Windows Malware Analysis

Accelerated

with Memory Dumps

Version 2.0

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

Any of these:

- Basic and intermediate level Windows memory dump analysis using WinDbg
- C/C++/C# debugging skills
- Malware analysis (not WinDbg)
Training Goals

- Learn fundamentals of malware analysis
- Learn techniques and commands in the context of x86 and x64 memory dumps
Training Principles

- Talk only about what I can show
- Lots of pictures
- Original content and examples
Agenda

User space process memory
- Review of fundamentals
- Exercises

Kernel and physical space memory
- Review of fundamentals
- Exercises
Malware and Victimware

Typical scenarios when we want to check for possible malware presence:

- System or application abnormal behavior
- Controlled crash dumps during or after tracing and monitoring
Pattern-Oriented Approach

- How malware can be written
- How can we see that in a dump file
- Using WinDbg as a support tool
Pattern-Oriented Diagnostic Analysis

**Diagnostic Pattern**: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

**Diagnostic Problem**: a set of indicators (symptoms, signs) describing a problem.

**Diagnostic Analysis Pattern**: a common recurrent analysis technique and method of diagnostic pattern identification in a specific context.

**Diagnostics Pattern Language**: common names of diagnostic and diagnostic analysis patterns. The same language for any operating system: Windows, Mac OS X, Linux, ...

---

Information Collection (Scripts) → Information Extraction (Checklists) → Problem Identification (Patterns) → Problem Resolution, Troubleshooting Suggestions, Debugging Strategy

Memory Dump Analysis Patterns

Malware Analysis Patterns

© 2017 Software Diagnostics Services
Practice Exercises
Links

- Memory Dumps
  NOT IN THE PUBLIC PREVIEW VERSION

- Exercise Transcripts
  NOT IN THE PUBLIC PREVIEW VERSION
Exercise 0

- **Goal:** Install Debugging Tools for Windows and learn how to set up symbols correctly

- **Patterns:** Incorrect Stack Trace

- \AWMA-Dumps\Exercise-0-Download-Setup-WinDbg.pdf
User Space Memory
Space Review (x86)

User Space

Kernel Space

M1

MIDLL

ntdll

00000000

7fffffff

80000000

ffffffffff

M1.dmp

0:000> lm

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>013e0000</td>
<td>013f5000</td>
<td>M1</td>
</tr>
<tr>
<td>10000000</td>
<td>10039000</td>
<td>WinCRT</td>
</tr>
<tr>
<td>70120000</td>
<td>70134000</td>
<td>MIDLL</td>
</tr>
<tr>
<td>72a40000</td>
<td>72a91000</td>
<td>winspool</td>
</tr>
<tr>
<td>738b0000</td>
<td>73930000</td>
<td>uxtheme</td>
</tr>
<tr>
<td>73ec0000</td>
<td>73ed3000</td>
<td>dwmapi</td>
</tr>
<tr>
<td>746b0000</td>
<td>746c0000</td>
<td>CRYPTBASE</td>
</tr>
<tr>
<td>746c0000</td>
<td>74720000</td>
<td>sspicli</td>
</tr>
<tr>
<td>74a00000</td>
<td>74a83000</td>
<td>clbcatq</td>
</tr>
<tr>
<td>74a90000</td>
<td>74b20000</td>
<td>gd132</td>
</tr>
<tr>
<td>74b20000</td>
<td>74baf000</td>
<td>oleaut32</td>
</tr>
<tr>
<td>74bb0000</td>
<td>74baa000</td>
<td>lpk</td>
</tr>
<tr>
<td>74bc0000</td>
<td>74c6c000</td>
<td>msvcrt</td>
</tr>
<tr>
<td>74ca0000</td>
<td>74d6c000</td>
<td>msctf</td>
</tr>
<tr>
<td>750d0000</td>
<td>75170000</td>
<td>advapi32</td>
</tr>
<tr>
<td>751e0000</td>
<td>752f0000</td>
<td>kernel32</td>
</tr>
<tr>
<td>752f0000</td>
<td>75309000</td>
<td>sechost</td>
</tr>
<tr>
<td>75390000</td>
<td>753d7000</td>
<td>KERNELBASE</td>
</tr>
<tr>
<td>753e0000</td>
<td>75437000</td>
<td>shlwapi</td>
</tr>
<tr>
<td>760d0000</td>
<td>7622c000</td>
<td>ole32</td>
</tr>
<tr>
<td>76230000</td>
<td>762cd000</td>
<td>usp10</td>
</tr>
<tr>
<td>764d0000</td>
<td>765c0000</td>
<td>rpcrt4</td>
</tr>
<tr>
<td>765c0000</td>
<td>766c0000</td>
<td>user32</td>
</tr>
<tr>
<td>766c0000</td>
<td>76720000</td>
<td>imm32</td>
</tr>
<tr>
<td>76fe0000</td>
<td>77160000</td>
<td>ntdll</td>
</tr>
</tbody>
</table>

© 2017 Software Diagnostics Services
Space Review (x64)

User Space
- ntdll
- M1

Kernel Space
- MIDLL
- M1 DLL
- M1.dmp

0:00> lm

start                  end                 module name
00000000`76be0000 00000000`76cff000   kernel32
00000000`76d00000 00000000`76dfa000   user32
00000000`76e00000 00000000`76fa9000   ntdll
00000000`76f80000 00000000`76fb8000   M1
000007fe`f9d20000 000007fe`f9d70000   M1DLL
000007fe`fb700000 000007fe`fb718000   dwmapi
000007fe`fb990000 000007fe`fb9e6000   uxtheme
000007fe`fd190000 000007fe`fd1f0000   CRYPTBASE
000007fe`fd4e0000 000007fe`fd54c000   KERNELBASE
000007fe`fd6b0000 000007fe`fd749000   clbcatq
000007fe`fd940000 000007fe`fd9e6000   advapi32
000007fe`fd940000 000007fe`fd9e6000   rpcrt4
000007fe`fd940000 000007fe`fd9e6000   gdi32
000007fe`fd940000 000007fe`fd9e6000   ole32
000007fe`febe0000 000007fe`febe8000   imm32
000007fe`fed10000 000007fe`fed1e000   lpk
000007fe`fed20000 000007fe`fed2e000   msctf
000007fe`feca0000 000007fe`feca8000   sechost
000007fe`feec0000 000007fe`feef0000   usp10
000007fe`fef90000 000007fe`ff0f0000   msvcr7
000007fe`ff030000 000007fe`ff187000   oleaut32

© 2017 Software Diagnostics Services
Exercise M1A

- **Goal:** Look at module headers and version information before load

- **Patterns:** Unknown Module

- \AWMA-Dumps\Exercise-M1A.pdf
Dynamic Linking Design
After Dynamic Linking

PE (EXE)
Import Address Table (IAT) Directory
...
00000001'3f8a9280:00000000'76d19e74
...

call qword ptr [00000001'3f8a9280]
...
.text (Code)

PE (DLL)
00000000'76d19e74:
...
.text (Code)
Exercise M1B

- **Goal:** Look at address map, module headers and version information after load, check IAT, check import library calls, check module integrity

- **Patterns:** Unknown Module

- \AWMA-Dumps\Exercise-M1B.pdf
Packed Code and Data

- Less/No strings
- Less/No code signatures
- Less/No import functions
- Possibly different sections

Example: UPX
void main()
{
    foo();
    crash();
}

void foo()
{
    char sz[256] = "Some String";
    bar();
}

void bar()
{
    do();
}

void crash()
{
    WER();
}
Exercise M2

- **Goal:** Diagnose packed and hidden modules and their execution residues

- **Patterns:** Packed Code, Hidden Module, Pre-Obfuscation Residue, Execution Residue, String Hint

- \AWMA-Dumps\Exercise-M2.pdf
Malware Requirements

Module A ➔ Module B

? ➔
Malware Architecture

- **Before load**

- **After load: Hooksware**
Hooksware (Patching)

Module A → Malware → Module B

0:004> u wininet!InternetReadFile
wininet!InternetReadFile:
7758654b e90044ac88 jmp 0004a9d0
77586550 83ec24 sub esp,24h
77586553 53 push ebx

0:004> u 0004a9d0
0004a9d0 55 push ebp
0004a9d1 8bec mov ebp,esp
0004a9d3 6a00 push 0

0:004> u 008f0000
008f0000 8bff mov edi,edi
008f0002 55 push ebp
008f0003 8bec mov ebp,esp
008f0005 e94665c976 jmp wininet!InternetReadFile+0x5 (77586550)

© 2017 Software Diagnostics Services
Exercise M3

- **Goal:** Diagnose malware in victimware process memory dumps

- **Patterns:** Stack Trace Collection, RIP Stack Trace, Hookware, Patched Code, Hidden Module, Deviant Module, String Hint, Fake Module, No Component Symbols, Namespace

- \AWMA-Dumps\Exercise-M3.pdf
DLL Injection

Debugging TV Frame 0x20

Homework: InjectionResidue.DMP
Pathways

- Import Address Table
- System call dispatch
- Exception handling
Pattern Links

Stack Trace Collection
RIP Stack Trace
Hooksware
Hidden Module
String Hint
Fake Module
Patched Code

Packed Code
No Component Symbols
Pre-Obfuscation Residue
Deviant Module
Unknown Module
Execution Residue
Namespace
Kernel Space Memory
Space Review (x86)

User Space

Kernel Space

nt
hal
driver

MEMORY.DMP

© 2017 Software Diagnostics Services
Space Review (x64)

User Space

Kernel Space

ntdll
MI
User Space

MIDL
hal
nt
driver

Kernel Space

00000000`0000000

ffffff`fffffff

0000007ff`fffffff

ffffff`fffffff

M1

MEMORY.DMP

© 2017 Software Diagnostics Services
Driver PE Format

- Non-Paged code
- Page code
- Non-Paged data
- Paged data
- Discardable code and data
Suspicious Behaviour

- BSOD
- CPU consumption
- Network communication
- Slow system
CRITICAL_STRUCTURE_CORRUPTION (109)
This bugcheck is generated when the kernel detects that critical kernel code or data have been corrupted. There are generally three causes for a corruption:
1) A driver has inadvertently or deliberately modified critical kernel code or data. See http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx
2) A developer attempted to set a normal kernel breakpoint using a kernel debugger that was not attached when the system was booted. Normal breakpoints, "bp", can only be set if the debugger is attached at boot time. Hardware breakpoints, "ba", can be set at any time.
3) A hardware corruption occurred, e.g. failing RAM holding kernel code or data.
Arguments:
Arg1: a4a039d897c2787e, Reserved
Arg2: b4b7465eea408b28, Reserved
Arg3: fffff88000f2ef1c, Failure type dependent information
Arg4: 0000000000000002, Type of corrupted region, can be
   0 : A generic data region
   1 : Modification of a function or .pdata
   2 : A processor IDT
   3 : A processor GDT
   4 : Type 1 process list corruption
   5 : Type 2 process list corruption
   6 : Debug routine modification
   7 : Critical MSR modification
The First Steps

- Check the current thread: `!thread -1 3f`
- Check the current process: `!process -1 3f`
- Check the current CPU IDT
- Check the current thread raw stack
- Check running and ready threads
- List all processes and threads
- List all CPUs IDT
IDT

- Interrupt processing
- One for each CPU

!idt

!idt -a
Raw Stack

- System threads
- Kernel stacks for process threads
- **Scripting all threads**
Processes and Threads

- !process 0 0
- !process 0 3f
- !sprocess <session> 3f
- !for_each_thread "command"
- !vm
Attached Threads

THREAD fffffa80033b5b50  Cid 0004.0030  Teb: 0000000000000000  Win32Thread: 0000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
ffffff880021d9750  SynchronizationEvent
Not impersonating
DeviceMap  fffff8a00000088f0
Owning Process  fffffa80033879e0  Image: System
Attached Process  fffffa800439c620  Image: AppA.exe
Wait Start TickCount  30819  Ticks: 14746574 (2:15:54:08.028)
Context Switch Count  2800
UserTime  00:00:00.000
KernelTime  00:00:00.374
Win32 Start Address nt!ExpWorkerThread (0xfffff8000189e530)
Stack Init fffff880021d9db0  Current fffff880021d9470
Base fffff880021da000  Limit fffff880021d4000  Call 0
Priority 12  BasePriority 12  UnusualBoost 0  ForegroundBoost 0  IoPriority 2  PagePriority 5
CPU Spikes

- !running [-i] [-t]*
- !ready [f]*
- Ticks: 0

- Scripting

* doesn't show correct user space stack trace
Exercise M4

- **Goal:** Navigate through kernel space memory regions, list and analyze CPUs, processes and threads

- **Patterns:** Stack Trace Collection, Execution Residue, Self-Diagnosis

- \AWMA-Dumps\Exercise-M4.pdf
SSDT

System Service Dispatch Table

```
1: kd> uf ntdll!NtReadFile
ntdll!NtReadFile:
77870074 b802010000    mov    eax,102h
77870079 ba0003fe7f    mov    edx,offset SharedUserData!SystemCallStub (7ffe0300)
7787007e ff12          call   dword ptr [edx]
77870080 c22400        ret     24h

1: kd> dps nt!KiServiceTable+102*4 L1
8199302b 81880a2c nt!NtReadFile
```

User Space/Mode

Kernel Space/Mode

© 2017 Software Diagnostics Services
IRP Dispatch

IRP * = IoAllocateIrp(...)
IoCallDriver(DEVICE_OBJECT *, IRP *)

ntkrnlmp.exe

IRP

Driver.sys

DEVICE_OBJECT

ntdll.dll

DEVICE_OBJECT

kernel32.dll

DEVICE_OBJECT

Application.exe

DEVICE_OBJECT

ReadFile

NtReadFile

NtReadFile

Dispatch

Kernel Mode/Space

User Mode/Space

Malware

© 2017 Software Diagnostics Services
Device Driver Example

1: kd> !drvobj \Driver\CmBatt 3
Driver object (87668378) is for:
   \Driver\CmBatt
Driver Extension List: (id , addr)

Device Object list:
849e38a0  848c29b8

DriverEntry:  85a399bc  CmBatt!GsDriverEntry
DriverStartIo:  00000000
DriverUnload:  85a38b06  CmBatt!CmBattUnload
AddDevice:  85a38588  CmBatt!CmBattAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE  85a38b40  CmBatt!CmBattOpenClose
[01] IRP_MJ_CREATE_NAMEDPIPE  8181d171  nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE  85a38b40  CmBatt!CmBattOpenClose
[03] IRP_MJ_READ  87fe6226  ModuleA+0x3464  
[04] IRP_MJ_WRITE  8181d171  nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION  8181d171  nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION  8181d171  nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA  8181d171  nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA  8181d171  nt!IopInvalidDeviceRequest
[...]

© 2017 Software Diagnostics Services
False Positives

- Raw Pointer
- RIP Stack Trace
- .reload
Exercise M5

- **Goal:** Navigate CPUs, check IDT and SSDT, navigate through drivers and check their dispatch tables

- **Patterns:** Driver Device Collection, Raw Pointer, Out-of-Module Pointer

- \AWMA-Dumps\Exercise-M5.pdf
Direct Dump Manipulation

- Malware effects modeling
- Process and complete dumps
- `ep <address> value`
- `.dump /f <file name>`
Physical Space Memory
Space Review

Complete stack traces (x64 + x86)
Exercise M6

- **Goal:** Navigate processes in a complete memory dump, check x64 SSDT entries, check process and thread tokens, discover hidden processes and drivers, check IRP stacks

- **Patterns:** Deviant Token, Hidden Process, Hidden Module, Stack Trace Collection (I/O)

- \AWMA-Dumps\Exercise-M6.pdf
Memory Acquisition

Pattern Links

Self-Diagnosis
Driver Device Collection
Raw Pointer
Out-of-Module Pointer
Deviant Token
Hidden Process
Stack Trace Collection (I/O)
Resources

- WinDbg Help / [WinDbg.org](http://WinDbg.org) (quick links) / [DumpAnalysis.org](http://DumpAnalysis.org)
- The Rootkit Arsenal (2nd edition)
- Windows Internals, 6th ed.
- [Practical Foundations of Windows Debugging, Disassembling, Reversing](http://PracticalFoundations.com)
- [Memory Dump Analysis Anthology](http://MemoryDumpAnalysis.com) (Volumes 1 – 10)
Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!