

Memory Analysis Cheat Sheet for Microsoft Windows XP SP2

Methodology

1. Acquire memory dump
2. Identify OS version
3. Build lists of important structures
4. Look for "missing" objects
5. Look for "odd" data

Crash Dump

Use system control panel to set dump type to "full" (recommended) or "kernel". Also set location. Reboot.

For PS/2 keyboard:

```
HKLM\System\CurrentControlSet\Services\i8042prt
↳Parameters
Create DWORD value "CrashOnCtrlScroll"
Set value to 1
```

For USB keyboard:

```
HKLM\System\CurrentControlSet\Services\kbdhid\
↳Parameters
Create DWORD value "CrashOnCtrlScroll"
Set value to 1
```

Only works on Microsoft Windows Server 2003

Press and hold **right** [CTRL] key, press [SCROLL LOCK] twice.

Identify OS Version

```
ostest.pl file
kern.pl file
python volatools ident -f file
```

Microsoft Debugger

URL of symbol server:
<http://msdl.microsoft.com/download/symbols>

General commands

```
.hh      open help file
.cls     clear screen
.reload  reloads symbols
        /f      force immediate load
        /i      ignore version mismatch, implies /f
        /v      verbose
?        resolve symbol
poi(arg) dereference pointer arg
```

Change context

```
.process [/p [/r]] [eprocess]
        /p      translate PTE
        /r      also reload user-mode symbols
```

Display Memory

```
dx [/p] range
        /p      range is physical address
ranges:
offset
name[+/-]offset
start end pair of addresses
Lnum    repeat implicit size num times
```

```
db      display bytes and ASCII characters
dw      display WORD values
dd      display DWORD values
dq      display QWORD values
dp      display pointers (like dd/dq)
da      display ASCII characters
du      display UNICODE characters
```

```
d        repeat last display command
```

Display Structures

```
dt [options] [module:]name [field] [address]
options:
-a [num] display num array elements
-b        display blocks recursively
-r num   recursively display sub-structures down
         to num levels (1..9)
-p        address is a physical address
-v        report overall size and element count
```

Display Executable Objects

```
!process pid[eprocess] [flags]
flags:
0x01    display time and priority statistics
0x02    display threads and events associated
         with process
0x04    display threads. If bit 0x02 is not set,
         uses single line. If bit 0x02 is set, then
         also displays a stack trace.
0x08    display return address and ESP
0x10    set process context, improving stack
         dumps
```

```
!peb [peb]
```

```
!m options [m namepattern] [M pathpattern]
```

```
options:
f        display image full pathname
n        display image name
o        display only loaded modules
t        display timestamp
```

```
!thread [-p] [-t] ethread [flags]
```

```
flags:
0x02    display wait states
0x04    display stack trace, use with 0x02
0x08    display return address and ESP
0x10    set process context, improving stack
         dumps
```

```
!teb [teb]
```

```
!chkimg [options] [-mmw logfile logoptions] [module]
```

```
options:
-as      scan all sections
-ss      scan only section name
-r        scan range from start to end
-nospec  scan also special parts of HAL and kernel
-d        display summary
-db       display hex view of changed locations
-v        verbose output
-mmw     log to file
```

```
logoptions:
```

```
v        virtual address
r        offset within module
s        symbol
S        section
e        expected value
w        found value
```

```
!for_each_module "!chkimg @#ModuleName"
```

Display other Objects

Generic object information:

```
!object address
!object 0 class
!object \path
```

```
!handle -1|0|index [flags]
```

```
flags:
0x01    basic handle information
0x02    objects
0x04    free handle entries
0x10    use kernel handle table
0x20    interpret as thread ID or process ID
```

Memory Analysis Cheat Sheet for Microsoft Windows XP SP2

!job 0 [pid|eprocess] [flags] [pid|eprocess] [type]

flags:

0x01 show job settings

0x02 show all processes in job

type: Event, Section, File, Port, Directory, SymbolicLink, Mutant, WindowStation, Semaphore, Key, Token, Process, Thread, Desktop, IoCompletion, Timer, Job, and WaitablePort

Memory Usage

Display physical memory usage statistics

!memusage [flags]

flags:

0x00 display general information and data from PFN database

0x08 display only general information

Display virtual memory usage statistics

!vm [flags]

flags:

0x01 omit process-specific statistics

0x02 include memory management thread stacks

0x04 include terminal server memory usage

0x08 include the page file write log

0x10 include working set owner thread stacks

0x20 include kernel virtual address usage

Display control area of section

!ca address get address from !memusage output

Convert physical address into virtual address

!ptov pfn

Convert virtual address into physical address

!vtop pagedirpfn virtualaddress

List of system PTE

!sysptes [flags]

flags:

0x01 only free PTE

0x02 free PTE in the global special pool

0x04 detailed information about any system PTE that are allocated to mapping locked pages

0x08 non-paged pool expansion free PTE

Display PDE and PTE

!pte virtualaddress

!pte pteaddress

!pte literaladdress 1

Display information about a Page Frame

!pfn pfn

Pool Allocations

Find allocations by tag

!poolfind tagstring|0xtagvalue [pool]

pool:

0 non-paged pool

1 paged pool

Display information about a special allocation

!pool [address] [flags]

flags:

0x01 also display content

0x02 suppress header info for all but the selected pool allocation

Show graphical map of pool utilization

!xpoolmap [pool]

pool:

0 non-paged pool

1 paged pool

PTFinder

by Andreas Schuster

Find processes and threads

--help full help

--usage help about command line options

--threads include threads in output (default)

--nothreads suppress threads in output (reduces clutter)

--processes include processes in output (default)

--noprocesses suppress processes in output (hardly useful)

--3GB assume /3GB boot option

--dotfile file produce DOT output for GraphViz

--xmlfile file produce XML output for KnTList

PoolTools

by Andreas Schuster

Examine memory pool allocations

All PoolTools share a common set of options:

--help full help

--usage help about command line options

--dbfile SQLite database to hold case data

--dumpfile memory dump in raw ('dd') format

--level num suppress candidates below confidence level (0 ... 100)

PoolFinder

Build a list of pool allocations

--3GB assume /3GB boot option

--checkpage special handling for pagefile.sys

--dbcreate file create database file and table

--stricttags apply strict checks to pool tags

PoolGrep

Search pool allocations for strings

--free search also freed (returned) allocations

--ignorecase case-insensitive search

Hint: insert null bytes in order to search for UCS-2 characters:

poolgrep ... t\000e\000s\000t\000

PoolDump

Produce hex dump of pool allocations

--free search also freed (returned) allocations

--nofree ignore freed allocations

--protection select allocations by protection bit (on/off/any, default: any)

--tag select allocation by pool tag (regular expressions are allowed)

PoolView

Interpret data in selected pool allocations

--class class select class of objects to view

--list-classes list available classes

Memory Analysis Cheat Sheet for Microsoft Windows XP SP2

Volatility

by Aaron Walters and Nick Petroni
Contributions by Brendan Dolon-Gavitt.

```
python volatility ident -f file      list open files
python volatility datetime -f file  date and time of
                                     memory image
python volatility pslist -f file     list processes
python volatility psscan -f file     scan for processes
python volatility thrdscan -f file   scan for threads
python volatility dlllist -f file    list DLLs
python volatility modules -f file    list modules
python volatility sockets -f file    list open sockets
python volatility sockscan -f file   scan for open sockets
python volatility connections -f file list TCP established
                                     connections
python volatility connscan -f file   scan for TCP
                                     connections
python volatility vadinfo -f file    information about every
                                     single VAD
python volatility vaddump -f file     save memory region
                                     into separate files
python volatility vadwalk -f file    produce VAD tree
```

Windows IR/CF Tools

by Harlan Carvey

```
ostest.pl      guess OS version through fingerprinting
                of System PID and PageDir address
kern.pl        guess OS version from kernel's version
                resource
```

Enumerations

Dispatcher Object Types

Id	Type	Structure
0	Notification Event	_KEVENT
1	Synchronization Event	_KEVENT
2	Mutant (Mutex)	_KMUTANT
3	Process	_KPROCESS
4	Queue	_KQUEUE
5	Semaphore	_KSEMAPHORE
6	Thread	_KTHREAD
8	Notification Timer	_KTIMER
9	Synchronization Timer	_KTIMER

IO Object Types

Id	Type	Structure
1	Adapter	_ADAPTER_OBJECT
2	Controller	_CONTROLLER_OBJECT
3	Device	_DEVICE_OBJECT
4	Driver	_DRIVER_OBJECT
5	File	_FILE_OBJECT
6	IRP	_IRP
7	Master Adapter	
8	Open Packet	
9	Timer	_IO_TIMER
10	Volume Parameter Block	_VPB
11	Error Log	
12	Error Message	
13	Device Object Extension	_DEVOBJ_EXTENSION
18	APC	_KAPC
19	DPC	_KDPC
20	Device Queue	_KDEVICE_QUEUE
21	Event Pair	
22	Interrupt	_KINTERRUPT
23	Profile	

Pool Types

Shown are the values as found in memory, not as declared in the DDK/WDK header files.

Pool types > 32 indicate a session pool.

0 = free		
	non-paged pool	paged pool
normal	1	2
must succeed	3	-
cache aligned	5	6
cache aligned, must succeed	7	-

Important Structures

Object

```
struct _OBJECT_HEADER, 12 elements, 0x20 bytes
+0x000 PointerCount : Int4B
+0x004 HandleCount : Int4B
+0x004 NextToFree : Ptr32 to
+0x008 Type : Ptr32 to _OBJECT_TYPE
+0x00c NameInfoOffset : UChar
+0x00d HandleInfoOffset : UChar
+0x00e QuotaInfoOffset : UChar
+0x00f Flags : UChar
+0x010 ObjectCreateInfo : Ptr32
+0x010 QuotaBlockCharged : Ptr32
+0x014 SecurityDescriptor : Ptr32
+0x018 Body
```

Information taken from "Undocumented Windows 2000 Secrets" (by Sven B. Schreiber).

```
- 0x50 _OBJECT_QUOTA_CHARGES (optional)
- 0x40 _OBJECT_HANDLE_DB ( optional)
- 0x38 _OBJECT_NAME (optional)
- 0x28 _OBJECT_CREATOR_INFO (optional)
- 0x18 _OBJECT_HEADER (mandatory)
0x00 object
```

Note: not all structures have to be present, see Flags in _OBJECT_HEADER:

```
0x01 has _OBJECT_CREATE_INFO
0x02 created by kernel
0x04 has OBJECT_CREATOR_INFO
0x20 has SECURITY_DESCRIPTOR
0x40 no _OBJECT_HANDLE_DB
```

Synchronizable Object

```
struct _DISPATCHER_HEADER, 6 elements, 0x10 bytes
+0x000 Type : UChar
+0x001 Absolute : UChar
+0x002 Size : UChar
+0x003 Inserted : UChar
+0x004 SignalState : Int4B
+0x008 WaitListHead : struct _LIST_ENTRY
```

Process

Note: Only selected members are shown below.

```
struct _EPROCESS, 107 elements, 0x260 bytes
+0x000 Header : struct _DISPATCHER_HEADER
+0x018 DirectoryTableBase : (2 elements) Uint4B
```

Memory Analysis Cheat Sheet for Microsoft Windows XP SP2

```
+0x050 ThreadListHead : struct _LIST_ENTRY
+0x070 CreateTime      : _LARGE_INTEGER
+0x078 ExitTime       : _LARGE_INTEGER
+0x084 UniqueProcessId : Ptr32
+0x088 ActiveProcessLinks : struct _LIST_ENTRY
+0x0c4 ObjectTable    : Ptr32 to _HANDLE_TABLE
+0x138 SectionObject  : Ptr32
+0x13c SectionBaseAddress : Ptr32
+0x174 ImageFileName  : (16 elements) UChar
+0x1a0 ActiveThreads  : UInt4B
+0x1b0 Peb            : Ptr32 _PEB
+0x1b8 ReadOperationCount : _LARGE_INTEGER
+0x1c0 WriteOperationCount : _LARGE_INTEGER
+0x1c8 OtherOperationCount : _LARGE_INTEGER
+0x1d0 ReadTransferCount : _LARGE_INTEGER
+0x1d8 WriteTransferCount : _LARGE_INTEGER
+0x1e0 OtherTransferCount : _LARGE_INTEGER
+0x1f4 SeAuditProcessCreationInfo : Ptr32
+0x24c ExitStatus     : Int4B
+0x252 SubSystemMinorVersion : UChar
+0x253 SubSystemMajorVersion : UChar
```

Thread

Note: Only selected members are shown below.

```
struct _ETHREAD, 54 elements, 0x258 bytes
+0x000 Header      : struct _DISPATCHER_HEADER
+0x020 Teb         : Ptr32 to
+0x1b0 ThreadListEntry : struct _LIST_ENTRY
+0x1c0 CreateTime  : _LARGE_INTEGER
+0x1c8 ExitTime    : _LARGE_INTEGER
+0x1ec Cid         : struct _CLIENT_ID
+0x000 UniqueProcess : Ptr32 to
+0x004 UniqueThread : Ptr32 to
```

Job

Note: Only selected members are shown below.

```
struct _EJOB, 45 elements, 0x180 bytes
+0x010 JobLinks    : struct _LIST_ENTRY
+0x018 ProcessListHead : struct _LIST_ENTRY
+0x07c TotalProcesses : UInt4B
+0x080 ActiveProcesses : UInt4B
+0x084 TotalTerminatedProcesses : UInt4B
```

Combined Structures

Note: Additional structures can be inserted before the _OBJECT_HEADER. Shown below is the configuration which is most likely to be found.

Device

```
0x00 _POOL_HEADER
0x08 _OBJECT_DIRECTORY_ENTRY
0x0c _OBJECT_NAME_INFORMATION
0x18 _OBJECT_HEADER
0x30 _DEVICE_OBJECT
0xf4 reserved
varies _DEVOBJ_EXTENSION
```

Some device structures also contain a _OBJECT_QUOTA_CHARGES structure.

Driver

```
0x00 _POOL_HEADER
0x08 _OBJECT_DIRECTORY_ENTRY
0x0c _OBJECT_NAME_INFORMATION
0x18 _OBJECT_HEADER
0x30 _DRIVER_OBJECT
0xd8 _DRIVER_EXTENSION
```

File

```
0x00 _POOL_HEADER
0x08 _OBJECT_HANDLE_DB
0x10 _OBJECT_HEADER
0x28 _FILE_OBJECT
```

Process

```
0x00 _POOL_HEADER
0x08 _OBJECT_HEADER
0x20 _EPROCESS
```

Thread

```
0x00 _POOL_HEADER
0x08 _OBJECT_HEADER
0x20 _ETHREAD
```

Timer

```
0x00 _POOL_HEADER
0x08 _OBJECT_HEADER
0x20 _ETIMER
```

Symbols

System Configuration

```
nt!KeNumberProcessors
nt!KeActiveProcessors
nt!KiProcessorBlock
```

```
nt!KeBootTime          use !filetime to decode
nt!KeBootTimeBias
```

Memory Information

```
nt!MmPagesSize
nt!MmLowestPhysicalPage
nt!MmHighestPhysicalPage
nt!MmNumberOfPhysicalPages
nt!MmPfnDatabase
nt!MmNumberOfPagingFiles
```

```
nt!MmPagedPoolStart
nt!MmPagedPoolEnd
nt!MmNonPagedPoolStart
nt!MmNonPagedPoolEnd
```

```
nt!MmBadPageListHead
nt!MmFreePageListHead
nt!MmRomPageListHead
nt!MmStandbyPageListHead
nt!MmZeroedPageListHead
```

Important Object Types

```
nt!PsJobType          Ptr to job _OBJECT_TYPE
nt!PsProcessType     Ptr to process _OBJECT_TYPE
nt!PsThreadType      Ptr to thread _OBJECT_TYPE
```

```
nt!PsActiveProcessHead Ptr to _EPROCESS+0x088
nt!PsInitialSystemProcess Ptr to _EPROCESS of System
nt!PsIdleProcess       Ptr to _EPROCESS of Idle
nt!KiIdleProcess       _EPROCESS of Idle
```

```
nt!PspJobList         Ptr to _EJOB-0x010
nt!PspCidTable
nt!PsLoadedModuleList
```

Networking

```
ArpInterfaceList
AddrObjTable
AddrObjTableSize
```

System Functions

```
nt!KeServiceDescriptorTable
nt!KeServiceDescriptorTableShadow
```

Helpful Lists

```
nt!CmpHiveListHead
nt!HandleTableListHead
nt!ObpKernelHandleTable
nt!KiTimerTableListHead
nt!ObpRootDirectoryObject
```

Memory Analysis Cheat Sheet for Microsoft Windows XP SP2

SQLite

SQLite is the database backend used by PoolTools.

```
select ::= SELECT [ ALL | DISTINCT ] result
         [ FROM table-list ]
         [ WHERE expr ]
         [ GROUP BY expr ]
         [ compound-op select* ]
         [ ORDER BY sort-expr-list ]
         [ LIMIT integer [(OFFSET | ,) integer ] ]
         ;
result  ::= result-column [, result-column]*
result-column ::= * | table-name.* | expr [[AS] string]
table-list ::= table [join-op table join-args]
table      ::= table-name [AS alias] | (select) [AS alias]
join-op    ::= , | [NATURAL] [LEFT | RIGHT | FULL]
           [OUTER | INNER | CROSS] JOIN
join-args  ::= [ON expr] [USING (id-list)]
sort-expr-list ::= expr [sort-order] [, expr [sort-order]]*
sort-order  ::= [COLLATE collation-name] [ASC | DESC]
compound-op ::= UNION | ALL | INTERSECT | EXCEPT
```