Windows Malware Analysis

Accelerated

with Memory Dumps

Version 3.0

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
- Use memory dumps from the variety of systems up to Windows 11
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, ...
Practice Exercises
Links

- Memory Dumps
  Included in Exercise 0

- Exercise Transcripts
  Included in this book
Exercise 0

- **Goal:** Install WinDbg Preview or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly

- **Patterns:** Stack Trace; Incorrect Stack Trace

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

User Space

Kernel Space

M1

M1.dmp

MIDL

ntdll

00000000

7fffffff

80000000

ffffffff

0:000> lm
start    end        module name
004e0000 004fa000   M1
5bd60000 5be2b000  CoreMessaging
68e80000 69113000  CoreUIComponents
6c150000 6c167000  MIDL
6c2e0000 6c376000  TextShaping
704a0000 70581000  textinputframework
70760000 707e2000  uxtheme
71440000 714e0000  apphelp
73a30000 73a3b000  CRYPTBASE
73a90000 73aa2000  kernel_appcore
73b30000 73b85000  Oleacc
74c90000 74d7a000  wintypes
75ef0000 75f0f000  RPCRT4
75fb0000 760b2000  msvcp_win
761d0000 7626c000  OLEAUT32
76450000 764b4000  bcryptPrimitives
765d0000 766a4000  sechost
766b0000 7672c000  advapi32
76730000 767a4000  win32u
767c0000 767e5000  IMM32
76920000 76a32000  ucrtnbase
76a40000 76cc000  combase
76d30000 76e28000  KERNEL32
76e20000 76e2e000  msvcrt
76ef0000 77147000  KERNELBASE
77150000 77172000  GDI32
771d0000 772af000  gdi32full
772b0000 7745c000  USER32
77900000 779da000  MSCTF
77b40000 77ce000  ntdll
Exercise M1A

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

- **Patterns:** Unknown Module

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

PE (EXE)

Import Address Table (IAT) Directory

... 00007ff7`6af0f2c0:00007ff8`9a1e4620 ...

... call qword ptr [00007ff7`6af0f2c0] ...

.text (Code)

PE (DLL)

... 00007ff9`9a1e4620: ...

.text (Code)
Exercise M1B

Goal: Look at address map, module headers and version information after load, check IAT, check import library calls, and 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();
}

Thread Raw Stack Data

0:000> kc
module!crash+30
module!main+10

© 2022 Software Diagnostics Services
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
Malware Architecture

- Before load

- After load: Hooksware
Hooksware (Patching)

Module A

Module B

Malware

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)
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
Call Hint
Region Hint
Parameter Hint

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

0: kd> lmk

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>80200000</td>
<td>8020a000</td>
<td>BATTC</td>
</tr>
<tr>
<td>8020a000</td>
<td>8020c900</td>
<td>compbatt</td>
</tr>
<tr>
<td>8020d000</td>
<td>80215000</td>
<td>msisadrv</td>
</tr>
<tr>
<td>80215000</td>
<td>8021e000</td>
<td>WMILIB</td>
</tr>
<tr>
<td>8021e000</td>
<td>8022b000</td>
<td>WDFLDR</td>
</tr>
<tr>
<td>8022b000</td>
<td>80266000</td>
<td>CLFS</td>
</tr>
<tr>
<td>80266000</td>
<td>8026e000</td>
<td>BOOTVID</td>
</tr>
</tbody>
</table>

[…]

81800000 81ba1000 nt
81ba1000 81bd5000 hal

[…]

87eb3000 87ed6000 ndiswan
87ed6000 87ee1000 ndistapi
87ee1000 87ef8000 rasl2tp
87ef8000 87f03000 TDI

[…]

937b4000 93800000 srv
9446d000 94480000 dump_LSI_SCSI
96ca1000 96cc9000 fastFat
### Memory Map

<table>
<thead>
<tr>
<th>Start Address</th>
<th>End Address</th>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>fffffff00000000</td>
<td>fffffff7ffffffff</td>
<td>fffffff00000000</td>
</tr>
<tr>
<td>fffbc928dce40000</td>
<td>fffbc928dce40000</td>
<td>win32k</td>
</tr>
<tr>
<td>fffbc928e11000000</td>
<td>fffbc928e45100000</td>
<td>win32kbase</td>
</tr>
<tr>
<td>fffbc928e8a820000</td>
<td>fffbc928ea90000000</td>
<td>win32kfull</td>
</tr>
<tr>
<td>fffbc928ead300000</td>
<td>fffbc928ead300000</td>
<td>cdd</td>
</tr>
<tr>
<td>fff807608f000000</td>
<td>fff807608f000000</td>
<td>mcupdate_GenuineIntel</td>
</tr>
<tr>
<td>fff807609000000</td>
<td>fff807609000000</td>
<td>hal</td>
</tr>
<tr>
<td>fff8076091c0000</td>
<td>fff8076091c0000</td>
<td>kd</td>
</tr>
<tr>
<td>fff80760cb000000</td>
<td>fff80760cb000000</td>
<td>dump_diskdump</td>
</tr>
<tr>
<td>fff80760cf000000</td>
<td>fff80760cf000000</td>
<td>intelppm</td>
</tr>
<tr>
<td>fff80760c73000000</td>
<td>fff80760c73000000</td>
<td>NdisVirtualBus</td>
</tr>
<tr>
<td>fff80760ca600000</td>
<td>fff80760ca600000</td>
<td>swenum</td>
</tr>
<tr>
<td>fff80760cbb000000</td>
<td>fff80760cbb000000</td>
<td>rdpbus</td>
</tr>
<tr>
<td>fff80760cb000000</td>
<td>fff80760cb000000</td>
<td>ushubub</td>
</tr>
<tr>
<td>fff80760cb000000</td>
<td>fff80760cb000000</td>
<td>USBD</td>
</tr>
</tbody>
</table>

---

**User Space**
- ntdll
- M1

**Kernel Space**
- hal
- nt
- driver

**MEMORY.DMP**
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
BSOD

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
- !for_each_thread "command"
- !vm
Attached Threads

THREAD ffffa80033b5b50  Cid 0004.0030  Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
    ffffa80021d9750  SynchronizationEvent
Not impersonating
DeviceMap         ffffa800000088f0
Owning Process   ffffa80033879e0  Image: System
Attached Process ffffa800439c620  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 (0xfffff8000000000000000000000000000 WAIT:
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

**User Space/Mode**

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00007ffe`5b023800</td>
<td>mov</td>
<td>r10,rcx</td>
</tr>
<tr>
<td>00007ffe`5b023803</td>
<td>mov</td>
<td>eax,6</td>
</tr>
<tr>
<td>00007ffe`5b023808</td>
<td>test</td>
<td>byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1</td>
</tr>
<tr>
<td>00007ffe`5b023810</td>
<td>jne</td>
<td>ntdll!NtReadFile+0x15 (00007ffe`5b023815)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Branch</td>
</tr>
<tr>
<td>ntdll!NtReadFile+0x12:</td>
<td>syscall</td>
<td></td>
</tr>
<tr>
<td>00007ffe`5b023812</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00007ffe`5b023814</td>
<td>ret</td>
<td></td>
</tr>
<tr>
<td>ntdll!NtReadFile+0x15:</td>
<td>int</td>
<td>2Eh</td>
</tr>
<tr>
<td>00007ffe`5b023815</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00007ffe`5b023817</td>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

**Kernel Space/Mode**

<table>
<thead>
<tr>
<th>Address</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fffff807`62780750</td>
<td>mov</td>
<td>r11,rsp</td>
</tr>
</tbody>
</table>

© 2022 Software Diagnostics Services
IRP Dispatch

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

Malware

ntkrnlmp.exe
IRP
Driver.sys

ntdll.dll
kernel32.dll
Application.exe

Kernel Mode/Space
User Mode/Space

Malware

© 2022 Software Diagnostics Services
Device Driver Example

1: kd> !drvobj \Driver\CmBatt 3
Driver object (ffffbe0c87852e10) is for:
   \Driver\CmBatt

Driver Extension List: (id, addr)

Device Object list:
  fffffbe0c8784c790

DriverEntry:     fffff8076925d010   CmBatt!GsDriverEntry
DriverStartIo:   00000000
DriverUnload:    fffff80769257d80   CmBatt!CmBattUnload
AddDevice:       fffff8076925a590   CmBatt!CmBattAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE    fffff80769257680   CmBatt!CmBattOpenClose
[01] IRP_MJ_CREATE_NAMED_PIPE    fffff80762233c40   nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE     fffff80769257680   CmBatt!CmBattOpenClose
[03] IRP_MJ_READ      fffff80762233c40   nt!IopInvalidDeviceRequest
[03] IRP_MJ_READ      fffff80843322a80   ModuleA+0x3464
[04] IRP_MJ_WRITE     fffff80762233c40   nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION fffff80762233c40   nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION fffff80762233c40   nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA  fffff80762233c40   nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA    fffff80762233c40   nt!IopInvalidDeviceRequest
[...]
IRP Communication

I/O Manager (ntkrnlmp.exe)

IoCallDriver(…, IRP *)

IRP

I/O Stack Location 1
I/O Stack Location 2
I/O Stack Location 3

Dispatch

DriverA.sys

IoCallDriver(…, IRP *)

IRP

I/O Stack Location 1
I/O Stack Location 2
I/O Stack Location 3

Dispatch

DriverB.sys

IoCallDriver(…, IRP *)

IRP

I/O Stack Location 1
I/O Stack Location 2
I/O Stack Location 3

Dispatch

DriverC.sys

IoCallDriver(…, IRP *)
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, and 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 (quick links)
- DumpAnalysis.org / SoftwareDiagnostics.Institute / PatternDiagnostics.com
- Debugging.TV / YouTube.com/DebuggingTV / YouTube.com/PatternDiagnostics
- The Rootkit Arsenal (2nd edition)
- Practical Foundations of Windows Debugging, Disassembling, Reversing, 2nd Edition
- Encyclopedia of Crash Dump Analysis Patterns, 3rd edition
- Memory Dump Analysis Anthology (Diagnomicon)
Q&A

Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!