black hat® EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

One-byte Modification for Breaking Memory Forensic Analysis

Takahiro Haruyama / Hiroshi Suzuki
Internet Initiative Japan Inc.

for submission

Summary

- Memory Forensics Overview
  - Memory Acquisition
  - Memory Analysis
- Previous Works: Anti Memory Forensics
- Proposed Anti Analysis Method
- Improvement Plans
- Wrap-up

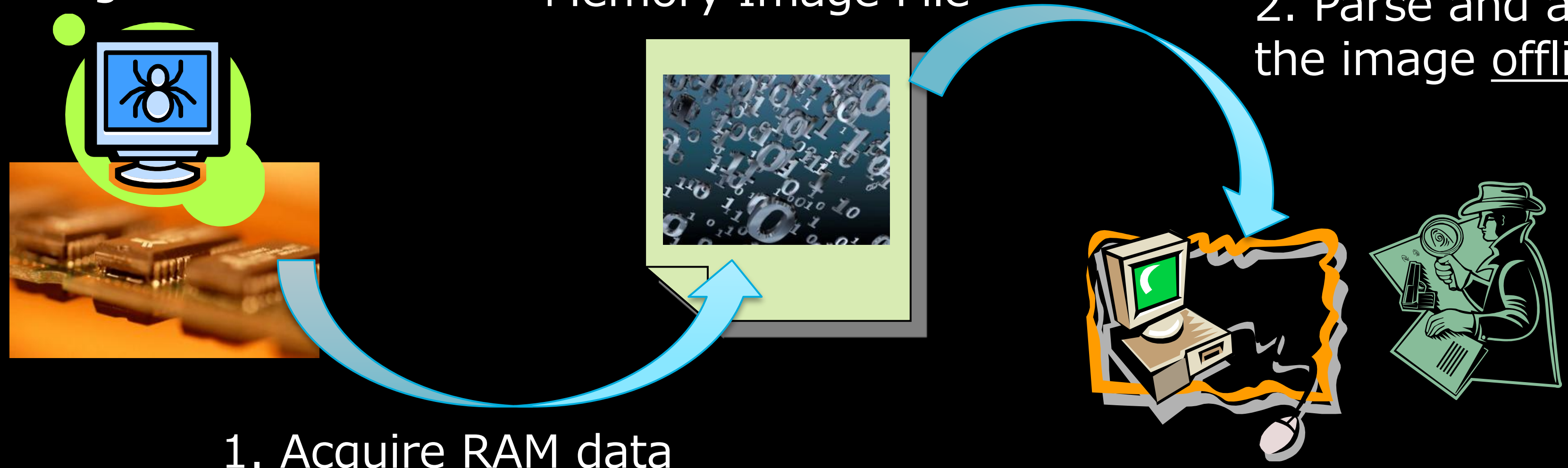
MEMORY FORENSICS OVERVIEW

black hat®
EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

What’s Memory Forensics?

- Analyzing volatile data is important to detect threats quickly
  - increasing amounts of disk data
  - anti disk forensic methods used by malwares
- Memory forensics became popular over the last few years
- 2 steps for memory forensics
  - memory acquisition and memory analysis



Target Machine

Memory Image File

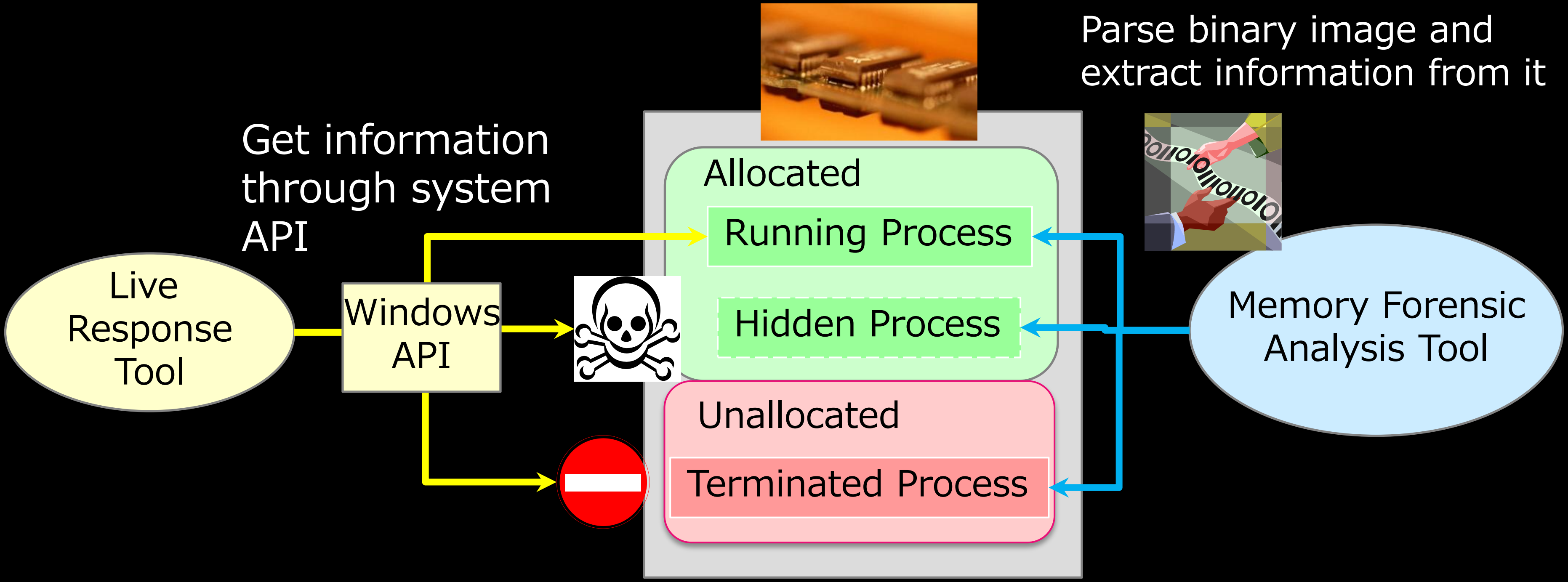
2. Parse and analyze the image offline

1. Acquire RAM data as an image file

Investigator’s Machine

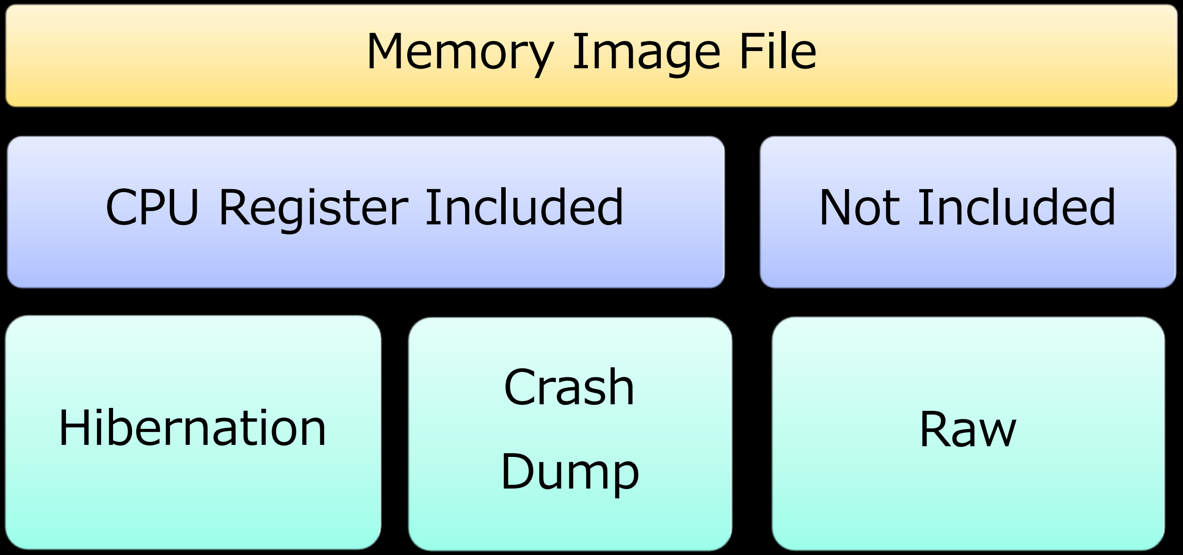
Why Memory Forensics?

- Offline parsing a memory image doesn’t use system APIs
- Memory forensics can get
  - unallocated data (e.g., terminated process)
  - data hidden by malware (e.g., hidden process)


Parse binary image and extract information from it
Get information through system API
Allocated
Running Process
Hidden Process
Unallocated
Terminated Process
Live Response Tool
Windows API
Memory Forensic Analysis Tool

Some Formats / Acquisiton Tools

- Raw Image Acquisition
  - HBGary FastDump Pro [1]
  - Guidance WinEn [2]
  - MoonSols Windd [3]
- Crash Dump Image Acquisition
  - MoonSols Windd
- Memory Image Conversion
  - MoonSols Windows Memory Toolkit [3]



Memory Image File

CPU Register Included

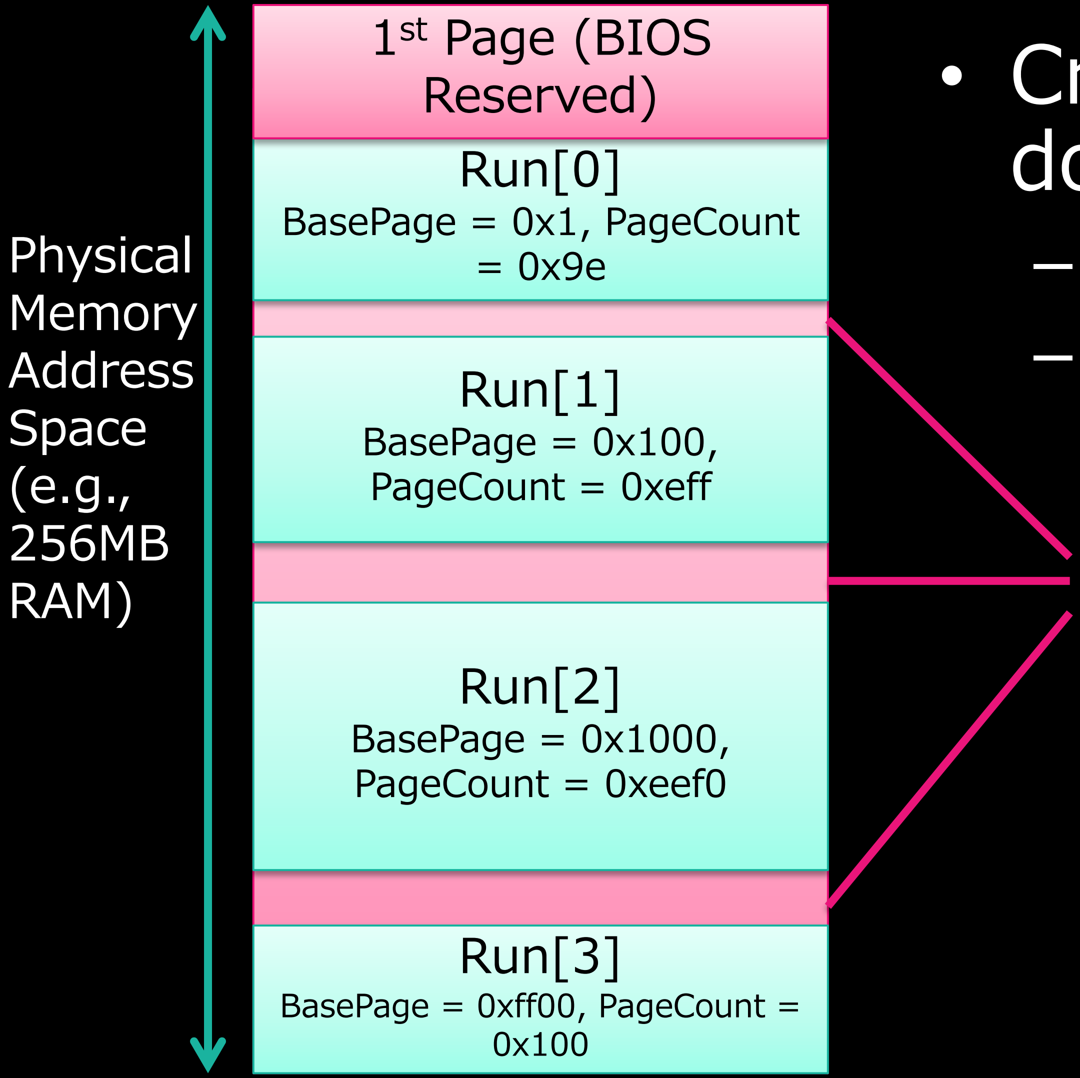
Not Included

Hibernation

Crash Dump

Raw

Difference between Raw Image and Crash Dump


Physical Memory Address Space (e.g., 256MB RAM)
1st Page (BIOS Reserved)
Run[0]
BasePage = 0x1, PageCount = 0x9e
Run[1]
BasePage = 0x100, PageCount = 0xeff
Run[2]
BasePage = 0x1000, PageCount = 0xeef0
Run[3]
BasePage = 0xff00, PageCount = 0x100

- Crash dump file doesn't include
  - 1st Page
  - Pages reserved by devices

Address Space Reserved by Devices (Not Included in crash dump)

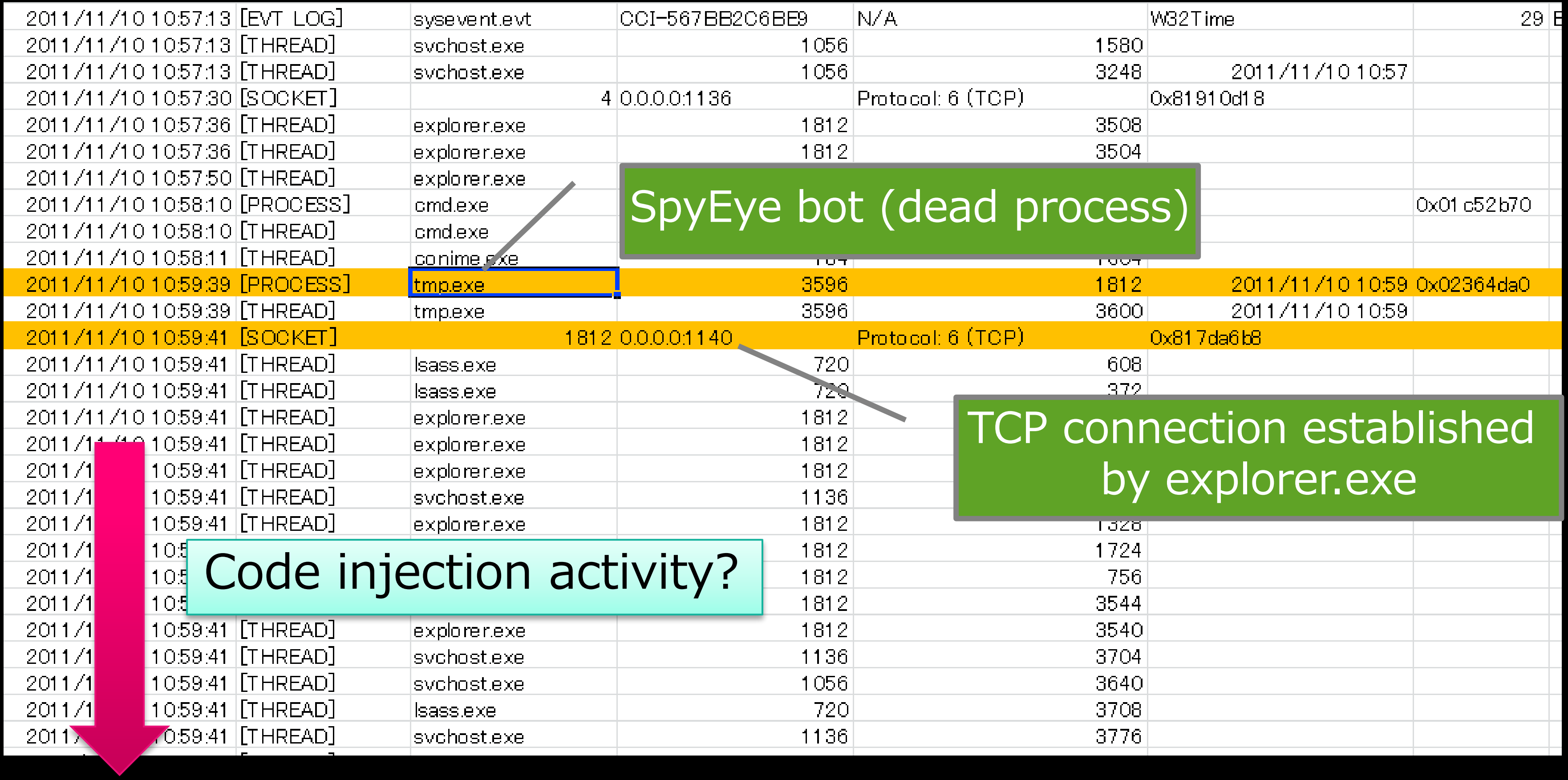
Evaluation of Memory Acquisition Tools

- Can raw image acquisition tools get 1st page and device-reserved pages? [4]
  - WinEn
  - Win32dd /c 0
    - Memory Content (/c) option
      - Caution: /c 0 option may cause BSOD on x64 machine

	WinEn	FDPro	Win32dd /c 0	Win32dd /c 1	Win32dd /c 2
1st Page	✓	✓	✓		✓
Device reserved pages	✓		✓		

Analysis Example: Making Object Creation Timeline

- Volatility Framework [5]
  - timeliner plugin [6]
    - used kernel objects (process/thread/socket)
    - event logs



2011/11/10 10:57:13	[EVT LOG]	sysevent.evt	CCI-567BB2C6BE9	N/A	W32Time	29	E
2011/11/10 10:57:13	[THREAD]	svchost.exe	1056	1580			
2011/11/10 10:57:13	[THREAD]	svchost.exe	1056	3248	2011/11/10 10:57		
2011/11/10 10:57:30	[SOCKET]	4	0.0.0.0:1136	Protocol: 6 (TCP)	0x81910d18		
2011/11/10 10:57:36	[THREAD]	explorer.exe	1812	3508			
2011/11/10 10:57:36	[THREAD]	explorer.exe	1812	3504			
2011/11/10 10:57:50	[THREAD]	explorer.exe	[illegible]	[illegible]			
2011/11/10 10:58:10	[PROCESS]	cmd.exe	[illegible]	[illegible]	[illegible]	0x01c52b70	
2011/11/10 10:58:10	[THREAD]	cmd.exe	[illegible]	[illegible]	[illegible]		
2011/11/10 10:58:11	[THREAD]	conime.exe	[illegible]	[illegible]			
2011/11/10 10:59:39	[PROCESS]	tmp.exe	3596	1812	2011/11/10 10:59	0x02364da0	
2011/11/10 10:59:39	[THREAD]	tmp.exe	3596	3600	2011/11/10 10:59		
2011/11/10 10:59:41	[SOCKET]	1812	0.0.0.0:1140	Protocol: 6 (TCP)	0x817da6b8		
2011/11/10 10:59:41	[THREAD]	lsass.exe	720	608			
2011/11/10 10:59:41	[THREAD]	lsass.exe	720	372			
2011/11/10 10:59:41	[THREAD]	explorer.exe	1812	[illegible]			
2011/11/10 10:59:41	[THREAD]	explorer.exe	1812	[illegible]			
2011/1 10:59:41	[THREAD]	explorer.exe	1812	[illegible]			
2011/1 10:59:41	[THREAD]	svchost.exe	1136	[illegible]			
2011/1 10:59:41	[THREAD]	explorer.exe	1812	1328			
2011/1 10:5	[illegible]	[illegible]	1812	1724			
2011/1 10:5	[illegible]	[illegible]	1812	756			
2011/1 10:5	[illegible]	[illegible]	1812	3544			
2011/1 10:59:41	[THREAD]	explorer.exe	1812	3540			
2011/1 10:59:41	[THREAD]	svchost.exe	1136	3704			
2011/1 10:59:41	[THREAD]	svchost.exe	1056	3640			
2011/1 10:59:41	[THREAD]	lsass.exe	720	3708			
2011 0:59:41	[THREAD]	svchost.exe	1136	3776			

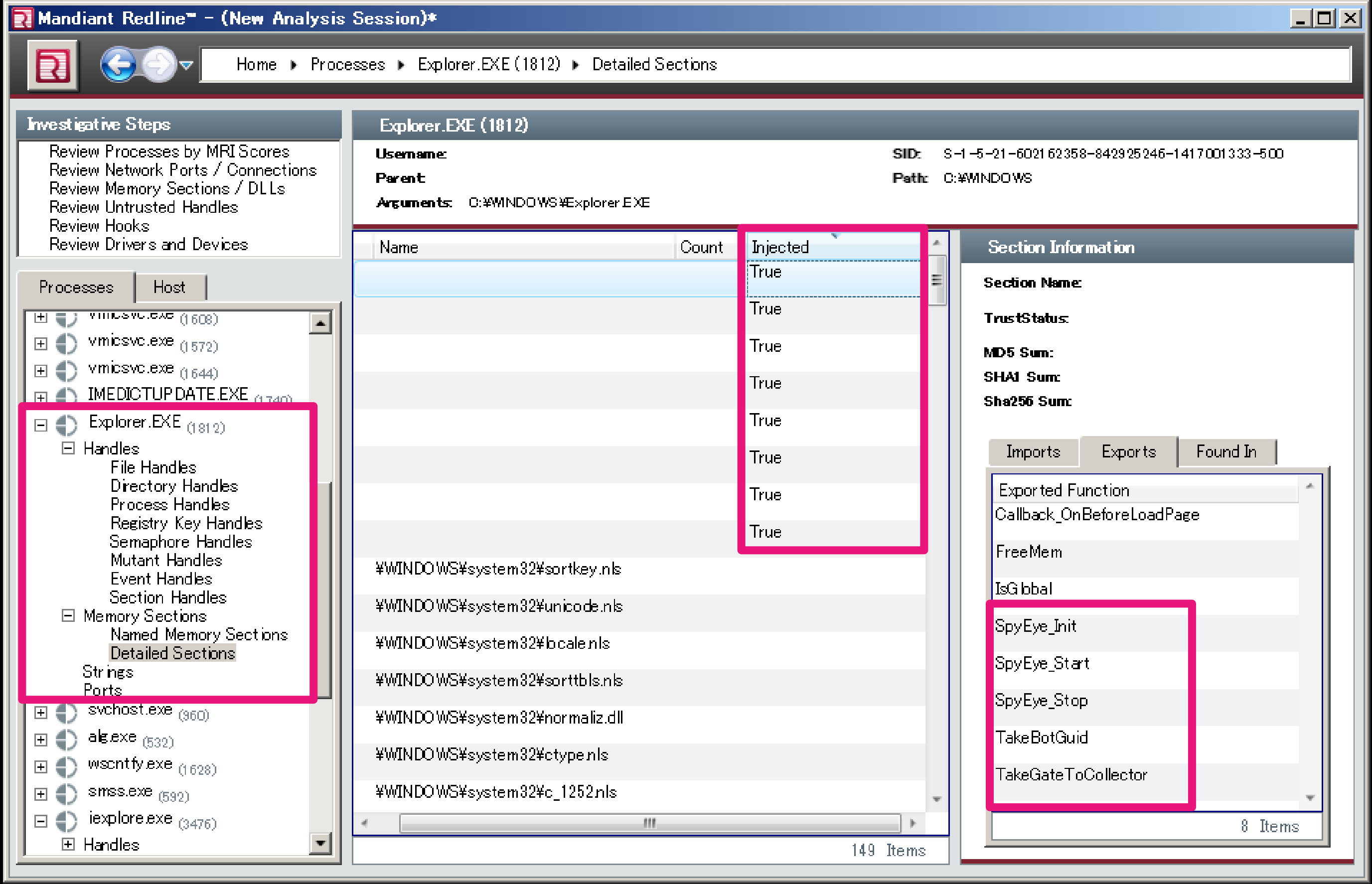
SpyEye bot (dead process)

TCP connection established by explorer.exe

Code injection activity?

Analysis Example: Detecting Code Injection

- Detecting code injection
  - Volatility Framework malfind
  - EnCase EnScript [7] VadDump
  - Mandiant Redline [8] (GUI front-end for Memoryze [9])
- The tools check protection flag of Virtual Address Descriptor



Mandiant Redline™ - (New Analysis Session)*
Home ▸ Processes ▸ Explorer.EXE (1812) ▸ Detailed Sections
Investigative Steps
Processes
Host
Imports
Exports
Found In

Investigative Steps:
- Review Processes by MRI Scores
- Review Network Ports / Connections
- Review Memory Sections / DLLs
- Review Untrusted Handles
- Review Hooks
- Review Drivers and Devices

Processes:
- vmicsvc.exe (1608)
- vmicsvc.exe (1572)
- vmicsvc.exe (1644)
- IMEDICTUPDATE.EXE (1740)
- Explorer.EXE (1812)
  - Handles
    - File Handles
    - Directory Handles
    - Process Handles
    - Registry Key Handles
    - Semaphore Handles
    - Mutant Handles
    - Event Handles
    - Section Handles
  - Memory Sections
    - Named Memory Sections
    - Detailed Sections
  - Strings
  - Ports
- svchost.exe (960)
- alg.exe (532)
- wscntfy.exe (1628)
- smss.exe (592)
- iexplore.exe (3476)
  - Handles

Explorer.EXE (1812)

Username:

Parent:

Arguments: C:¥WINDOWS¥Explorer.EXE

SID: S-1-5-21-602162358-842925246-1417001333-500

Path: C:¥WINDOWS

Name	Count	Injected
		True
		True
		True
		True
		True
		True
		True
		True
¥WINDOWS¥system32¥sortkey.nls		
¥WINDOWS¥system32¥unicode.nls		
¥WINDOWS¥system32¥locale.nls		
¥WINDOWS¥system32¥sorttbls.nls		
¥WINDOWS¥system32¥normaliz.dll		
¥WINDOWS¥system32¥ctype.nls		
¥WINDOWS¥system32¥c_1252.nls		

149 Items

Section Information

Section Name:

TrustStatus:

MD5 Sum:

SHA1 Sum:

Sha256 Sum:

Exported Function
Callback_OnBeforeLoadPage
FreeMem
IsGlobal
SpyEye_Init
SpyEye_Start
SpyEye_Stop
TakeBotGuid
TakeGateToCollector

8 Items

Comparison of Memory Analysis Tools

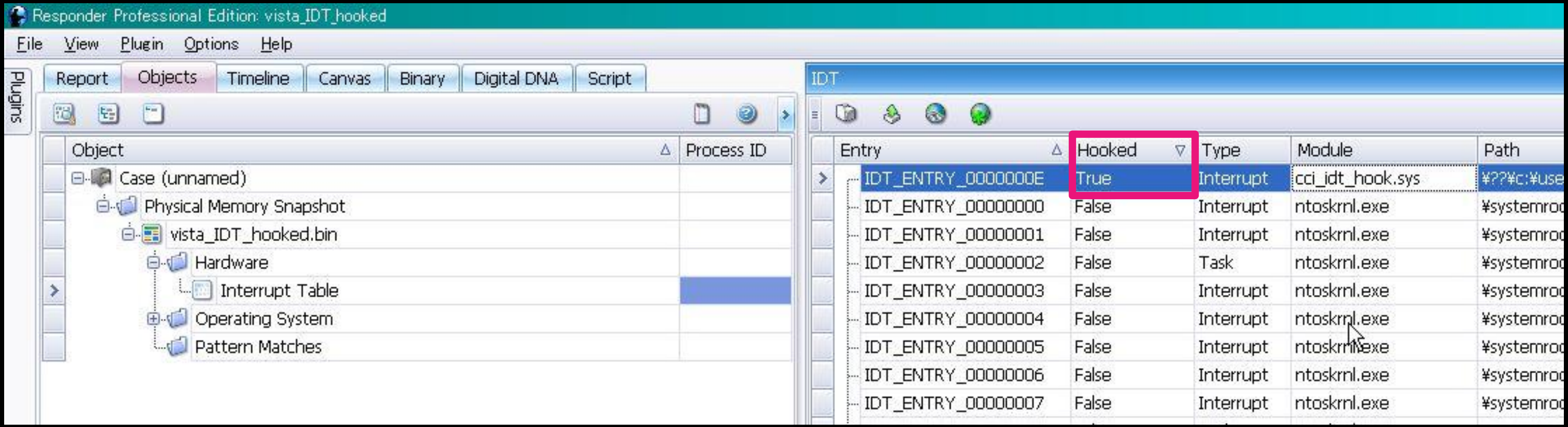
	Mandiant Redline (Memoryze)	HBGary Responder	Volatility Framework 2.0	EnCase EnScirpt
Supported Windows OS	All	All	XP/Vista/7/ 2003/2008	XP/7/2003/ 2008
Supported Image Format	Raw	Raw	Raw Crash dump Hibernation	Raw Crash dump
Supported CPU Architecture	Intel x86 AMD x64	Intel x86 AMD x64	Intel x86	Intel x86 AMD x64
Extracting dead process/closed connection	No	No	Yes	Yes
Note	Malware Risk Index, MemD5	Digital DNA, code graphing	Open source, rich plugins	Multilingual search, Entropy

PREVIOUS WORKS:
ANTI MEMORY FORENSICS

black hat®
EUROPE
March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands
HELDHAFTIG VASTBERADEN BARMHARTIG

Anti Acquisition Methods: Shadow Walker [10]

- ShadowWalker is proposed by Sherri Sparks and Jamie Butler to hide malicious memory regions
  - Installed page fault handler makes de-synchronized DTLB/ITLB
    - data access -> random garbage data
    - execute access -> rootkit code
- Memory acquisition tools cannot prevent ShadowWalker from hiding memory pages
  - But Analysis tools can detect the IDT hooking

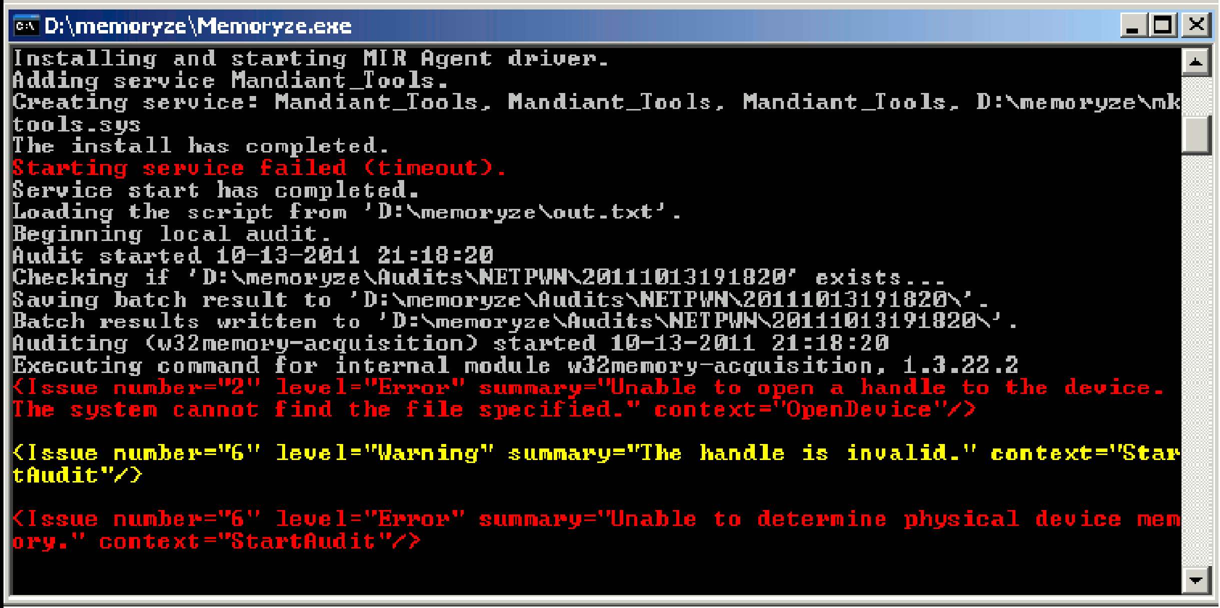


Object	Process ID
Case (unnamed)	
Physical Memory Snapshot	
vista_IDT_hooked.bin	
Hardware	
Interrupt Table	
Operating System	
Pattern Matches	

Entry	Hooked	Type	Module	Path
IDT_ENTRY_0000000E	True	Interrupt	cci_idt_hook.sys	¥??¥c:¥use
IDT_ENTRY_00000000	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000001	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000002	False	Task	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000003	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000004	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000005	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000006	False	Interrupt	ntoskrnl.exe	¥systemro
IDT_ENTRY_00000007	False	Interrupt	ntoskrnl.exe	¥systemro

Anti Acquisition Methods: Meterpreter Anti Memory Forensics Script [11]

- Proof of concept script
  - killing specified processes or preventing driver loadings with the aim of memory acquisition failure
- Very easy to implement
  - The evasion is also easy (e.g., random name)
  - Preventing driver loadings has an impact on the running system



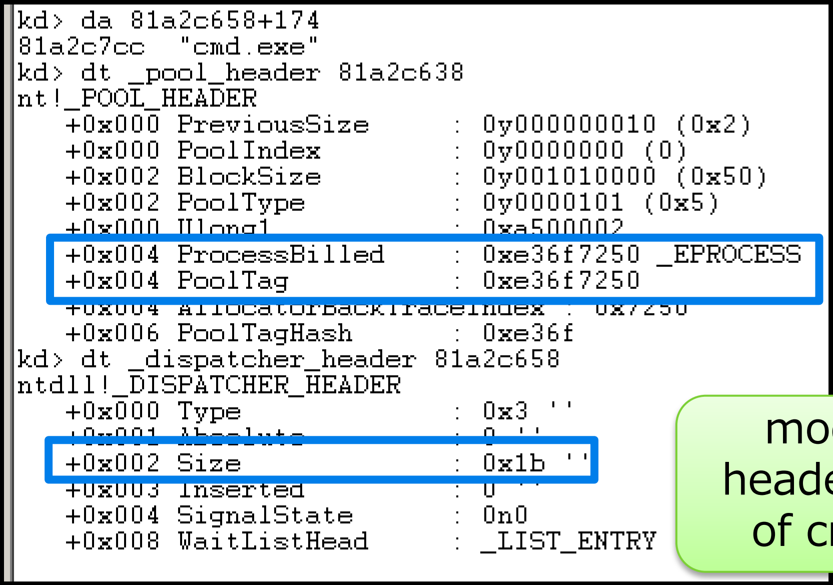
```
Installing and starting MIR Agent driver.
Adding service Mandiant_Tools.
Creating service: Mandiant_Tools, Mandiant_Tools, Mandiant_Tools, D:\memoryze\mk
tools.sys
The install has completed.
Starting service failed (timeout).
Service start has completed.
Loading the script from 'D:\memoryze\out.txt'.
Beginning local audit.
Audit started 10-13-2011 21:18:20
Checking if 'D:\memoryze\Audits\NETPWN\20111013191820' exists...
Saving batch result to 'D:\memoryze\Audits\NETPWN\20111013191820\'.
Batch results written to 'D:\memoryze\Audits\NETPWN\20111013191820\'.
Auditing (w32memory-acquisition) started 10-13-2011 21:18:20
Executing command for internal module w32memory-acquisition, 1.3.22.2
<Issue number="2" level="Error" summary="Unable to open a handle to the device.
The system cannot find the file specified." context="OpenDevice"/>

<Issue number="6" level="Warning" summary="The handle is invalid." context="Star
tAudit"/>

<Issue number="6" level="Error" summary="Unable to determine physical device mem
ory." context="StartAudit"/>
```

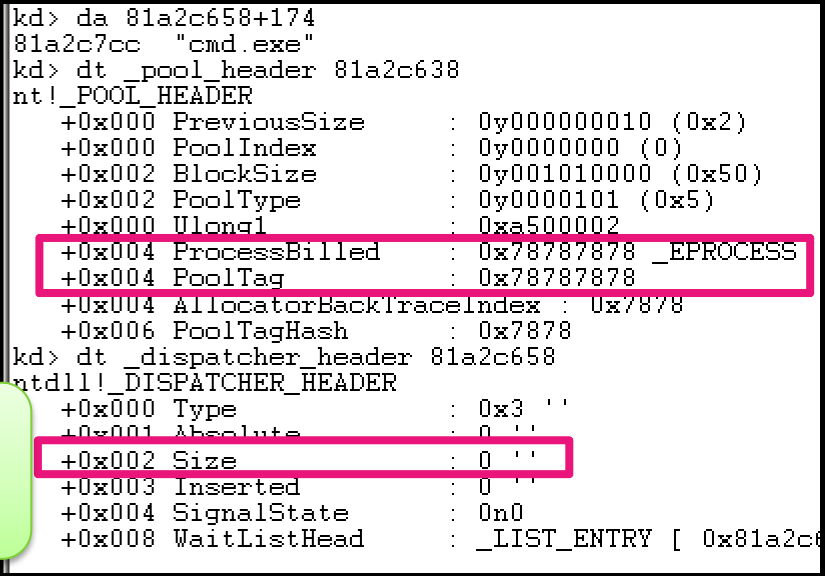
Anti Analysis Method: Anti Object Carving

- Object carving is one technique to extract kernel object information
  - e.g., process object (_EPROCESS)
    - PTFinder: Type/Size in _DISPATCHER_HEADER
    - Volatility Framework: PoolTag in _POOL_HEADER
- Brendan Dolan-Gavitt et al. warned an attacker could change the values to hide a specified object [12]
  - Instead, they proposed robust signatures causing BSOD or functionality failures if the values are changed



```
kd> da 81a2c658+174
81a2c7cc  "cmd.exe"
kd> dt _pool_header 81a2c638
nt!_POOL_HEADER
   +0x000 PreviousSize     : 0y000000010 (0x2)
   +0x000 PoolIndex        : 0y0000000 (0)
   +0x002 BlockSize        : 0y001010000 (0x50)
   +0x002 PoolType         : 0y0000101 (0x5)
   +0x000 Ulong1           : 0xa5000002
   +0x004 ProcessBilled    : 0xe36f7250 _EPROCESS
   +0x004 PoolTag          : 0xe36f7250
   +0x004 AllocatorBackTraceIndex : 0x7250
   +0x006 PoolTagHash      : 0xe36f
kd> dt _dispatcher_header 81a2c658
ntdll!_DISPATCHER_HEADER
   +0x000 Type             : 0x3 ''
   +0x001 Absolute         : 0 ''
   +0x002 Size             : 0x1b ''
   +0x003 Inserted         : 0 ''
   +0x004 SignalState      : 0n0
   +0x008 WaitListHead     : _LIST_ENTRY
```

modifying header values of cmd.exe

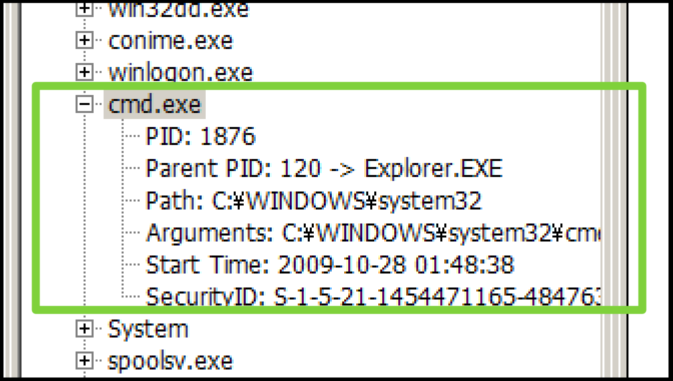


```
kd> da 81a2c658+174
81a2c7cc  "cmd.exe"
kd> dt _pool_header 81a2c638
nt!_POOL_HEADER
   +0x000 PreviousSize     : 0y000000010 (0x2)
   +0x000 PoolIndex        : 0y0000000 (0)
   +0x002 BlockSize        : 0y001010000 (0x50)
   +0x002 PoolType         : 0y0000101 (0x5)
   +0x000 Ulong1           : 0xa5000002
   +0x004 ProcessBilled    : 0x78787878 _EPROCESS
   +0x004 PoolTag          : 0x78787878
   +0x004 AllocatorBackTraceIndex : 0x7878
   +0x006 PoolTagHash      : 0x7878
kd> dt _dispatcher_header 81a2c658
ntdll!_DISPATCHER_HEADER
   +0x000 Type             : 0x3 ''
   +0x001 Absolute         : 0 ''
   +0x002 Size             : 0 ''
   +0x003 Inserted         : 0 ''
   +0x004 SignalState      : 0n0
   +0x008 WaitListHead     : _LIST_ENTRY [ 0x81a2c6
```

Anti Analysis Method:
Anti Object Carving (Cont.)

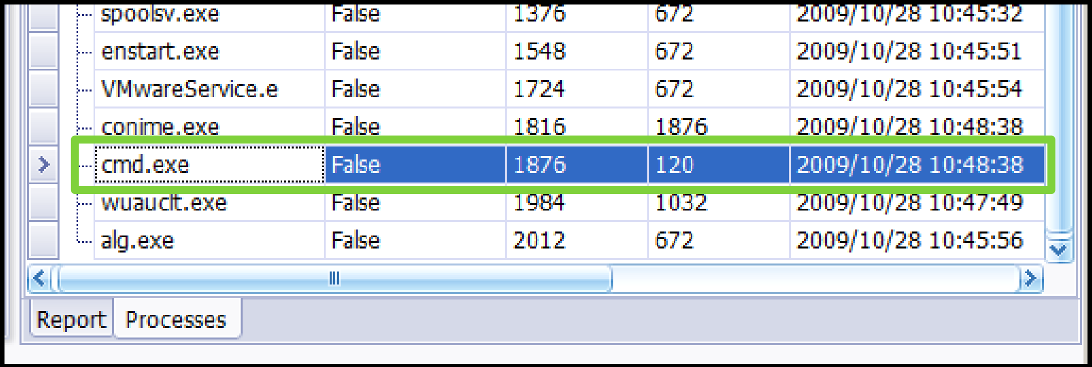
- Closed-source analysis tools can find the hidden process
  - How do they find it?
- Other than object carving, there are several key operations for analyzing memory image
  - The operations are robust?
- Let’s check it!

Memoryze



```
⊞ win32dd.exe
⊞ conime.exe
⊞ winlogon.exe
⊟ cmd.exe
    PID: 1876
    Parent PID: 120 -> Explorer.EXE
    Path: C:¥WINDOWS¥system32
    Arguments: C:¥WINDOWS¥system32¥cm
    Start Time: 2009-10-28 01:48:38
    SecurityID: S-1-5-21-1454471165-48476
⊞ System
⊞ spoolsv.exe
```

HBGary Responder



spoolsv.exe	False	1376	672	2009/10/28 10:45:32
enstart.exe	False	1548	672	2009/10/28 10:45:51
VMwareService.e	False	1724	672	2009/10/28 10:45:54
conime.exe	False	1816	1876	2009/10/28 10:48:38
cmd.exe	False	1876	120	2009/10/28 10:48:38
wuauclt.exe	False	1984	1032	2009/10/28 10:47:49
alg.exe	False	2012	672	2009/10/28 10:45:56

PROPOSED ANTI ANALYSIS METHOD

black hat®
EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

HELDHAFTIG VASTBERADEN BARMHARTIG

Abstract of Proposed Method

- Researched implementations of three major tools
  - Volatility Framework 2.0
  - Mandiant Memoryze 2.0
  - HBGary Responder Community Edition 2.0
- Found three operations executed in memory analysis include a few unconsidered assumptions
  - Proposed method modifies one-byte of data related to the operations
    - The data is defined as “Abort Factor”
  - It can’t hide specific objects, but can abort analyses
  - No impact on the running system
    - No BSOD, no errors for a few days to 2 weeks

Sensitive Three Operations in Memory Analysis

- Virtual address translation in kernel space
- Guessing OS version and Architecture
- Getting kernel objects
  - traversing linked lists or binary trees
  - object carving

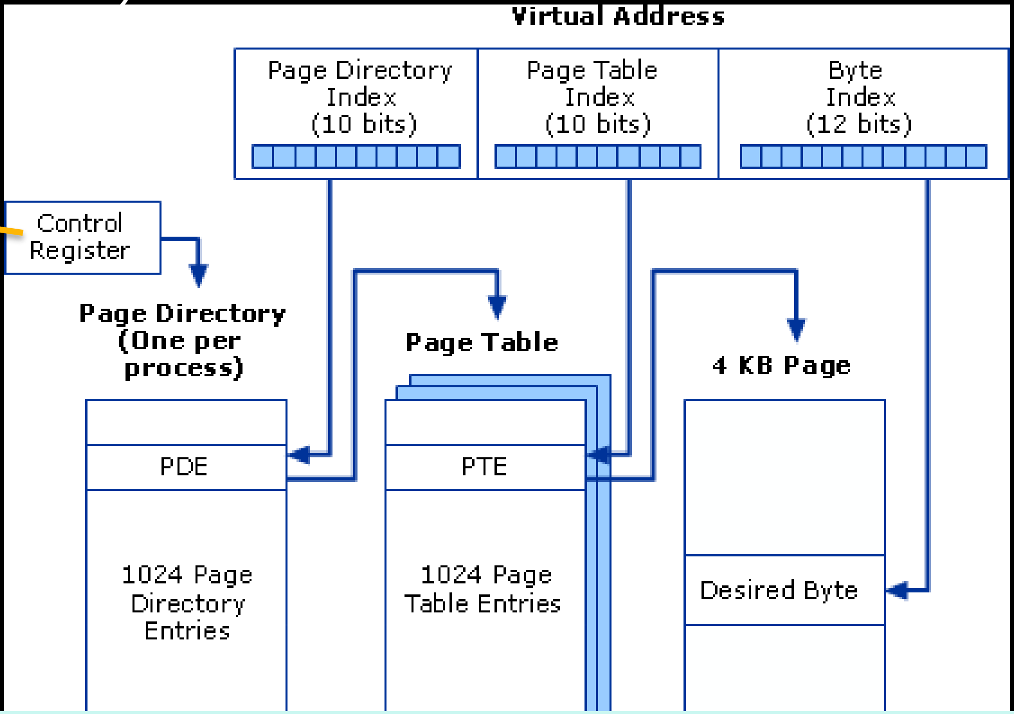
Sensitive Three Operations in Memory Analysis

- Virtual address translation in kernel space
- Guessing OS version and Architecture
- Getting kernel objects
  - traversing linked lists or binary trees
  - object carving

Virtual Address Translation in Kernel Space

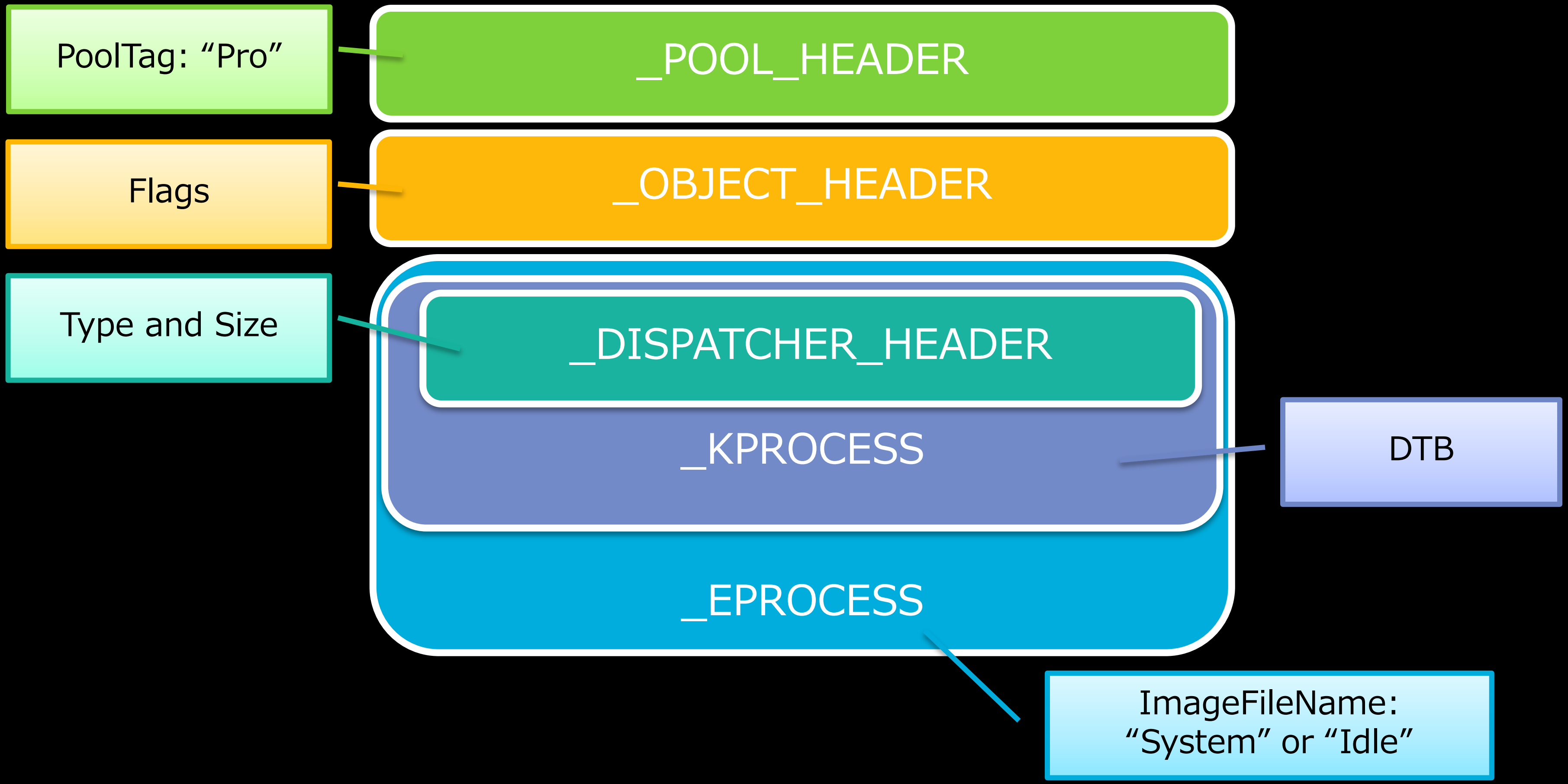
- OS switches its context by loading Directory Table Base (DTB) of each process
  - DTB is stored in each process object (_EPROCESS)
- Initially, analysis tools must get DTB value for kernel space
- Two processes have the kernel DTB
  - PsInitialSystemProcess (System process)
  - PsIdleProcess (Idle process)

OS loads
<u>Directory Table Base</u>
(Start physical address for
address translation)
into Control Register (CR3)


Virtual Address
Page Directory Index (10 bits)
Page Table Index (10 bits)
Byte Index (12 bits)
Control Register
Page Directory (One per process)
Page Table
4 KB Page
PDE
PTE
1024 Page Directory Entries
1024 Page Table Entries
Desired Byte

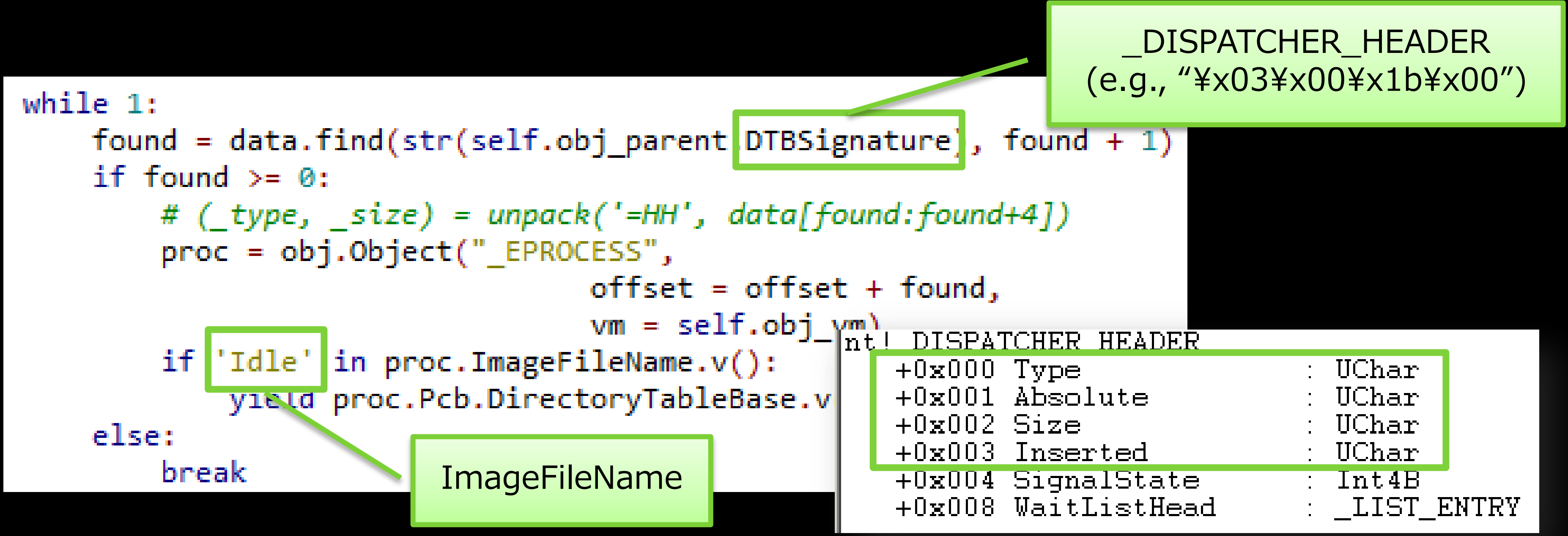
x86 Address Translation - How PAE X86 Works
http://technet.microsoft.com/en-us/library/cc736309(WS.10).aspx

Virtual Address Translation in Kernel Space: Process Object Structure



Virtual Address Translation in Kernel Space: Volatility Framework

- Search _DISPATCHER_HEADER to get _EPROCESS
- Check whether the ImageFileName is “Idle”
  - If the process is Idle, get DTB value in _KPROCESS



_DISPATCHER_HEADER (e.g., “¥x03¥x00¥x1b¥x00”)

```
while 1:
    found = data.find(str(self.obj_parent.DTBSignature), found + 1)
    if found >= 0:
        # (_type, _size) = unpack('=HH', data[found:found+4])
        proc = obj.Object("_EPROCESS",
                             offset = offset + found,
                             vm = self.obj_vm)
        if 'Idle' in proc.ImageFileName.v():
            yield proc.Pcb.DirectoryTableBase.v
    else:
        break
```

```
nt!_DISPATCHER_HEADER
   +0x000 Type             : UChar
   +0x001 Absolute         : UChar
   +0x002 Size             : UChar
   +0x003 Inserted         : UChar
   +0x004 SignalState      : Int4B
   +0x008 WaitListHead     : _LIST_ENTRY
```

ImageFileName

Virtual Address Translation in Kernel Space: Mandiant Memoryze

- Search “System” to find ImageFileName in _EPROCESS of PsInitialSystemProcess
- Validate by using _DISPATCHER_HEADER in the _KPROCESS
  - All _DISPATCHER_HEADER patterns are checked

OS version	**_DISPATCHER_HEADER Byte Sequence**
XP 32bit	03 00 1B 00
2003 32bit	03 00 1E 00
2003 64bit	03 00 2E 00
Vista 32bit	03 00 20 00
Vista 64bit	03 00 30 00
7 32bit	03 00 26 00
7 64bit	03 00 58 00

Virtual Address Translation in Kernel Space: Mandiant Memoryze (Cont.)

- Validate by using the following values
  - Flags in _OBJECT_HEADER
    - The distance between PoolTag and _EPROCESS is calculated according to the value
  - PoolTag in _POOL_HEADER
    - Search PoolTag from _EPROCESS position and check whether the search hit offset is equal to the calculated distance
- If all data is valid, get the DTB value

Virtual Address Translation in Kernel Space: HBGary Responder

- Search _DISPATCHER_HEADERs to get _EPROCESS
- Get DTB value from the result and validate it
- Responder seems to be equipped with the algorithm guessing kernel DTB
  - If DTBs of PsInitialSystemProcess and PsIdleProcess are not found, a guessed DTB value is used

Virtual Address Translation in Kernel Space: Related Data

Tool	Related Data	Abort Factor	Remarks
Volatility Framework	_DISPATCHER_HEADER	X	PsIdleProcess
	ImageFileName in _EPROCESS	X	
Mandiant Memoryze	_DISPATCHER_HEADER	X	PsInitialSystemProcess
	PoolTag in _POOL_HEADER	X	
	Flags in _OBJECT_HEADER	X	
	ImageFileName in _EPROCESS	X	
HBGary Responder	_DISPATCHER_HEADER		original guessing algorithm

Sensitive Three Operations in Memory Analysis

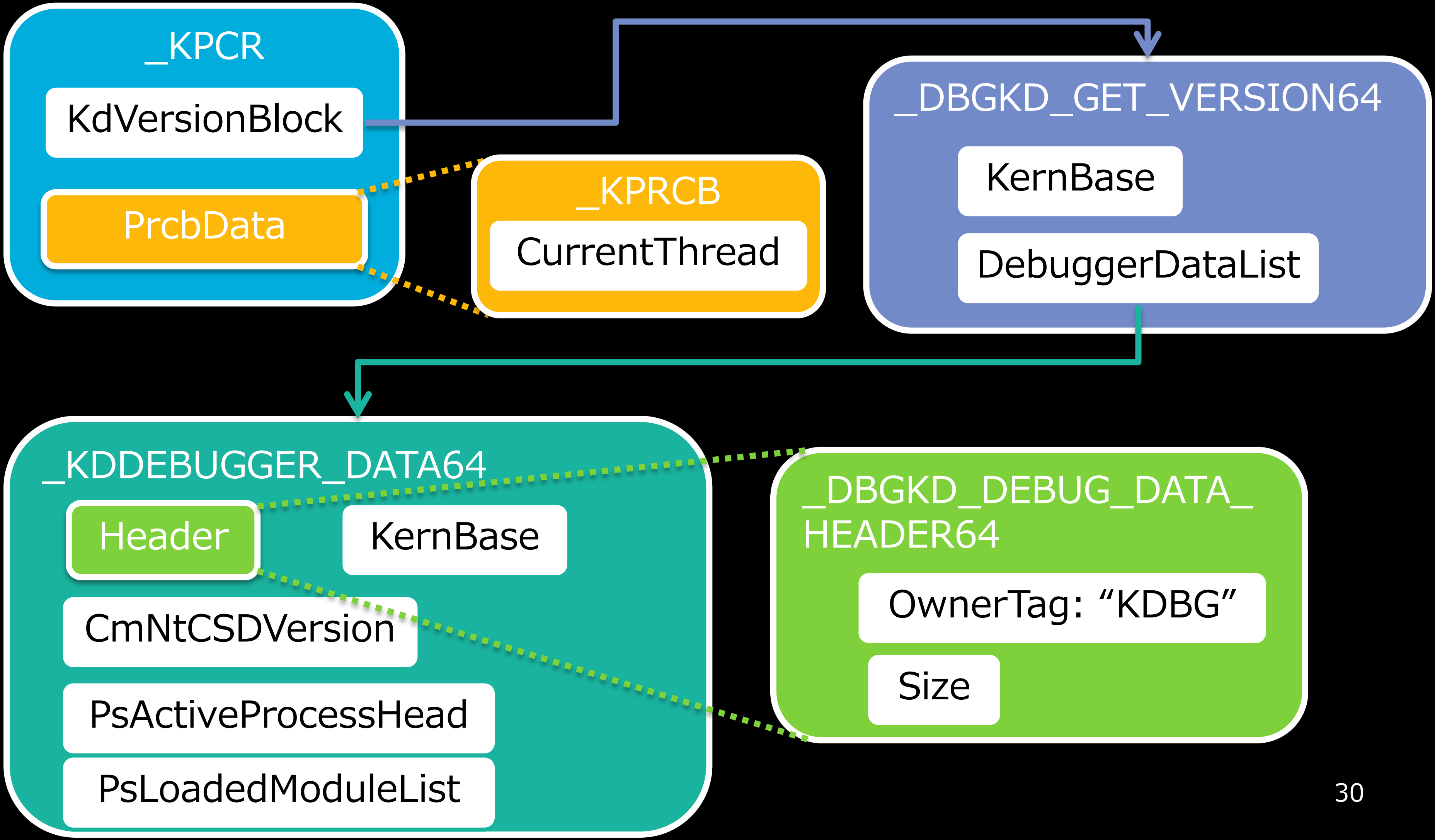
- Virtual address translation in kernel space
- Guessing OS version and Architecture
- Getting Kernel Objects
  - traversing linked lists or binary trees
  - object carving

Guessing OS version and Architecture

- Size and definition of kernel data structures differ according to
  - OS version (e.g., XP SP2/SP3, 7 SP0/SP1)
  - architecture (x86 and x64)
- All analysis tools guess the version using debug structures

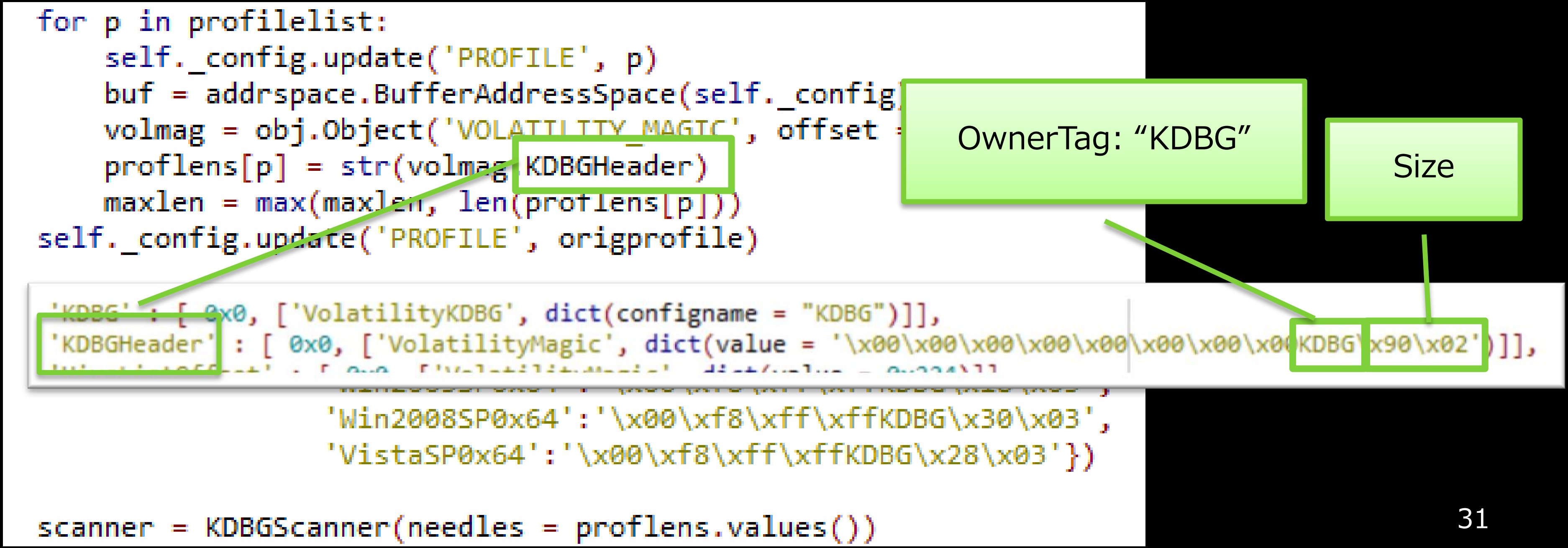
OS version	**_EPROCESS size (bytes)**
Windows XP SP3 32bit	0x260
Windows 7 SP0 32bit	0x2C0
Windows 7 SP0 64bit	0x4D0
Windows Vista SP2 32bit	0x270
Windows Vista SP2 64bit	0x3E8

Guessing OS version and Architecture: Debug Structures and Key Values



Guessing OS version and Architecture: Volatility Framework

- Users must specify OS version and Architecture
  - e.g., --profile=WinXPSP2x86
- If the version is unknown, imageinfo command can guess it
  - scan _DBGKD_DEBUG_DATA_HEADER64 [13]



```
for p in profilelist:
    self._config.update('PROFILE', p)
    buf = addrspace.BufferAddressSpace(self._config
    volmag = obj.Object('VOLATILITY_MAGIC', offset =
    proflens[p] = str(volmag.KDBGHeader)
    maxlen = max(maxlen, len(proflens[p]))
self._config.update('PROFILE', origprofile)
```

```
'KDBG' : [ 0x0, ['VolatilityKDBG', dict(configname = "KDBG")]],
'KDBGHeader' : [ 0x0, ['VolatilityMagic', dict(value = '\x00\x00\x00\x00\x00\x00\x00\x00KDBG\x90\x02')]],
```

```
                  'Win2008SP0x64':'\x00\xf8\xff\xffKDBG\x30\x03',
                  'VistaSP0x64':'\x00\xf8\xff\xffKDBG\x28\x03'})

scanner = KDBGScanner(needles = proflens.values())
```

Guessing OS version and Architecture: Mandiant Memoryze

- Supposedly determine OS and architecture based on _DISPATCHER_HEADER
- Validate them by using an offset value of ImageFileName in _EPROCESS

OS version	offset value of ImageFileName
XP 32bit	0x174
2003 32bit SP0	0x154
2003 32bit SP1/SP2	0x164
XP/2003 64bit	0x268
Vista 32bit	0x14C
Vista 64bit	0x238
7 32bit	0x16C
7/2008 64bit	0x2E0

Guessing OS version and Architecture: Mandiant Memoryze (Cont.)

- Try to translate a virtual address of ThreadListHead in _KPROCESS
  - If possible, the OS version and architecture are correct
- Get SP version from CmNtCSDVersion in _KDDEBUGGER_DATA64

Guessing OS version and Architecture: HBGary Responder

- Get KernBase value
  - _DBGKD_GET_VERSION64 or _KDDEBUGGER_DATA64
- Validate the PE header signatures
  - DOS header “MZ” and NT header “PE”
- Get OS version
  - OperatingSystemVersions in Optional Header
    - e.g., Windows7
      - MajorOperatingSystemVersion=6
      - MinorOperatingSystemVersion=1
- Get more specific version
  - TimeDataStamp in File header

Guessing OS version and Architecture: Related Data

Tool	Related Data	Abort Factor	Remarks
Volatility Framework	_DBGKD_DEBUG_DATA_HEADER64	X	
Mandiant Memoryze	_DISPATCHER_HEADER	X	PsInitialSystemProcess
	offset value of ImageFileName	X	
	ThreadListHead in _KPROCESS		
	CmNtCSDVersion in _KDDEBUGGER_DATA64		
HBGary Responder	KernBase in _DBGKD_GET_VERSION64 or _KDDEBUGGER_DATA64		PE Header of Windows kernel
	PE header signatures “MZ”/”PE”		
	OperatingSystemVersion in Optional Header	X	
	TimeDataStamp in File Header		

Sensitive Three Operations in Memory Analysis

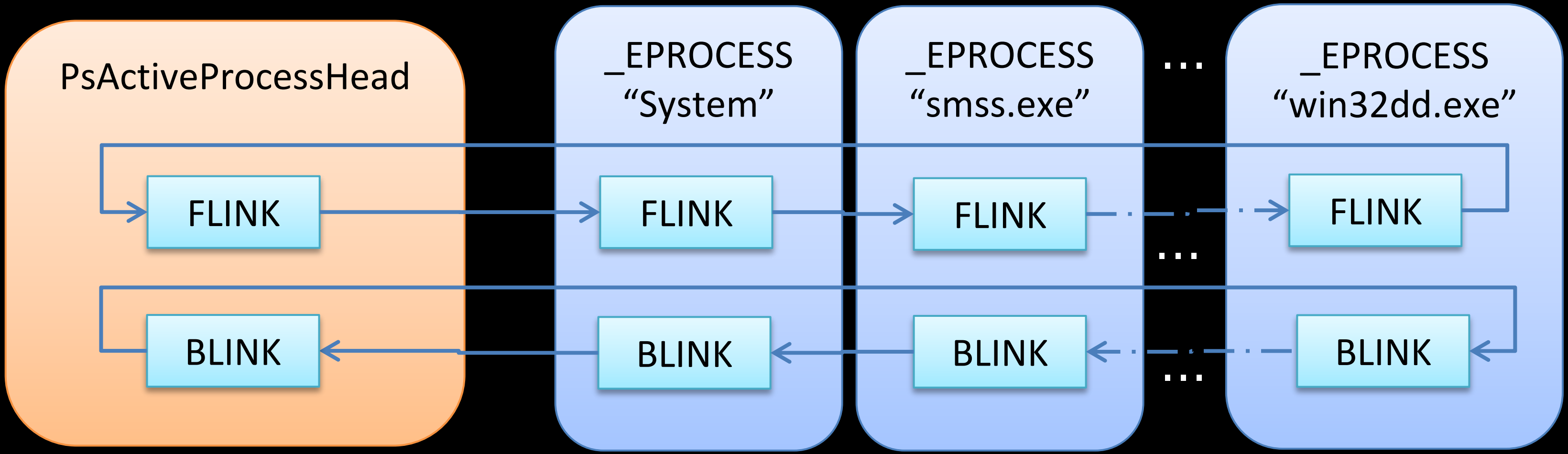
- Virtual address translation in kernel space
- Guessing OS version and Architecture
- Getting Kernel Objects
  - traversing linked lists or binary trees
  - object carving

Getting Kernel Objects

- Traversing linked lists or binary trees
  - Generally, use special lead/root addresses
    - PsActiveProcessHead for process list
    - PsLoadedModuleList for kernel module list
    - VadRoot for Virtual Address Descriptor tree
- Object carving
  - Generally, use fixed values in headers
    - _POOL_HEADER
    - _DISPATCHER_HEADER
- My research focused on getting _EPROCESS

Getting Kernel Objects: Process Linked List

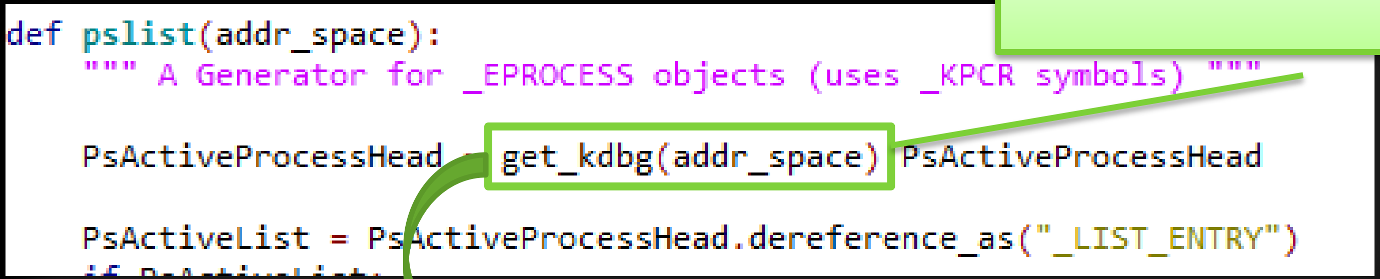
- Process list is two-way link
  - Each _EPROCESS includes ActiveProcessLinks
    - _LIST_ENTRY (Flink and Blink)
  - PsActiveProcessHead and PsInitialSystemProcess are bound up together



Getting Kernel Objects: Volatility Framework

- Traversing linked lists or binary trees
  - Search _DBGKD_DEBUG_DATA_HEADER64
  - get PsActiveProcessHead in _KDDEBUGGER_DATA64
- Object carving
  - use PoolTag in _POOL_HEADER

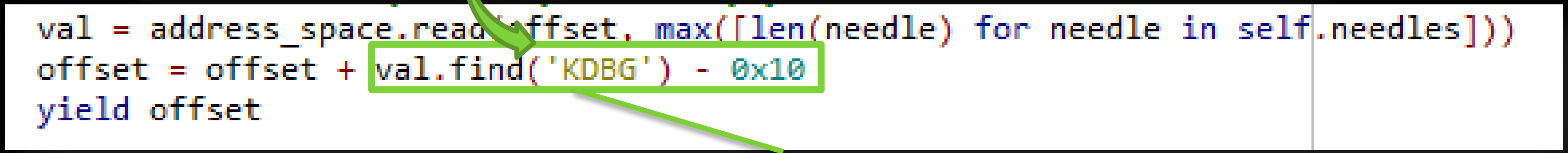
Executing KDBGScanner



```
def pslist(addr_space):
    """ A Generator for _EPROCESS objects (uses _KPCR symbols) """

    PsActiveProcessHead = get_kdbg(addr_space).PsActiveProcessHead

    PsActiveList = PsActiveProcessHead.dereference_as("_LIST_ENTRY")
    if PsActiveList:
```



```
val = address_space.read(offset, max([len(needle) for needle in self.needles]))
offset = offset + val.find('KDBG') - 0x10
yield offset
```

Getting _DBGKD_DEBUG_DATA_HEADER64
(= _KDDEBUGGER_DATA64) address

Getting Kernel Objects: Mandiant Memoryze

- Object carving
  - find _EPROCESS using address values
    - e.g.,
      - DTB is 0x20-bytes aligned
      - (Peb & 0x7ffd0000) == 0x7ffd0000
      - (ActiveProcessLinks.Flink & 0x80000000) == 0x80000000
  - similar to robust signatures proposed by Brendan Dolan-Gavitt et al. [12]

Getting Kernel Objects: HBGary Responder

- Traversing linked lists or binary trees
  - get CurrentThread in _KPRCB
  - get _EPROCESS from the thread
    - e.g., ApcState.Process in _KTHREAD (XP)
  - start to traverse process list from the _EPROCESS
    - “System” string is compared with ImageFileName of _EPROCESS
      - for identifying PsActiveProcessHead
      - for detecting hidden process

Getting Kernel Objects: Related Data

Tool	Related Data	Abort Factor	Remarks
Volatility Framework	_DBGKD_DEBUG_DATA_HEADER64	X	
	PsActiveProcessHead in _KDDEBUGGER_DATA64	X	
	PoolTag in _POOL_HEADER		
Mandiant Memoryze	address values in _EPROCESS (DTB, Peb, etc.)		
HBGary Responder	CurrentThread in _KPRCB		PsInitialSystemProcess
	_EPROCESS pointer in _KTHREAD		
	ImageFileName in _EPROCESS	X	

Abort Factors

Tool	Virtual Address Translation in Kernel Space	Guessing OS version and Architecture	Getting Kernel Objects
Volatility Framework	2 factors: _DISPATCHER_HEADER and ImageFileName (PsIdleProcess)	1 factor: _DBGKD_DEBUG_DATA_HEADER64	2 factors: _DBGKD_DEBUG_DATA_HEADER64 and PsActiveProcessHead
Mandiant Memoryze	4 factors: _DISPATCHER_HEADER, PoolTag, Flags and ImageFileName (PsInitialSystemProcess)	2 factors: _DISPATCHER_HEADER and offset value of ImageFileName (PsInitialSystemProcess)	None
HBGary Responder	None	1 factor: OperatingSystemVersion of kernel header	1 factor: ImageFileName (PsInitialSystemProcess)

Demo using PoC Driver (Video)

- Load a kernel driver into x86 XP VM
  - The driver modifies 1 byte of the following data
    - Size in _DISPATCHER_HEADER of PsIdleProcess
    - PoolTag in _POOL_HEADER of PsInitialSystemProcess
    - MajorOperatingSystemVersion in PE header of Windows kernel
- Check the modification using WinDbg
- Acquire the memory image using LiveCloudKd [14]
- Analysis using three tools

IMPROVEMENT PLANS

black hat®
EUROPE

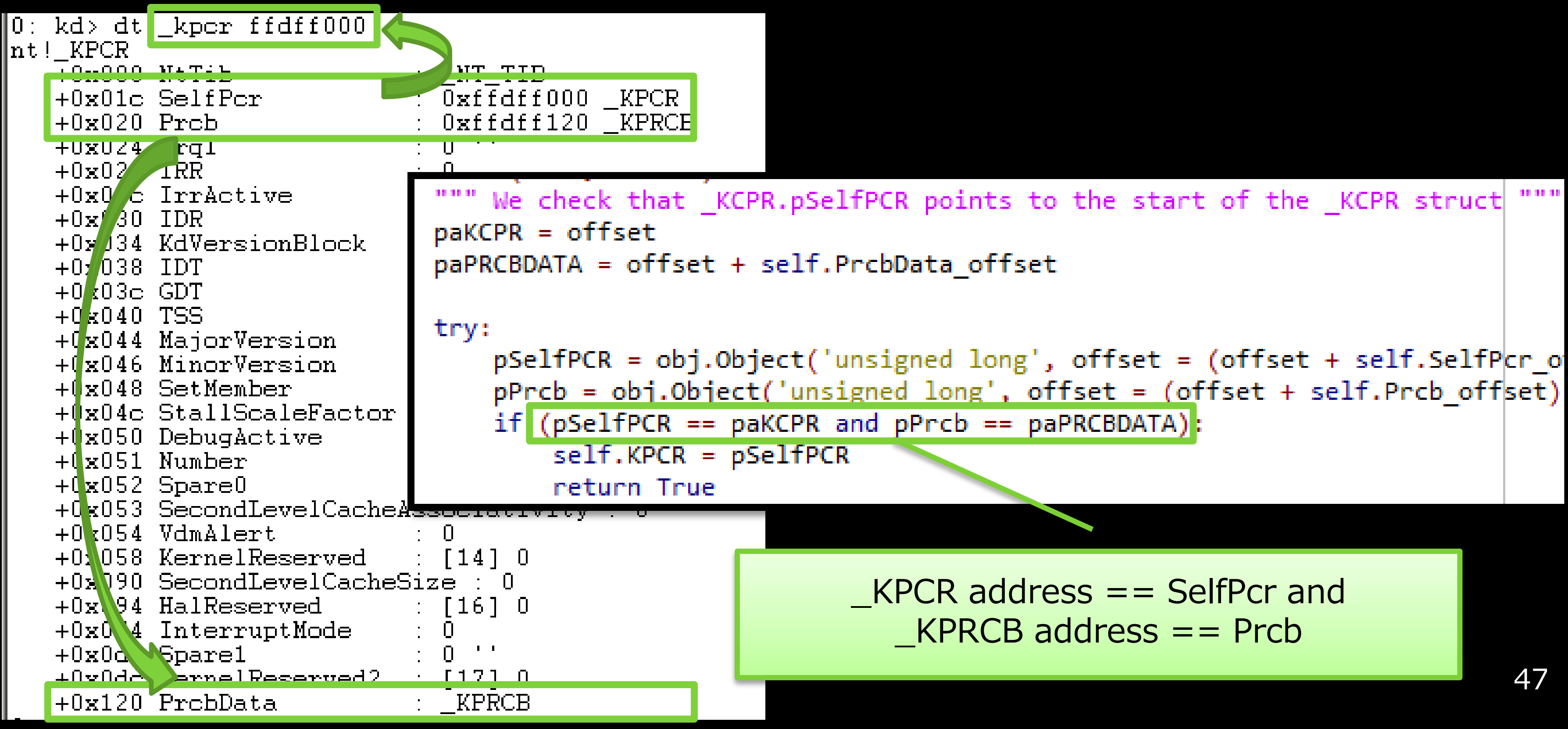
March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

Improvement Plans

- Guessing based on address values
- Minimum guessing
- Separating implementations to get kernel objects

Guessing Based on Address Values

- The modification of address values often causes BSOD or function failures
  - _EPROCESS object carving by Memoryze
  - _KPCR object carving by Volatility Framework [15]



```
0: kd> dt _kpcr ffdff000
nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x01c SelfPcr          : 0xffdff000 _KPCR
   +0x020 Prcb             : 0xffdff120 _KPRCB
   +0x024 Irql             : 0 ''
   +0x028 IRR              : 0
   +0x02c IrrActive
   +0x030 IDR
   +0x034 KdVersionBlock
   +0x038 IDT
   +0x03c GDT
   +0x040 TSS
   +0x044 MajorVersion
   +0x046 MinorVersion
   +0x048 SetMember
   +0x04c StallScaleFactor
   +0x050 DebugActive
   +0x051 Number
   +0x052 Spare0
   +0x053 SecondLevelCacheAssociativity : 0
   +0x054 VdmAlert         : 0
   +0x058 KernelReserved   : [14] 0
   +0x090 SecondLevelCacheSize : 0
   +0x094 HalReserved      : [16] 0
   +0x0d4 InterruptMode    : 0
   +0x0d8 Spare1           : 0 ''
   +0x0dc KernelReserved2  : [17] 0
   +0x120 PrcbData         : _KPRCB
```

```
""" We check that _KCPR.pSelfPCR points to the start of the _KCPR struct """
paKCPR = offset
paPRCBDATA = offset + self.PrcbData_offset

try:
    pSelfPCR = obj.Object('unsigned long', offset = (offset + self.SelfPcr_o
    pPrcb = obj.Object('unsigned long', offset = (offset + self.Prcb_offset)
    if (pSelfPCR == paKCPR and pPrcb == paPRCBDATA):
        self.KPCR = pSelfPCR
        return True
```

_KPCR address == SelfPcr and
_KPRCB address == Prcb

Minimum guessing (1)

- Support crash dump format
  - Register values cannot be modified

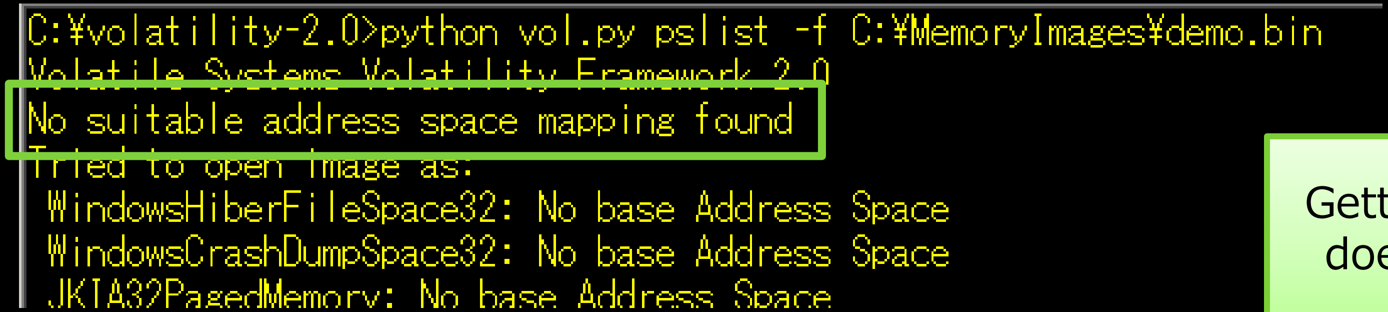
Data in crash dump header	**Extracted from (Win32dd implementation)**	**Abort Factor**
DTB	CR3 register	
OS version	nt!NtBuildNumber	X
PAE enabled	CR4 register	
PsActiveProcessHead	_KDDEBUGGER_DATA64	X
PsLoadedModuleList	_KDDEBUGGER_DATA64	X

Minimum guessing (2)

- Support argument passing options about DTB and OS version
  - Volatility Framework supports them
    - specify OS version by using “--profile” option
    - specify DTB value by using “--dtb” option

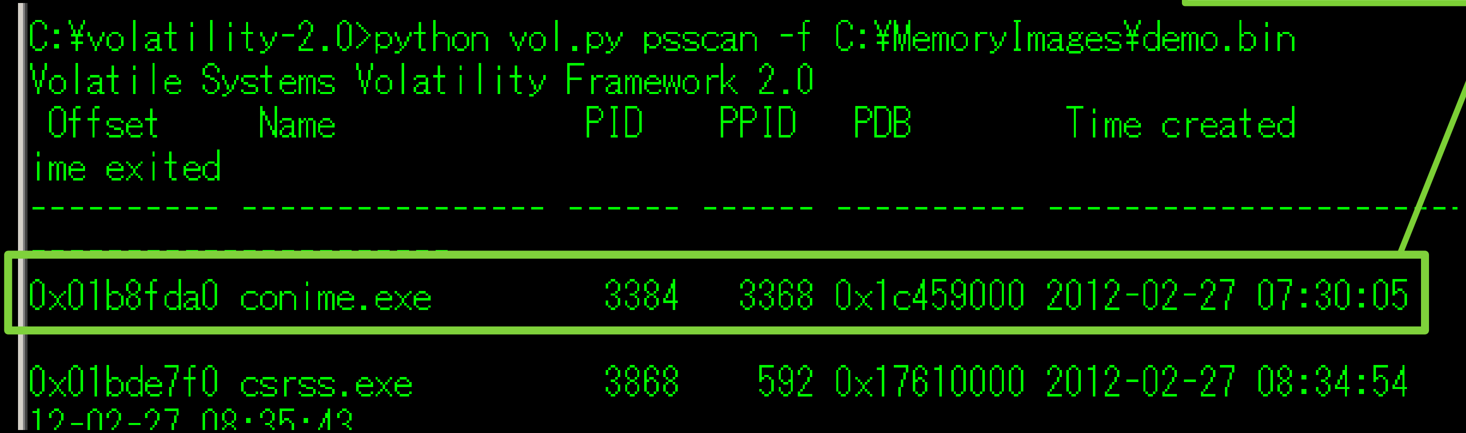
Separating implementations to get kernel objects

- If DTB value cannot be acquired, display the result minimally-extracted by object carving



```
C:¥volatility-2.0>python vol.py pslist -f C:¥MemoryImages¥demo.bin
Volatile Systems Volatility Framework 2.0
No suitable address space mapping found
Tried to open image as:
 WindowsHiberFileSpace32: No base Address Space
 WindowsCrashDumpSpace32: No base Address Space
 JKIA32PagedMemory: No base Address Space
```

Getting these information doesn’t need DTB value



```
C:¥volatility-2.0>python vol.py psscan -f C:¥MemoryImages¥demo.bin
Volatile Systems Volatility Framework 2.0
 Offset     Name             PID    PPID   PDB        Time created
ime exited
---------- ---------------- ------ ------ ---------- ------------------------
------------------------
0x01b8fda0 conime.exe         3384   3368 0x1c459000 2012-02-27 07:30:05

0x01bde7f0 csrss.exe          3868    592 0x17610000 2012-02-27 08:34:54
12-02-27 08:35:43
```

WRAP-UP

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

Wrap-up

- Proposed anti analysis method can abort memory analysis tools by modifying only one-byte
  - The method is effective for memory images of all OS versions and architectures
  - About the impact on the running system, long term evaluations may be needed
- I hope
  - Developers improve the implementations
  - Users figure out internals of memory analysis and deal with analysis errors

Questions?
(twitter: @cci_forensics)

Please complete the Speaker Feedback Surveys!

black hat®
EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

HELDHAFTIG VASTBERADEN BARMHARTIG

References

[1] HBGary FastDump Pro <http://www.hbgary.com/fastdump-pro>

[2] EnCase WinEn (build-in tool of EnCase) <http://www.guidancesoftware.com/>

[3] MoonSols Windows Memory Toolkit <http://www.moonsols.com/windows-memory-toolkit/>

[4] Reserved Address Space in Windows Physical Memory <http://cci.cocolog-nifty.com/blog/2011/02/device-reserved.html>

[5] Volatility Framework <https://www.volatilesystems.com/default/volatility>

[6] timeliner plugin <http://gleeda.blogspot.com/2011/09/volatility-20-timeliner-registryapi.html>

[7] Update: Memory Forensic EnScript <http://cci.cocolog-nifty.com/blog/2011/03/memory-forensic.html>

[8] Mandiant Redline <http://www.mandiant.com/products/free_software/redline/>

[9] Mandiant Memoryze <http://www.mandiant.com/products/free_software/memoryze/>

[10] "SHADOW WALKER" Raising The Bar For Rootkit <http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf>

[11] Meterpreter Anti Memory Forensics (Memoryze) Script <http://t0x1cs.blogspot.com/2012/02/meterpreter-anti-memory-forensics.html>

[12] Robust Signatures for Kernel Data Structures <http://www.cc.gatech.edu/~brendan/ccs09_siggen.pdf>

[13] Identifying Memory Images <http://gleeda.blogspot.com/2010/12/identifying-memory-images.html>

[14] YOUR CLOUD IS IN MY POCKET <https://media.blackhat.com/bh-dc-11/Suiche/BlackHat_DC_2011_Suiche_Cloud_Pocket-wp.pdf>

[15] Finding Object Roots in Vista (KPCR) <http://blog.schatzforensic.com.au/2010/07/finding-object-roots-in-vista-kpcr/>