Windows Operating System Agnostic Memory Analysis

By

James Okolica and Gilbert Peterson

Presented At

The Digital Forensic Research Conference

DFRWS 2010 USA  Portland, OR (Aug 2nd - 4th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http://dfrws.org
Windows Operating Systems Agnostic Memory Analysis

James Okolica
Gilbert Peterson
Overview

• A Process for Forensic Analysis of Memory

  - Retrieve Physical Memory
  - Extract Information
  - Generate human-understandable narrative

• Windows Operating System Agnostic Memory Analysis
  • Functionality
  • Experimental Set up and Results

• Next Steps
Forensic Analysis of Memory

Retrieve Physical Memory

Extract Information

Generate human-understandable narrative

Level of Detail

Level of Abstractions
Forensic Analysis of Memory

- Hardware based acquisition
- Virtual Machines
- Hibernation File
- Operating System Patches
- Software (Memoryze, MDD, Win32dd/ Win64dd)
Forensic Analysis of Memory

Develop America's Airmen Today ... for Tomorrow

- Memoryze (freeware)
- Volatility (open source in Python)
- CMAT (open source in C++)

Extract Information

Level of Detail

Level of Abstractions
CMAT Capabilities

- Programmatically determine O/S version
  - Physical Address Extensions enabled/disabled, 32-bit/64-bit

- Load O/S specific data structures
  - O/S-agnostic signatures for processes

- Locate data structures within PEs
  - O/S-agnostic retrieval of network activity

- Process Information
  - Network activity, objects accessed, modules loaded

- Configuration information
  - The users with accounts, application configurations

- Retrieve memory swapped to disk, memory-mapped files
CMAT – The Process of Extraction

1. Determine the O/S
2. Scan Memory for Processes & Hives
3. Retrieve Network Information
4. Load User List
5. Connect Users & Processes
CMAT – Determine the O/S

Develop America's Airmen Today ... for Tomorrow

Determine the O/S ➔ Scan Memory for Processes & Hives ➔ Retrieve Network Information ➔ Load User List ➔ Connect Users & Processes

- Scan memory for _DBGKD_DEBUG_DATA_HEADER64
  - OwnerTag = KDBG, Size less than 4K

_DEBUGKDEDEBUGGER_DATA64
0x00 List_Entry64 List
0x10 ULong OwnerTag
0x14 ULong Size

_KDEDEBUGGER_DATA64
0x00 _DBGKD_DEBUG_DATA_HEADER64 Header
0x18 ULong64 KernBase
0x20 ULong64 BreakPointwithStatus
0x28 ULong64 SavedContext
0x30 UShort ThCallBackStack
0x32 UShort NextCallBack
0x34 UShort FramePointer
0x36 UShort PAEEnabled:1
...
0x48 ULong64 PSLoadedModuleList
...
0xA0 ULong64 OBTypeObjectType

_DBGKD_GET_VERSION64
0x00 UShort MajorVersion
0x02 UShort MinorVersion
0x04 UChar ProtocolVersion
0x05 UChar KdSecondaryVersion
0x06 UShort Flags
0x08 UShort MachineType
0x0A UChar MaxPacketType
0x0B UChar MaxStateChange
0x0C UChar MaxManipulate
0x0D UChar Simulation
0x0E UShort[] Unused
0x10 UQuad KernBase
0x18 UQuad PsLoadedModuleList
0x20 UQuad DebuggerDataList

Debugging Tools for Windows\sdk\inc\wbdgexts.h
CMAT – Determine the O/S

- If _KDDEBUGGER_DATA64 isn’t present, go look for the kernel Portable Executable (due to Address Space Layout Randomization (ASLR))

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>Characteristics</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>TimeDataStamp</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>MajorVersion</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>MinorVersion</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>Debugger Information Type</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>Size of Data</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>AddressOfRawData</td>
</tr>
</tbody>
</table>

### .debug Section

- Base of Image Header
- MS-DOS 2.0 Compatible EXE Header
- OEM Identifier OEM Information
- Offset to PE Header
- MS-DOS 2.0 Stub Program and Relocation Table
- Unused
- PE Header (Aligned on 8-byte boundary)
- Section Headers
- Import Pages
  - Import Information
  - Export Information
  - Base relocations
  - Resource Information

### .debug data

- Portable Database Filename
  - “ntoskrnl.pdb”
  - “ntkrnlnpa.pdb”
  - “ntkrpamp.pdb”
  - “ntkrnlmp.pdb”

Debugging Tools for Windows\sdk\inc\wbdgexts.h
If `_KDDEBUGGER_DATA64` isn’t present, go look for the kernel Portable Executable (due to Address Space Layout Randomization (ASLR))

- **CMAT – Determine the O/S**
  - Scan Memory for Processes & Hives
  - Retrieve Network Information
  - Load User List
  - Connect Users & Processes

This program cannot be run in DOS mode

Base of Image Header

MS-DOS 2.0 Section (for MS-DOS compatibility, only)

Import Pages
- Import information
- Export information
- Base relocations
- Resource information

MS-DOS 2.0 Stub Program and Relocation Table

Offset to PE Header

OEM Information

OEM Identifier

Unused

MS-DOS 2.0 Compatible EXE Header

Debugging Tools for Windows\sdk\inc\wbdgexts.h
CMAT – Determine the O/S

- If _KDDEBUGGER_DATA64 isn’t present, go look for the kernel Portable Executable (due to Address Space Layout Randomization (ASLR))

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>MachineType</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>0x10b = 32 bit, 0x20b = 64 bit</td>
</tr>
<tr>
<td>64</td>
<td>2</td>
<td>MajorVersion</td>
</tr>
<tr>
<td>66</td>
<td>2</td>
<td>MinorVersion</td>
</tr>
</tbody>
</table>

Base of Image Header

MS-DOS 2.0 Section (for MS-DOS compatibility, only)

Debugging Tools for Windows\sdk\inc\wbdgexts.h
CMAT – Determine the O/S

- At this point, we know:
  - 32-bit or 64-bit
  - PAE enabled or disabled
  - Machine Type
  - Operating System version

- We now need to find the tables that convert linear addresses to physical addresses
CMAT – Determine the O/S

Virtual Address – 32-bit with PAE Disabled

<table>
<thead>
<tr>
<th>Page Directory Table Index</th>
<th>Page Table Index</th>
<th>Page Frame Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>22</td>
<td>12</td>
</tr>
</tbody>
</table>

Virtual Address – 32-bit with Physical Address Extensions

<table>
<thead>
<tr>
<th>Page Directory Pointer Table Index</th>
<th>Page Directory Table Index</th>
<th>Page Table Index</th>
<th>Page Frame Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>

Virtual Address – 64-bit

<table>
<thead>
<tr>
<th>Page Map Level Table Index</th>
<th>Page Directory Pointer Table Index</th>
<th>Page Directory Table Index</th>
<th>Page Table Index</th>
<th>Page Frame Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>39</td>
<td>30</td>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>
CMAT – Determine the O/S

- Find self-referencing pages – potential page directory table base addresses (assume the first one is good)

32-bit

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55d000</td>
<td>xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x55dc00</td>
<td>0x55d000 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx</td>
</tr>
</tbody>
</table>

32-bit with Physical Address Extensions

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55d000</td>
<td>xxxxxxxx 00000000 xxxxxxxx 00000000 xxxxxxxx</td>
</tr>
<tr>
<td>0x55d010</td>
<td>xxxxxxxx 00000000 0x55d000 00000000 xxxxxxxx</td>
</tr>
</tbody>
</table>

64-bit

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55d000</td>
<td>xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x55df60</td>
<td>xxxxxxxx xxxxxxxx 0x55d000 00000000 xxxxxxxx</td>
</tr>
</tbody>
</table>
CMAT – Determine the O/S

- At this point, we know:
  - 32-bit or 64-bit
  - PAE enabled or disabled
  - Machine Type
  - Operating System version
  - The Page Directory Table Base for the Kernel

- We now need to retrieve the data structures for this version of the operating system, specifically this version of the kernel executable.
In addition to the name of the pdb file, .debug data also contains the globally unique identifier and age, the two values used to uniquely identify the pdb file on Microsoft’s Symbol Server.

**.debug Section**

- Offset: 0, Size: 4, Field: Characteristics
- Offset: 4, Size: 4, Field: TimeDataStamp
- Offset: 8, Size: 2, Field: MajorVersion
- Offset: 10, Size: 2, Field: MinorVersion
- Offset: 12, Size: 4, Field: Debugger Information Type
- Offset: 16, Size: 4, Field: Size of Data
- Offset: 20, Size: 4, Field: AddressOfRawData

**.debug data**

- Offset: 0, Size: 4, Field: GUID
- Offset: 4, Size: 20, Field: Age
- Offset: 24, Size: 24, Field: PDB Filename

**NOTE:** This varies by Visual Studio Compiler; however it is stable through XP, Vista, and 7.
CMAT – Determine the O/S

- With the PDB Name, GUID, and Age, a URL is constructed to query the Microsoft Symbol Server and download the PDB file

- Once retrieved in compressed form, the PDB file is uncompressed

- PDB files are structured like file systems

```c
#define PDB_SIGNATURE_200 "Microsoft C/C++ program database 2.00\n\n\x1AJG\0"
#define PDB_SIGNATURE_TEXT 40

typedef struct _PDB_SIGNATURE {
    BYTE abSignature[PDB_SIGNATURE_TEXT+4];
} PDB_SIGNATURE;

typedef struct _PDB_STREAM {
    DWORD dStreamSize; // in bytes, -1 = free stream
    PWORD pwStreamPages; // array of page numbers
} PDB_STREAM;

typedef struct _PDB_HEADER {
    PDB_SIGNATURE Signature; // PDB_SIGNATURE_200
    DWORD dPageSize; // 0x0400, 0x0800, 0x1000
    WORD wStartPage; // 0x0009, 0x0005, 0x0002
    WORD wFilePages; // file size / dPageSize
    PDB_STREAM RootStream; // stream directory
    WORD awRootPages[]; // pages containing PDB_ROOT
} PDB_HEADER;

typedef struct _PDB_ROOT {
    WORD wCount; // # of Streams
    WORD wReserved; // 0
    PDB_STREAM aStreams[];
} PDB_ROOT;

CMAT – Determine the O/S

- Some of the streams include:
  - Section data (e.g., where the .data section begins)
  - Structures (e.g., the _EPROCESS data structure)
  - Symbols (location of non-exported symbols, like _AddrObjTable)
  - Symbol Locations (adjustment to location of symbols within the .data)

- The location of these streams in the file system varies and they are not labeled. Heuristics are required:
  - The section stream seems to always begin with .data or .text
  - The structures stream seems to always begin at 0x38
  - The symbols stream seems to always begin with a 2 byte record size followed by the literal 0x110E
  - The symbol locations stream seems to always follow the symbols stream
CMAT – Determine the O/S

Develop America's Airmen Today ... for Tomorrow

<table>
<thead>
<tr>
<th>Determine the O/S</th>
<th>Scan Memory for Processes &amp; Hives</th>
<th>Retrieve Network Information</th>
<th>Load User List</th>
<th>Connect Users &amp; Processes</th>
</tr>
</thead>
</table>

Symbol Stream

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>UShort</td>
<td>Record Size</td>
</tr>
<tr>
<td>0x02</td>
<td>UShort</td>
<td>Unknown1</td>
</tr>
<tr>
<td>0x04</td>
<td>ULong</td>
<td>Unknown2</td>
</tr>
<tr>
<td>0x08</td>
<td>ULong</td>
<td>Offset</td>
</tr>
<tr>
<td>0x0C</td>
<td>UShort</td>
<td>Type</td>
</tr>
<tr>
<td>0x0E</td>
<td>*Char</td>
<td>Symbol Name</td>
</tr>
</tbody>
</table>

Section Stream

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>*Char</td>
<td>Name (not null terminated)</td>
</tr>
<tr>
<td>0x08</td>
<td>ULong</td>
<td>Virtual Size</td>
</tr>
<tr>
<td>0x0C</td>
<td>ULong</td>
<td>Virtual Address</td>
</tr>
<tr>
<td>0x10</td>
<td>ULong</td>
<td>Raw Size</td>
</tr>
<tr>
<td>0x14</td>
<td>ULong</td>
<td>Raw Pointer</td>
</tr>
<tr>
<td>0x18</td>
<td>ULong</td>
<td>Relocation Pointer</td>
</tr>
<tr>
<td>0x1C</td>
<td>ULong</td>
<td>Line Pointer</td>
</tr>
<tr>
<td>0x20</td>
<td>UShort</td>
<td>Relocation Count</td>
</tr>
<tr>
<td>0x22</td>
<td>UShort</td>
<td>Line Count</td>
</tr>
<tr>
<td>0x24</td>
<td>ULong</td>
<td>Characteristics</td>
</tr>
</tbody>
</table>

Symbol Relocation Stream

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>ULong</td>
<td>Relocation Address</td>
</tr>
<tr>
<td>0x04</td>
<td>ULong</td>
<td>Data Address</td>
</tr>
</tbody>
</table>

Structure Stream

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>UShort</td>
<td>Record Size</td>
</tr>
<tr>
<td>0x02</td>
<td>STRUCTURE_RECORD</td>
<td></td>
</tr>
</tbody>
</table>
CMAT – Determine the O/S

Develop America's Airmen Today ... for Tomorrow

- Determine the O/S
- Scan Memory for Processes & Hives
- Retrieve Network Information
- Load User List
- Connect Users & Processes

LF_FIELDLIST
- List of fields

LF_STRUCTURE
- UShort  Element Count
- UShort  Properties
- ULong   Field Index
- ULong   Derived
- ULong   Vshape
- UShort  Size
- Char*   Name

LF_POINTER
- ULong   Underlying Type
- ULong   Pointer Array

LF_MEMBER
- UShort  Properties
- ULong   Underlying Type
- UShort  Offset
- Char*   Name

LF_UNION
- UShort  Element Count
- UShort  Properties
- ULong   Field Index
- Char*   Name

LF_ENUM
- UShort  Element Count
- UShort  Properties
- ULong   Underlying Type
- ULong   Field Index
- Char*   Name

LF_PROCEDURE
- ULong   Return Value Type
- UChar   Call Type
- UChar   Unknown
- UShort  Element Count
- ULong   Field Index

LF_ENUMERATE
- UShort  Properties
- UShort  Value
- Char*   Name

LF_ARGLIST
- ULong   Element Count
- ULong[]  Arguments

- LF_ARRAY
- ULong   Underlying Type
- ULong   Index Type
- UShort  Size
- UShort  Unknown

- LF_BITFIELD
- ULong   Underlying Type
- UChar   Size
- UChar   Offset
- UShort  Unknown
CMAT – Determine the O/S

• At this point, we know:
  • 32-bit or 64-bit
  • PAE enabled or disabled
  • Machine Type
  • Operating System version
  • The Page Directory Table Base for the Kernel
  • All of the data structures that the kernel uses

• Now it’s time to start parsing the memory dump!
• Scan memory for
  • Potential processes (_DISPATCHER_HEADER: Type = 03, Absolute = 0, Size = O/S dependent)

_BEPROCESS
  0 _KPROCESS Pcb
  108 _EX_PUSH_LOCK ProcessLock
  112 _LARGE_INTEGER CreateTime
  120 _LARGE_INTEGER ExitTime
  128 _EX_RUNDOWN_REF RundownProtect
  132 32PVoid UniqueProcessId
  136 _LIST_ENTRY ActiveProcessLinks
  144 ULong[] QuotaUsage
  156 ULong[] QuotaPeak
  168 ULong CommitCharge
  172 ULong PeakVirtualSize
  176 ULong VirtualSize
  180 _LIST_ENTRY SessionProcessLinks
  188 32PVoid DebugPort
  192 32PVoid ExceptionPort
  196 *_HANDLE_TABLE ObjectTable
  200 _EX_FAST_REF Token

\_KPROCESS
  0 _DISPATCHER_HEADER Header
  16 _LIST_ENTRY ProfileListHead
  24 ULong[] DirectoryTableBase
  32 _KGDTENTRY LdtDescriptor
  40 _KIDTENTRY Int21Descriptor
  48 UShort IopmOffset
  50 UChar Iopl
  51 UChar Unused
  52 ULong ActiveProcessors
  56 ULong KernelTime
  60 ULong UserTime
  64 _LIST_ENTRY ReadyListHead
  72 _LIST_ENTRY SwapListEntry
  76 32PVoid VdmTrapcHandler
  80 _LIST_ENTRY ThreadListHead
  88 ULong ProcessLock
  92 ULong Affinity
  96 UShort StackCount
  98 RChar BasePriority
  107 UChar ExecuteOptions

_BDISPATCHER_HEADER
  0 UChar Type
  1 UChar Absolute
  2 UChar Size
  3 UChar Inserted
  4 Long SignalState
  8 _LIST_ENTRY WaitListHead
CMAT – Extract Registry Information


- Scan memory for
  - Potential hives (_HIVE: signature = bee0bee0)

```
_HIVE
0   _HHIVE
528 32PVoid[]
540  LIST_ENTRY
548  LIST_ENTRY
556  *_FAST_MUTEX
560  *_FAST_MUTEX
564  LIST_ENTRY
572  LIST_ENTRY
580  *_FILE_OBJECT
584  _UNICODE_STRING
592  _UNICODE_STRING
600  Ushort
602  Ushort
604  Ulong
608  Ulong
612  Ulong
616  Long
620  *CM_KEY_SECURITY_CACHE_ENTRY
624  LIST_ENTRY[]
1136 *EVENT
1140 *CM_KEY_CONTROL_BLOCK
...  

_HIVE
0   _HHIVE
4   Hive
8   FileHandles
12  NotifyList
16  HiveList
20  LRUViewListHead
24  PinViewListHead
28  HiveLock
32  ViewLock
36  _HBASE_BLOCK
40  _RTL_BITMAP
48  DirtyVector
52  ULong
56  FileName
60  ULong
64  UChar
68  SecurityCount
69  UChar
76  SecurityCacheSize
77  SecurityHitHint
80  ULong
84  SecurityCache
88  _DUAL[]
```

- Scan memory for
  - Potential hives (_HIVE: signature = bee0bee0)
CMAT – Extract Network Information

- Scan list of loaded modules to find address of tcpip.sys Portable Executable
- Locate the data structures in the data section of tcpip.sys
  - Windows XP: TCP: _AddrObjTable, _AddrObjTableSize, UDP: _TCBTable, _MaxHashTableSize
- Traverse the data structures for open port/socket information and associate them with relevant processes

CMAT – Load User List

- Traverse the REGISTRY/MACHINE/SOFTWARE hive for the key list: Microsoft/Windows NT/Current Version/ProfileList
- Create a user list structure using this list and associate each token id with the home directory and name
CMAT – Connect Users/Processes

- Connect the token id in the process with the token id in the user list and then store the user’s name in the process information.
Summary

At this point, we have:

- **Meta-Information**
  - 32-bit or 64-bit, PAE enabled or disabled, Machine Type, O/S Version
  - All of the data structures that the kernel uses

- **All the processes that were running on the machine**
  - The IP addresses and ports they were using
  - The objects (files, registry keys, etc.) they accessed
  - Modules loaded

- **The current configuration information for the machine**
  - The users with accounts
  - The last run configurations for the different applications

- **The portable executables for all active programs**
  - The actual code that was being executed
  - The current values of all of the variables
Experiments

• Test Environment
  • Applications: Internet Explorer, Word, PowerPoint, Visual Studio, Calculator, Kernel Debugger, two command line shells (one hidden by FUTo)
  • Memory Dump generated by Matthew Suiche’s Win32DD and Win64DD
  • Microsoft Utilities used: netstat
  • SysInternals utilities used: psinfo, psl list, logonsessions, handles, listdlls

• Results
  • CMAT provides equivalent information to SysInternals/ Microsoft tools
  • CMAT provides equivalent information to Volatility for Windows XP
Bottom Line

- CMAT provides equivalent results to other memory analysis tools without regard to the specific Windows O/S

Concerns
- CMAT still relies on some intrinsic knowledge of the O/S (e.g., there is a structure named _EPROCESS that stores process information)
  - In Windows 7, Microsoft changed how the _OBJECT_TYPE was stored within _OBJECT_HEADER
  - In Windows Vista, Microsoft implemented Address Space Layout Randomization (ASLR)
- Extracting symbols from proprietary executables (e.g., tcpip.sys) remains problematic because the symbol names are not published requiring reverse engineering.
CMAT-V extends CMAT into Virtual Machines allowing near real-time analysis of memory for Cyber SA.
Next Steps

- Is there a way to take this still overwhelming amount of information and abstract it to a level that is more useful to investigators?
  - Develop use cases for specific types of incidents
  - Determine what information investigators are most likely to need
  - Develop scripts to provide use case information in summary form while still allowing investigators to search for other information if necessary.
Questions

Contact information:
James (Jimmy) Okolica  jokolica@afit.edu  937-255-3636 x7255
Gilbert Peterson  gpeterson@afit.edu  937-255-3636 x4281
Live Cyber Forensics

- Business Productivity
  - Lost Revenue
  - Concern of the system coming back up
- Acquisition of volatile-only information
  - Network Traffic
  - Active process and user information
- Encrypted Hard Drives
- Memory Resident Malware
- Too much data
CMAT – Extract Registry Information

Develop America's Airmen Today ... for Tomorrow

- The Registry is basically a collection of file systems
  - To find information in them
    - Start at the Directory Base and add the offset to find the Table
    - Start at the Table Base and add the offset to find the location in the Hive Table
    - Move down through the “sub-directories” until arriving at the Value List

- 2 system hives – Machine and User
- 5 machine hives – Security Account Manager, Security, Software, Hardware, System
- 6 user hives – 2 each for
  - currently logged on user
  - local service user
  - the networks service user
• Enumerate system information
  • Operating system version
  • Number of processors
  • Physical address extensions enabled
  • Physical address of KDBG and (one of) the Kernel Page Directories
• Virtual address of the Kernel Portable Executable
• List of users with accounts
• Virtual address of each of the registry hives
CMAT – Display Information

- Enumerate process handles (e.g., files opened, registry keys accessed)
CMAT – Display Information

User Interface – Enumerate/ Dump System and Process Information

- Enumerate modules loaded by process

```
3/6: cmd.exe selected:
1  Display Process Environment Information
2  Display all DLLs loaded by process
3  Display all Files accessed by process
4  Display all Registry Keys accessed by process
5  Display all Sockets opened by process
<enter>: quit
```

Base Dll Name: cmd.exe  Full Name: C:\WINDOWS\system32\cmd.exe
Base Dll Name: ntdll.dll  Full Name: C:\WINDOWS\system32\ntdll.dll
Base Dll Name: kernel32.dll  Full Name: C:\WINDOWS\system32\kernel32.dll
Base Dll Name: user32.dll  Full Name: C:\WINDOWS\system32\user32.dll
Base Dll Name: shlwapi.dll  Full Name: C:\WINDOWS\system32\shlwapi.dll
Base Dll Name: shimmg.dll  Full Name: C:\WINDOWS\system32\shimmg.dll
Base Dll Name: winreg.dll  Full Name: C:\WINDOWS\system32\winreg.dll
Base Dll Name: wininet.dll  Full Name: C:\WINDOWS\system32\wininet.dll
Base Dll Name: advapi32.dll  Full Name: C:\WINDOWS\system32\advapi32.dll
Base Dll Name: rpcrt4.dll  Full Name: C:\WINDOWS\system32\rpcrt4.dll
Base Dll Name: secur32.dll  Full Name: C:\WINDOWS\system32\secur32.dll
Base Dll Name: wmmk.dll  Full Name: C:\WINDOWS\system32\wmmk.dll
Base Dll Name: ole32.dll  Full Name: C:\WINDOWS\system32\ole32.dll
Base Dll Name: oleui32.dll  Full Name: C:\WINDOWS\system32\oleui32.dll
Base Dll Name: shlwapi.dll  Full Name: C:\WINDOWS\system32\shlwapi.dll
Base Dll Name: userenv.dll  Full Name: C:\WINDOWS\system32\userenv.dll
Base Dll Name: usetheme.dll  Full Name: C:\WINDOWS\system32\usetheme.dll
Base Dll Name: imq32.dll  Full Name: C:\WINDOWS\system32\imq32.dll
Base Dll Name: comctl32.dll  Full Name: C:\WINDOWS\system32\comctl32.dll
Base Dll Name: apphelp.dll  Full Name: C:\WINDOWS\system32\apphelp.dll
Base Dll Name: msmctle.dll  Full Name: C:\WINDOWS\system32\msmctle.dll
Base Dll Name: msctftime.dll  Full Name: C:\WINDOWS\system32\msctftime.dll
Base Dll Name: msilib.dll  Full Name: C:\WINDOWS\system32\msilib.dll
```
CMAT – Display Information

• Enumerate process environment (e.g., fully qualified executable, command line)

• Enumerate process network activity