfo
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump <ImportError: No module
named Crypto.Hash>
    Suggested Profile(s) : Win7SP1x86, Win7SP0x86
                          AS Layer1 : JKIA32PagedMemoryPae <Kernel AS>
                          AS Layer2 : FileAddressSpace <C:\Users\Deok9\Desktop\DumpRA
M_CTF.vmem>
                          PAE type : PAE
                          DTB : 0x185000
                          KDBG : 0x8276ebe8L
                          KPCR : 0x8276fc00L
                          KUSER_SHARED_DATA : 0xffdf0000L
    Image date and time : 2011-03-31 14:41:00
    Image local date and time : 2011-03-31 14:41:00
    Number of Processors : 1
    Image Type :
```

해당 Profile 을 확인 한 후 Forensic 100 문제와 마찬가지로 실행중인 Process 목록을 확인하기 위해 psscan 명령을 수행하였다.

| | | | | | |
|------------|----------------|------|------|------------|---------------------|
| 0x1e923d40 | csrss.exe | 352 | 332 | 0x1ee13040 | 2011-03-31 14:38:19 |
| 0x1e98f098 | winlogon.exe | 392 | 332 | 0x1ee130c0 | 2011-03-31 14:38:20 |
| 0x1e9a8b20 | services.exe | 416 | 340 | 0x1ee13080 | 2011-03-31 14:38:20 |
| 0x1e9b4030 | lsass.exe | 424 | 340 | 0x1ee130e0 | 2011-03-31 14:38:21 |
| 0x1e9b6030 | lsmon.exe | 432 | 340 | 0x1ee13100 | 2011-03-31 14:38:21 |
| 0x1e9d9bc0 | svchost.exe | 556 | 416 | 0x1ee13120 | 2011-03-31 14:38:23 |
| 0x1ea01d40 | smss.exe | 216 | 4 | 0x1ee13020 | 2011-03-31 14:38:10 |
| 0x1ebf1b70 | SearchIndexer. | 1528 | 416 | 0x1ee13180 | 2011-03-31 14:39:05 |
| 0x1eea7030 | explorer.exe | 2004 | 1992 | 0x1ee13220 | 2011-03-31 14:38:55 |
| 0x1f088d40 | wininit.exe | 340 | 296 | 0x1ee130a0 | 2011-03-31 14:38:19 |
| 0x1f1ced40 | dwm.exe | 1080 | 800 | 0x1ee13240 | 2011-03-31 14:38:31 |
| 0x1fcce030 | nc.exe | 1720 | 1392 | 0x1ee133c0 | 2011-03-31 14:40:41 |
| 0x1ff97830 | cmd.exe | 1392 | 2004 | 0x1ee13280 | 2011-03-31 14:39:39 |
| 0x1ffef898 | System | 4 | 0 | 0x00185000 | 2011-03-31 14:38:10 |

"nc.exe"는 일반적으로 쓰지 않는 Program 이기 때문에 악의적인 File 이라 생각하고 관련 정보를 더 찾기 위해 netscan 명령을 수행하였다.

```
0x1fc49560 TCPv4 0.0.0.0:445 0.0.0.0:0 LISTENIN
G 4 System
0x1fc49560 TCPv6 :::445 :::0 LISTENIN
G 4 System
0x1f086df8 TCPv4 192.168.163.216:49158 88.190.230.12:48625 ESTABLIS
HED 1720 nc.exe
0x1e608d30 UDPv4 0.0.0.0:5355 ***
1140 svchost.exe 2011-03-31 14:38:55
0x1e60d378 UDPv4 0.0.0.0:0 ***
1140 svchost.exe 2011-03-31 14:38:55
0x1e60d378 UDPv6 :::0 ***
1140 svchost.exe 2011-03-31 14:38:55
```

nc.exe 에서 88.190.230.12 로 통신을 하고 있는 것을 확인할 수 있으며, 어떠한 data 를 사용자 PC 에서 보냈는지 확인하기 전에 우선 memdump 명령을 수행하였다.

```
c:\volatility>vol.py -f c:\Users\Deok9\Desktop\DumpRAM_CTF.vmem memdump -p 1720
--dump-dir c:\Users\Deok9\Desktop --profile="Win7SP1x86"
Volatile Systems Volatility Framework 2.0
*** Failed to import volatility.plugins.registry.lsadump <ImportError: No module
named Crypto.Hash>
*****
Writing nc.exe [ 1720] to 1720.dmp
```

이제 해당 1720.dump File 을 strings 명령으로 88.190.230.12 IP 가 포함된 것을 확인하여 보았다.

```
[Deok9@MAC-MINI Desktop]$
MAC :)
strings 1720.dmp | grep -C 5 "88.190.230.12"
%SystemRoot%\system32\mswsock.dll
mvvI
,{M%
mvvI
,{M%
w CKM88.190.230.12
t:$Bf
euLSeu
`ahu
ahu@
`huh
--
windir=C:\Windows

Secret pass is H4x0r

Nice job !
The hash is *****

secte.server_of_dark_hamster.com
88.190.230.12
```


"Secret pass is H4x0r"를 88.190.230.12 의 48625 port 로 전송하면 정답 Hash 값을 전송해 준다.

[+] 해당 문제에서 사용된 Volatility Plugin

Imageinfo : vol.py -f DumpRam_CTF.vmem imageinfo 를 통해 Memory dump 로부터 해당 Image File 의 Profile 확인

Psscan : vol.py -f DumpRam_CTF.vmem psscan 을 통해 Memory dump 로부터 Process 목록 확인

Netscan : vol.py -f DumpRam_CTF.vmem netscan 를 통해 Memory dump 로부터 연결/종료된 Network Session 확인

Memdump : vol.py -f DumpRam_CTF.vmem memdump 를 통해 nc.exe 의 memory dump 수행

-p option : 해당 pid 에 해당하는 memory dump 만 수행

Comment

위처럼 Memory dump 에서 악의적인 행위 및 Password 를 추출 가능하며 Volatility 를 사용하면 훨씬 수월하게 분석을 진행할 수 있다.

현재 Volatility 는 Linux 와 MAC Memory 분석도 지원하고 있으며 계속 추진 중이라고 한다. 개인적으로 MAC Memory 분석은 아직은 "Volafox"가 더 괜찮다고 생각하지만 기존 Volatility 처럼 활발하게 진행된다면 무시할 것이 못되기 때문에 여러 가지 기대를 해본다.