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

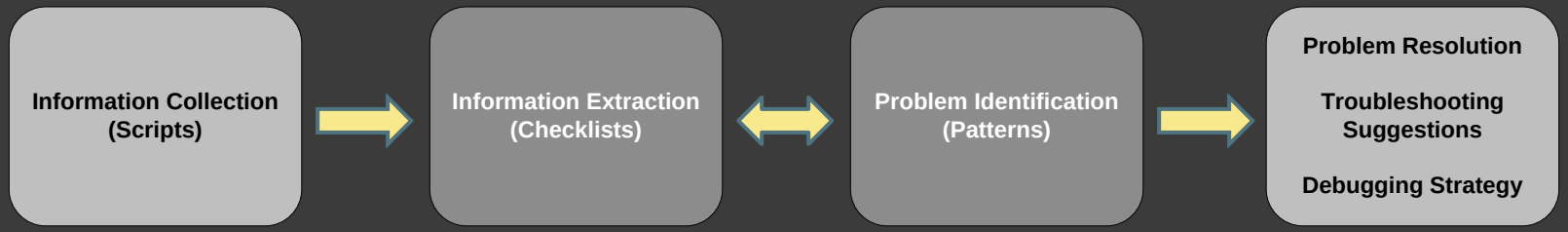
Pattern-Oriented Diagnostic Analysis

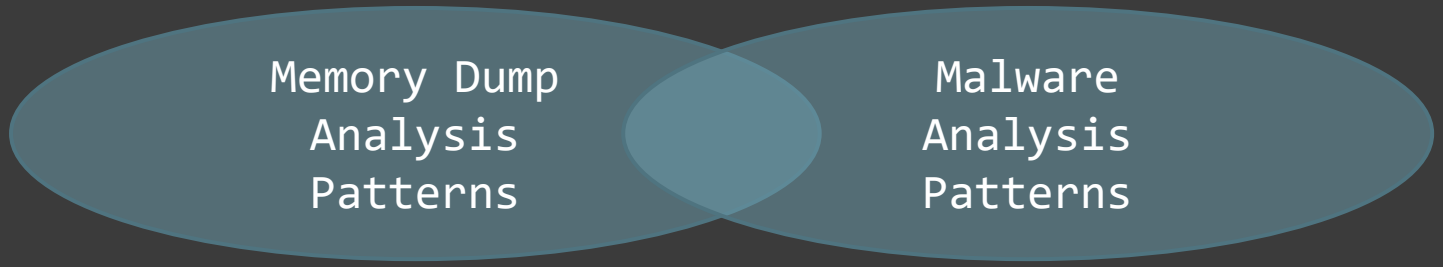
Diagnostic Pattern: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

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

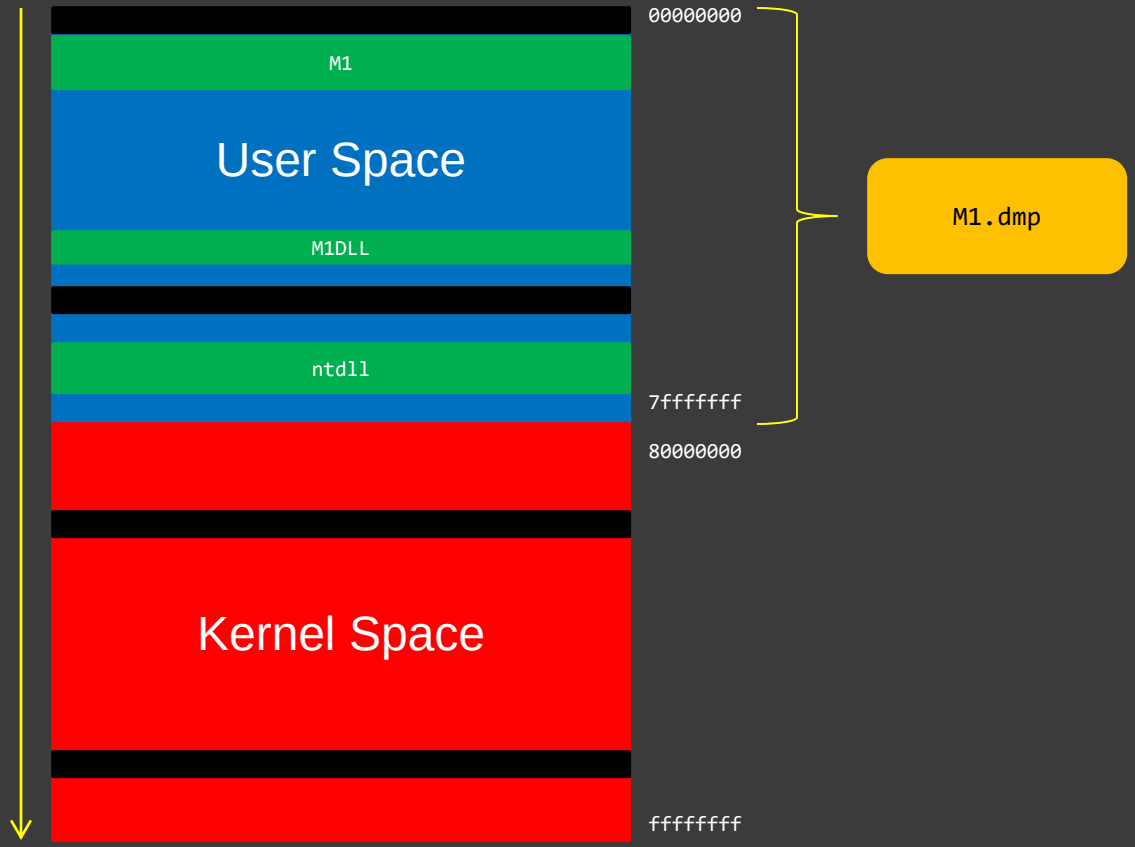
Included in this book

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

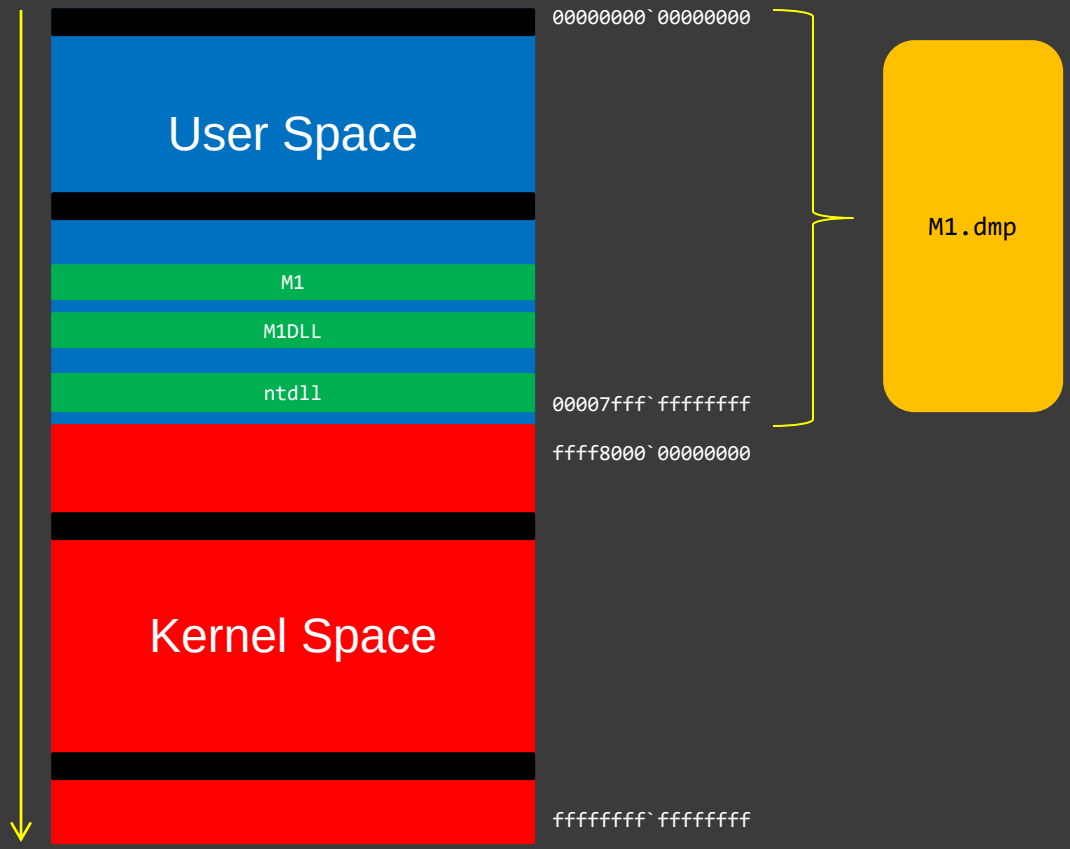
User Space Memory

Space Review (x86)



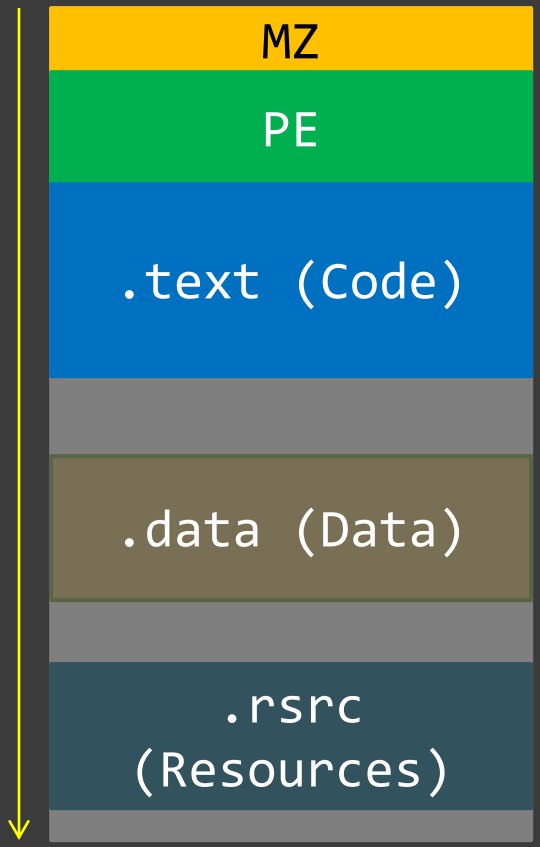

```
0:000> lm
start    end        module name
004e0000 004fa000   M1
5bd60000 5be2b000   CoreMessaging
68e80000 69113000   CoreUIComponents
6c150000 6c167000   M1DLL
6c2e0000 6c376000   TextShaping
704a0000 70581000   textinputframework
70760000 707e2000   uxtheme
71440000 714e0000   apphelp
73a30000 73a3b000   CRYPTBASE
73a90000 73aa2000   kernel_appcore
73b30000 73b85000   Oleacc
74c90000 74d7a000   wintypes
75ef0000 75fab000   RPCRT4
75fb0000 7602b000   msvcp_win
761d0000 7626c000   OLEAUT32
76450000 764b4000   bcryptPrimitives
765d0000 7664a000   sechost
766b0000 7672c000   advapi32
76730000 7674a000   win32u
767c0000 767e5000   IMM32
76920000 76a32000   ucrtbase
76a40000 76ccc000   combase
76d30000 76e20000   KERNEL32
76e20000 76ee2000   msvcrt
76ef0000 77147000   KERNELBASE
77150000 77172000   GDI32
771d0000 772af000   gdi32full
772b0000 7745c000   USER32
77900000 779da000   MSCTF
77b40000 77cea000   ntdll
```

Space Review (x64)




```
0:000> lm
start             end                 module name
00007ff7`9c540000 00007ff7`9c560000   M1
00007ffb`76830000 00007ffb`76899000   Oleacc
00007ffb`7c810000 00007ffb`7c93d000   textinputframework
00007ffb`7d9d0000 00007ffb`7da7e000   TextShaping
00007ffb`7fbd0000 00007ffb`7fbef000   M1DLL
00007ffb`8e9c0000 00007ffb`8ed2d000   CoreUIComponents
00007ffb`91880000 00007ffb`919b2000   CoreMessaging
00007ffb`95100000 00007ffb`951ac000   uxtheme
00007ffb`95cc0000 00007ffb`95e26000   wintypes
00007ffb`96e30000 00007ffb`96e48000   kernel_appcore
00007ffb`973f0000 00007ffb`973fc000   CRYPTBASE
00007ffb`98c20000 00007ffb`98d32000   gdi32full
00007ffb`98e00000 00007ffb`98e9d000   msvcp_win
00007ffb`98ea0000 00007ffb`99219000   KERNELBASE
00007ffb`99390000 00007ffb`994a1000   ucrtbase
00007ffb`994b0000 00007ffb`994d6000   win32u
00007ffb`994e0000 00007ffb`9955f000   bcryptPrimitives
00007ffb`99790000 00007ffb`9983e000   advapi32
00007ffb`99840000 00007ffb`99916000   OLEAUT32
00007ffb`99920000 00007ffb`99c99000   combase
00007ffb`99ca0000 00007ffb`99d5d000   KERNEL32
00007ffb`99d60000 00007ffb`99e80000   RPCRT4
00007ffb`9a1c0000 00007ffb`9a36c000   USER32
00007ffb`9ab30000 00007ffb`9abce000   sechost
00007ffb`9abd0000 00007ffb`9ac73000   msvcrt
00007ffb`9b290000 00007ffb`9b2b9000   GDI32
00007ffb`9b2c0000 00007ffb`9b3de000   MSCTF
00007ffb`9b420000 00007ffb`9b451000   IMM32
00007ffb`9b740000 00007ffb`9b949000   ntdll
```

EXE/DLL/SYS

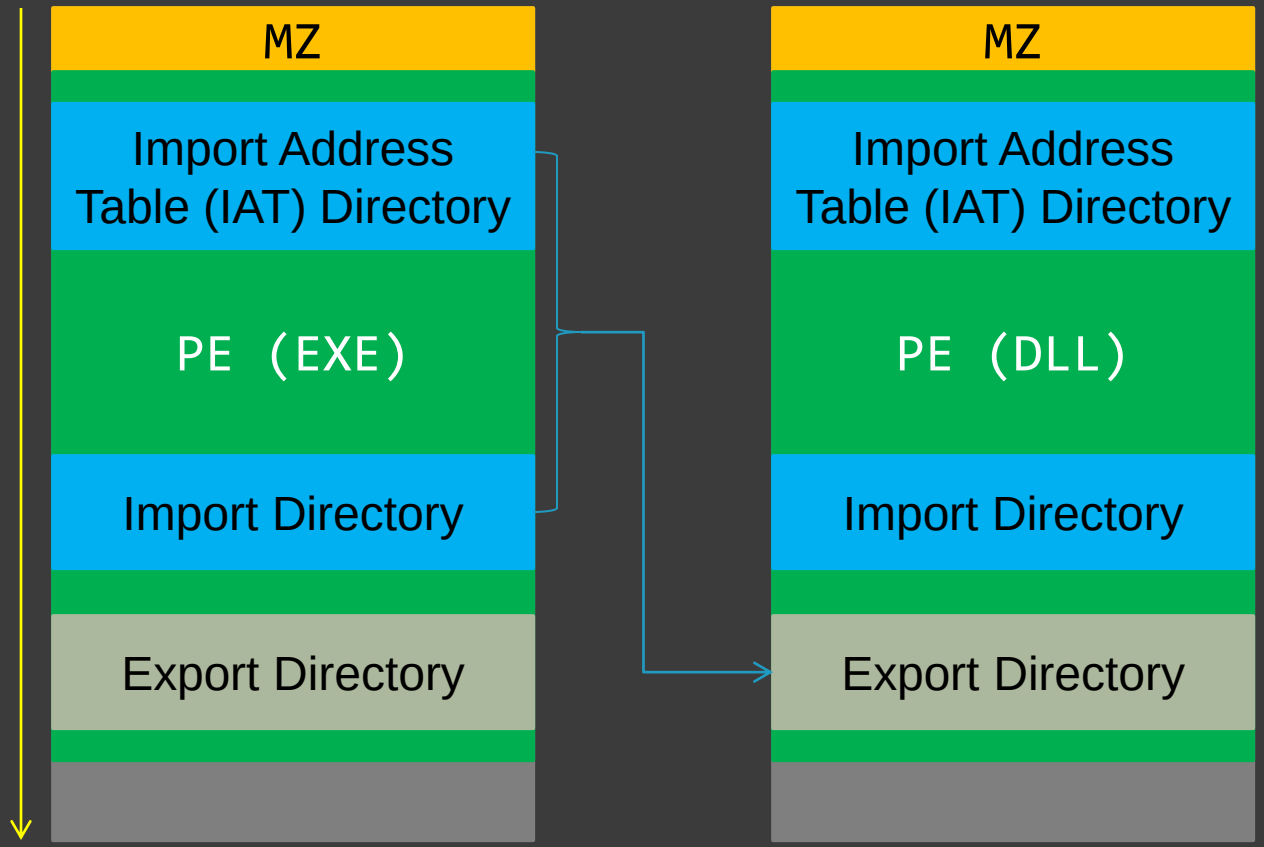



```
0:000> lm
start             end                 module name
00007ffb`7ed20000 00007ffb`7ed3f000   M1DLL
[…]

0:000> dc 00007ffb`7ed20000  L40
00007ffb`7ed20000  00905a4d 00000003 00000004 0000ffff  MZ..............
[…]

0:000> !dh 00007ffb`7ed20000
[…]
```

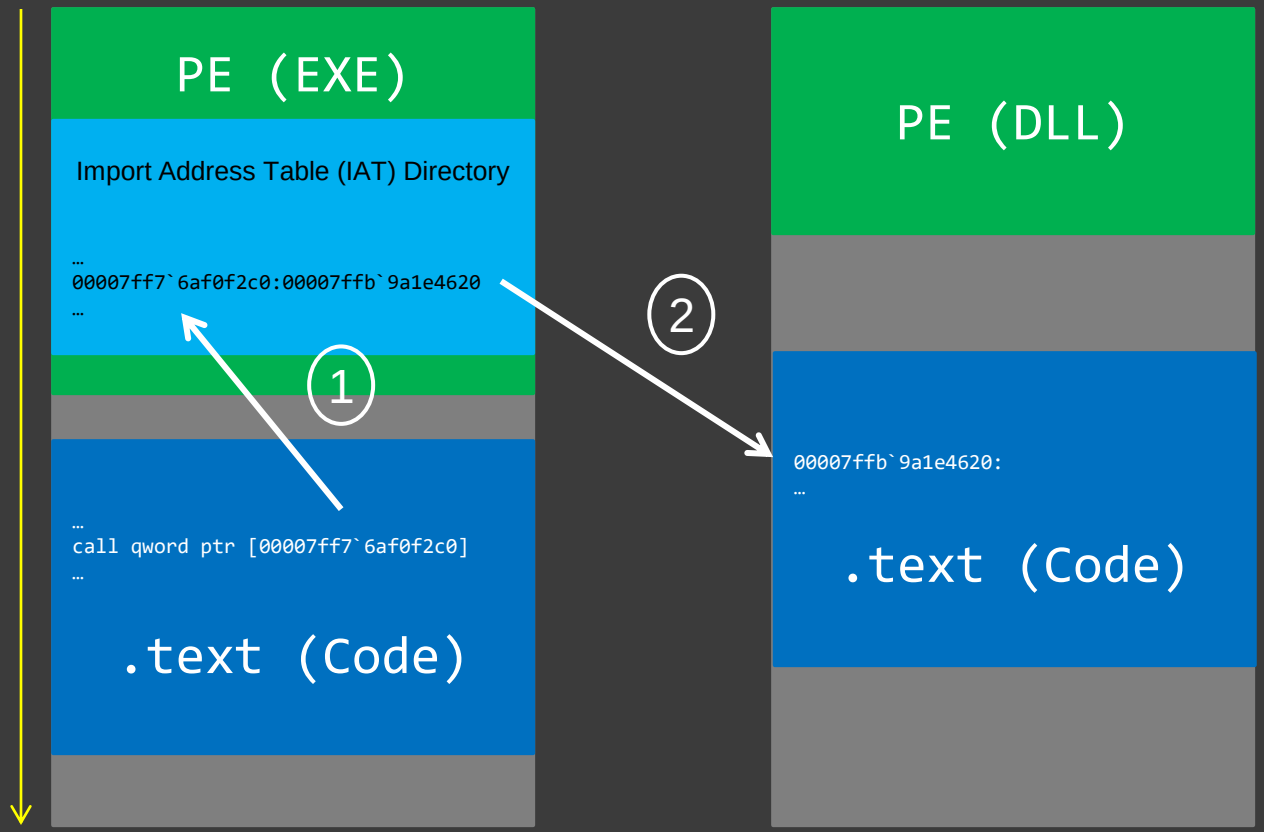
Exercise M1A

- **Goal:** Look at module headers and version information before loading
- **Patterns:** Unknown Module
- \AWMA-Dumps\Exercise-M1A.pdf

Dynamic Linking Design



After Dynamic Linking



Exercise M1B

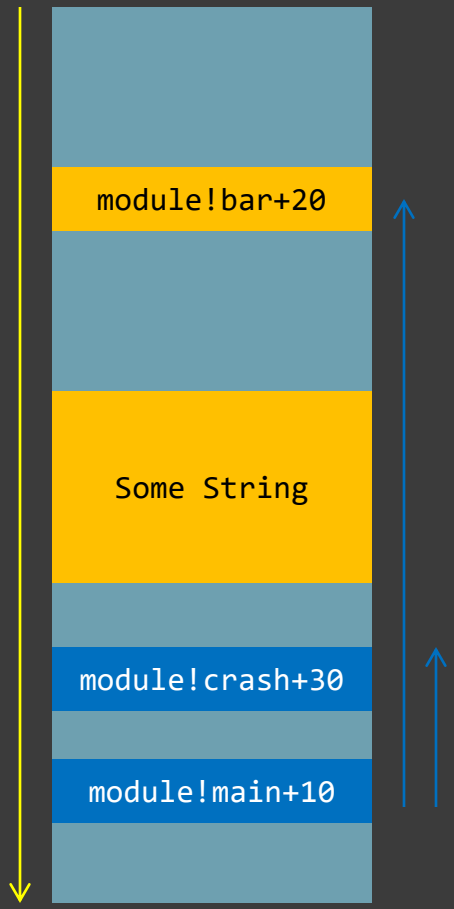
- **Goal:** Look at address map, module headers and version information after load, check IAT, check import library calls, and check module integrity
- **Patterns:** Unknown Module
- \AWMA-Dumps\Exercise-M1B.pdf

Packed Code and Data

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

Example: UPX

Thread Raw Stack Data



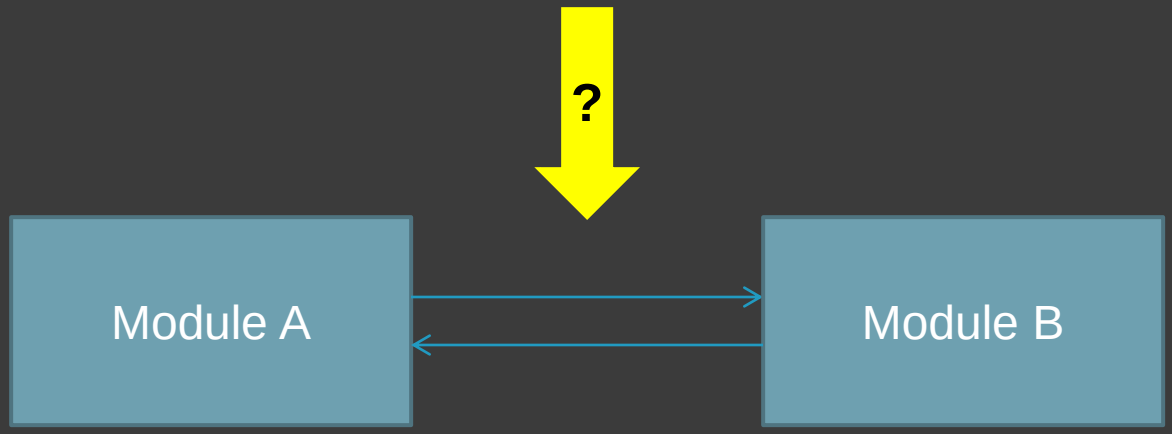

```
void main()
{
     foo();
     crash();
}

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

void bar()
{
     do();
}

void crash()
{
     WER();
}
```

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

Exercise M2

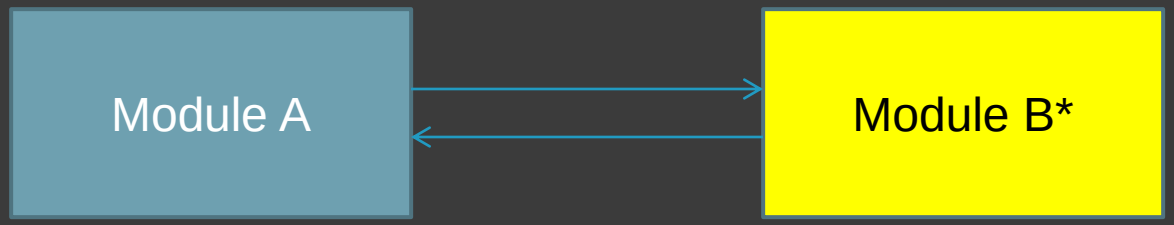
- **Goal:** Diagnose packed and hidden modules and their execution residues
- **Patterns:** Packed Code, Hidden Module, Pre-Obfuscation Residue, Execution Residue, String Hint
- \AWMA-Dumps\Exercise-M2.pdf

Malware Requirements



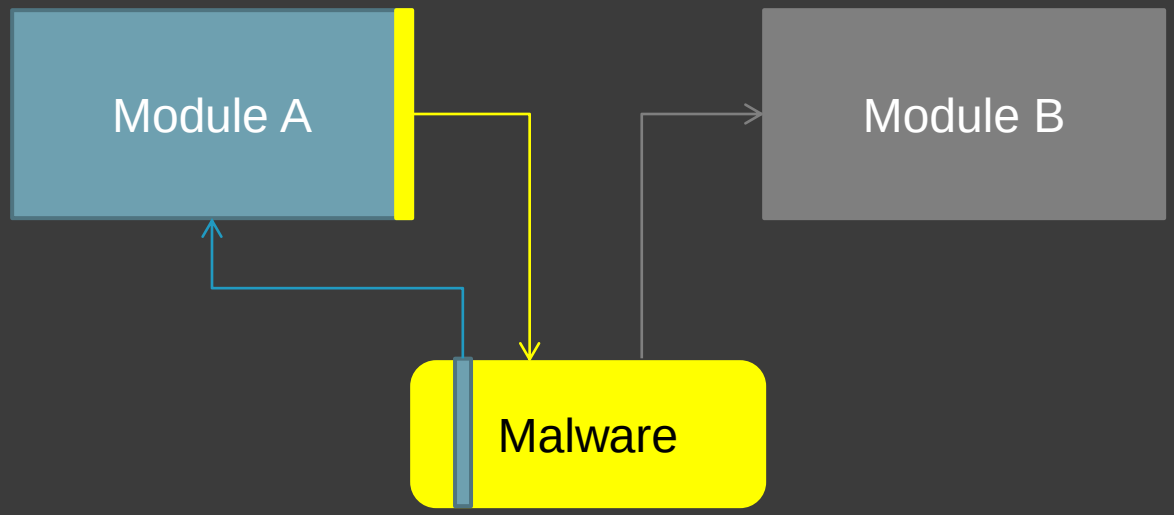
Malware Architecture

- Before load



- After load: Hooksware

Hooksware (Patching)




```
0:004> u wininet!InternetReadFile
wininet!InternetReadFile:
7758654b e98044ac88      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
[…]
```

1

2

```
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, Hooksware, 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

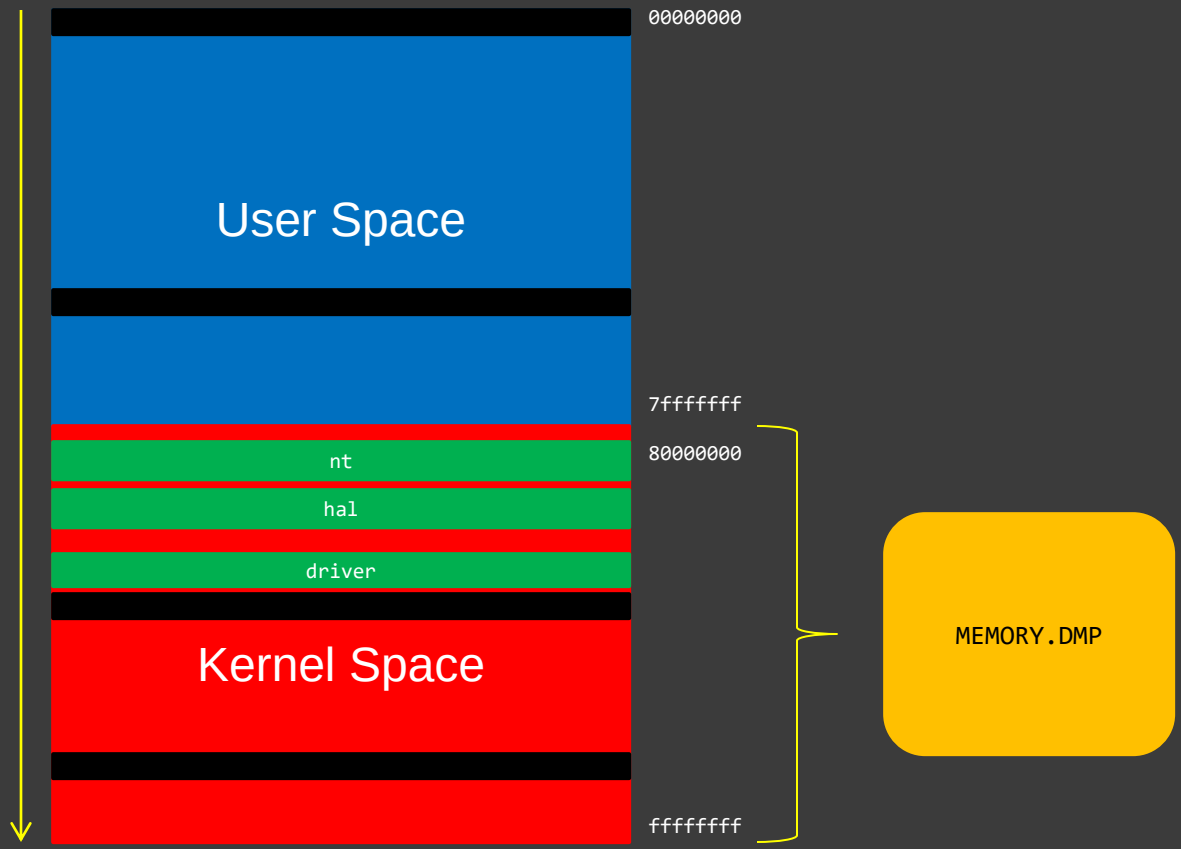
Pattern Links

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

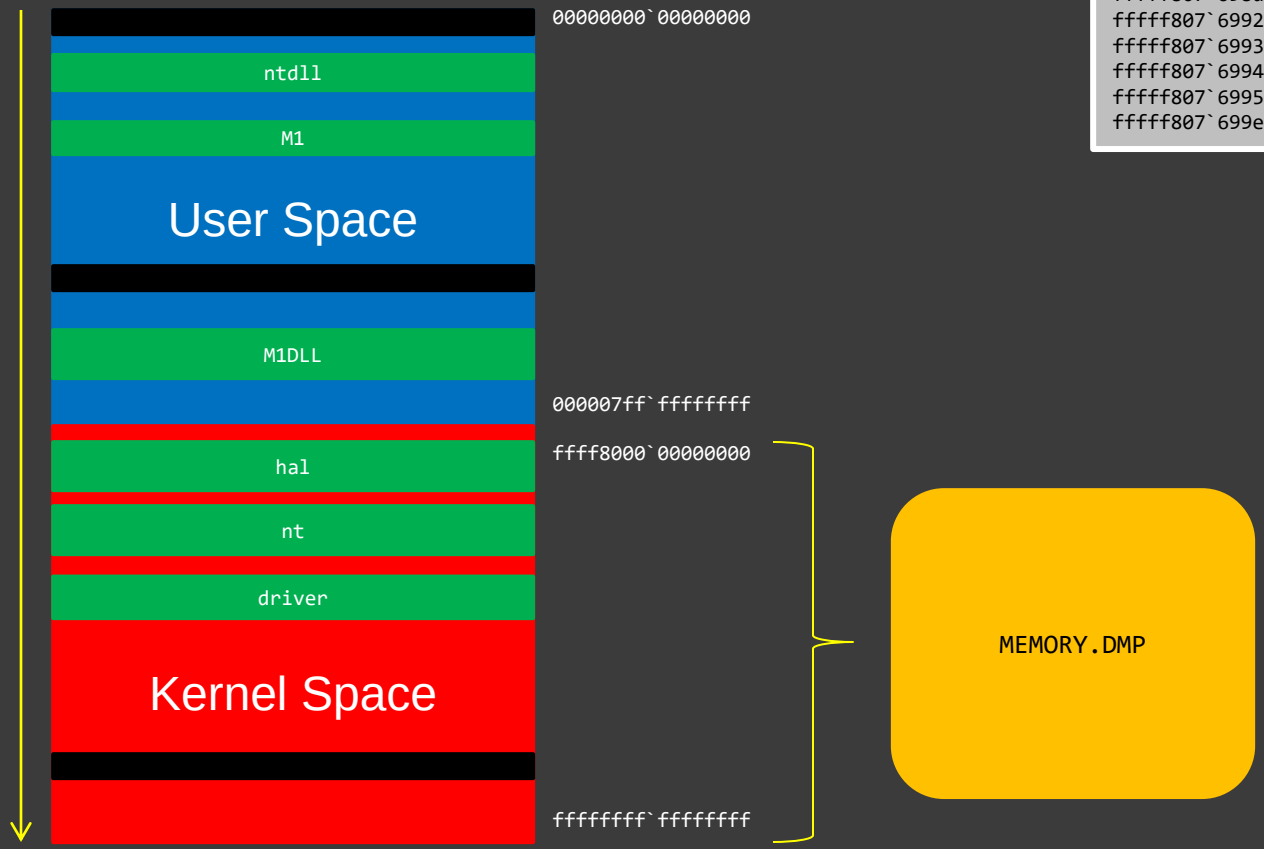
Space Review (x86)




```
0: kd> lmk
start    end        module name
80200000 8020a000   BATTC
8020a000 8020c900   compbatt
8020d000 80215000   msisadrv
80215000 8021e000   WMILIB
8021e000 8022b000   WDFLDR
8022b000 80266000   CLFS
80266000 8026e000   BOOTVID
[…]
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
```

Space Review (x64)

```
0: kd> lmk
start             end                 module name
ffffbc92`8dc40000 ffffbc92`8dcea000   win32k
ffffbc92`8e110000 ffffbc92`8e451000   win32kbase
ffffbc92`8e6d0000 ffffbc92`8ea82000   win32kfull
ffffbc92`8ea90000 ffffbc92`8ead3000   cdd
fffff807`608f0000 fffff807`60c73000   mcupdate_GenuineIntel
fffff807`60ca0000 fffff807`60ca6000   hal
fffff807`60cb0000 fffff807`60cbb000   kd
[…]
fffff807`698c0000 fffff807`698cf000   dump_diskdump
fffff807`698d0000 fffff807`6991c000   intelppm
fffff807`69920000 fffff807`6992d000   NdisVirtualBus
fffff807`69930000 fffff807`6993c000   swenum
fffff807`69940000 fffff807`6994f000   rdpbus
fffff807`69950000 fffff807`699d4000   usbhub
fffff807`699e0000 fffff807`699ee000   USBD
```



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 fffffa80033b5b50  Cid 0004.0030  Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT:
(WrPushLock) KernelMode Non-Alertable
 fffff880021d9750  SynchronizationEvent
 Not impersonating
 DeviceMap                 fffff8a0000088f0
 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

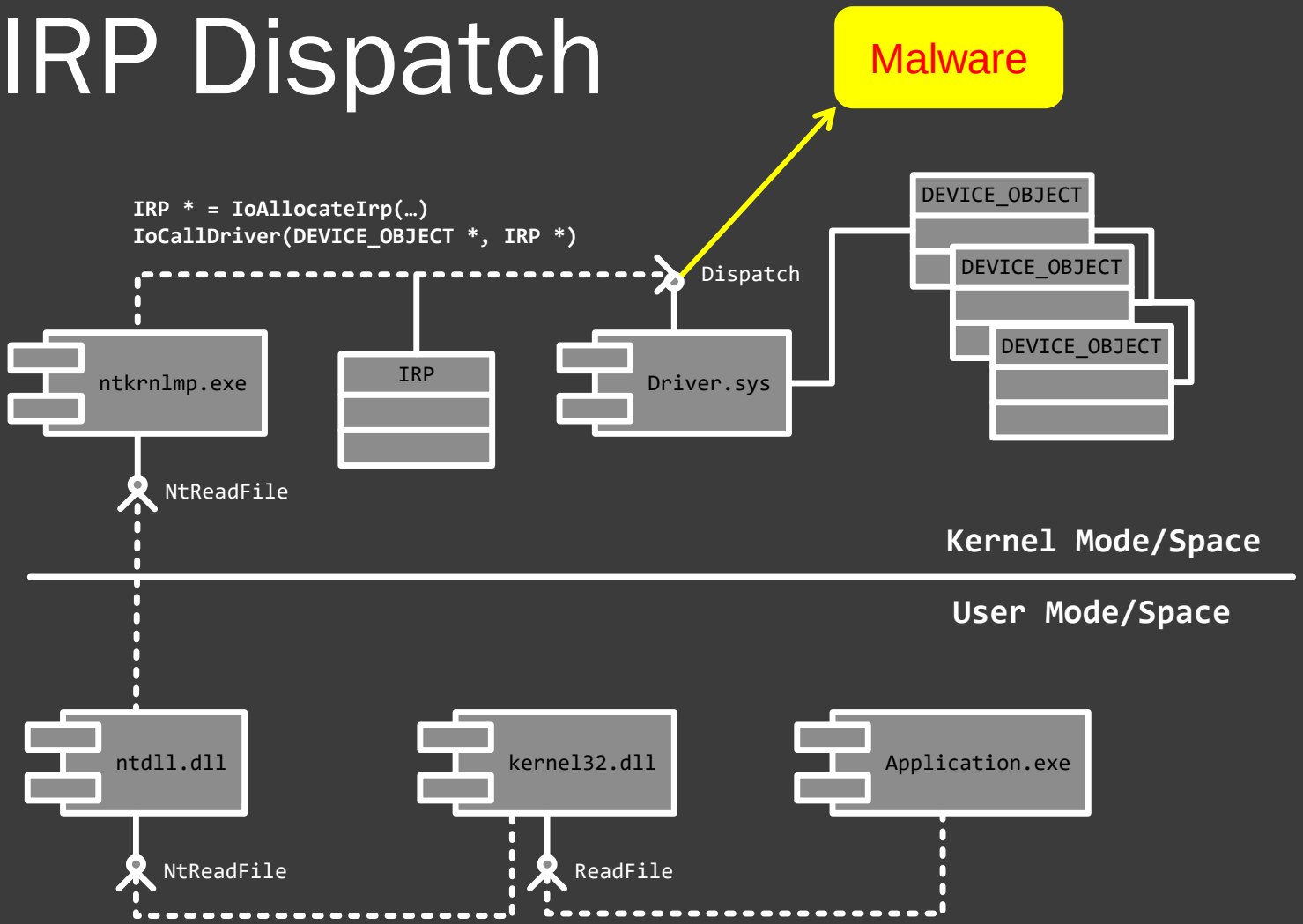
User Space/Mode

```
1: kd> uf ntdll!NtReadFIle
ntdll!NtReadFile:
00007ffe`5b023800 4c8bd1          mov     r10,rcx
00007ffe`5b023803 b806000000      mov     eax,6
00007ffe`5b023808 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`5b023810 7503            jne     ntdll!NtReadFile+0x15 (00007ffe`5b023815)  Branch

ntdll!NtReadFile+0x12:
00007ffe`5b023812 0f05            syscall
00007ffe`5b023814 c3              ret

ntdll!NtReadFile+0x15:
00007ffe`5b023815 cd2e            int     2Eh
00007ffe`5b023817 c3              ret
```

Kernel Space/Mode

```
1: kd> u nt!KiServiceTable + (dwo(nt!KiServiceTable+4*6) >>> 4) L1
nt!NtReadFile:
fffff807`62780750 4c8bdc          mov     r11,rsp
```

IRP Dispatch

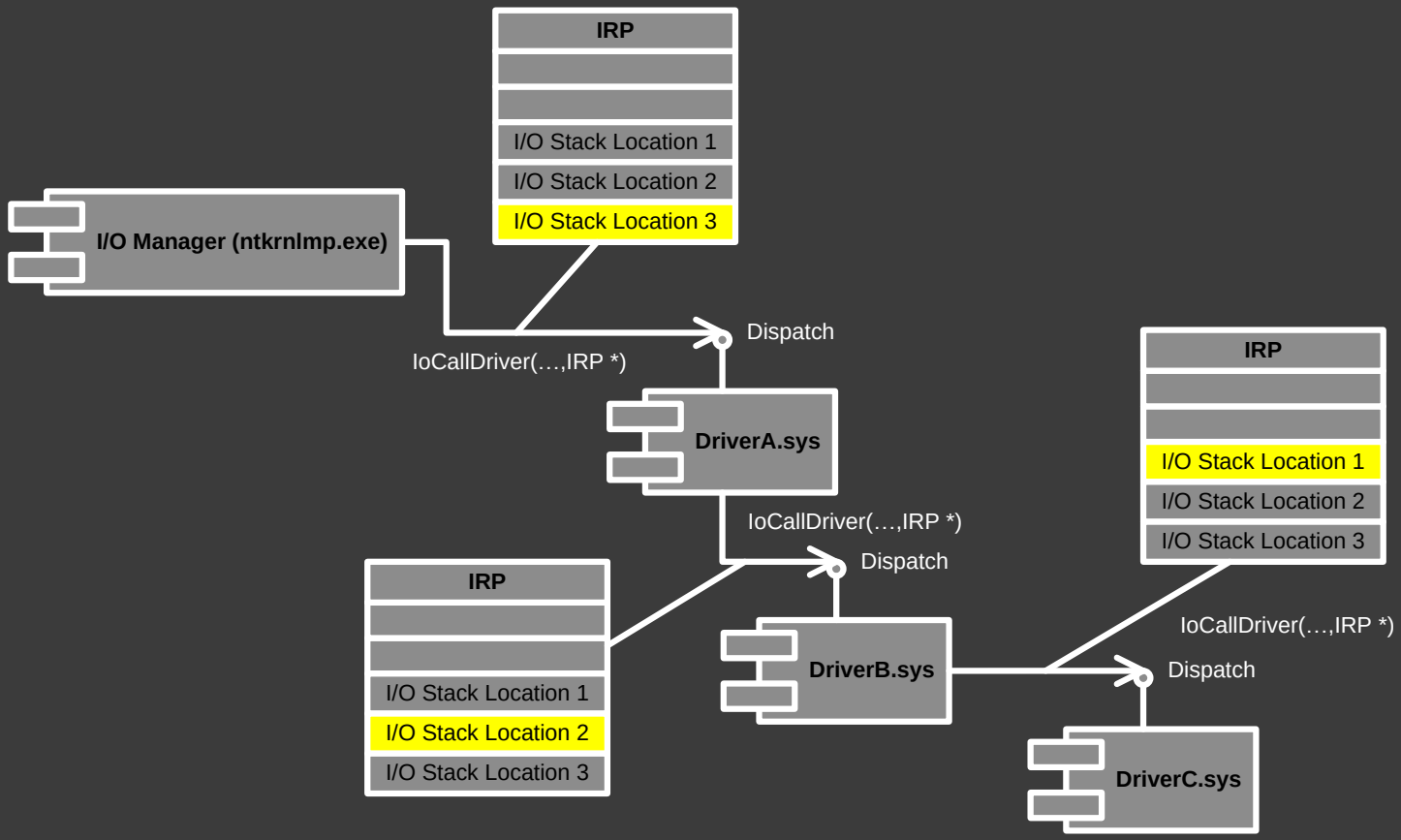


Device Driver Example

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

Driver Extension List: (id , addr)

Device Object list:
ffffbe0c8784c790

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
[…]
```

```
0: kd> !irpfind -v
```

IRP Communication



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

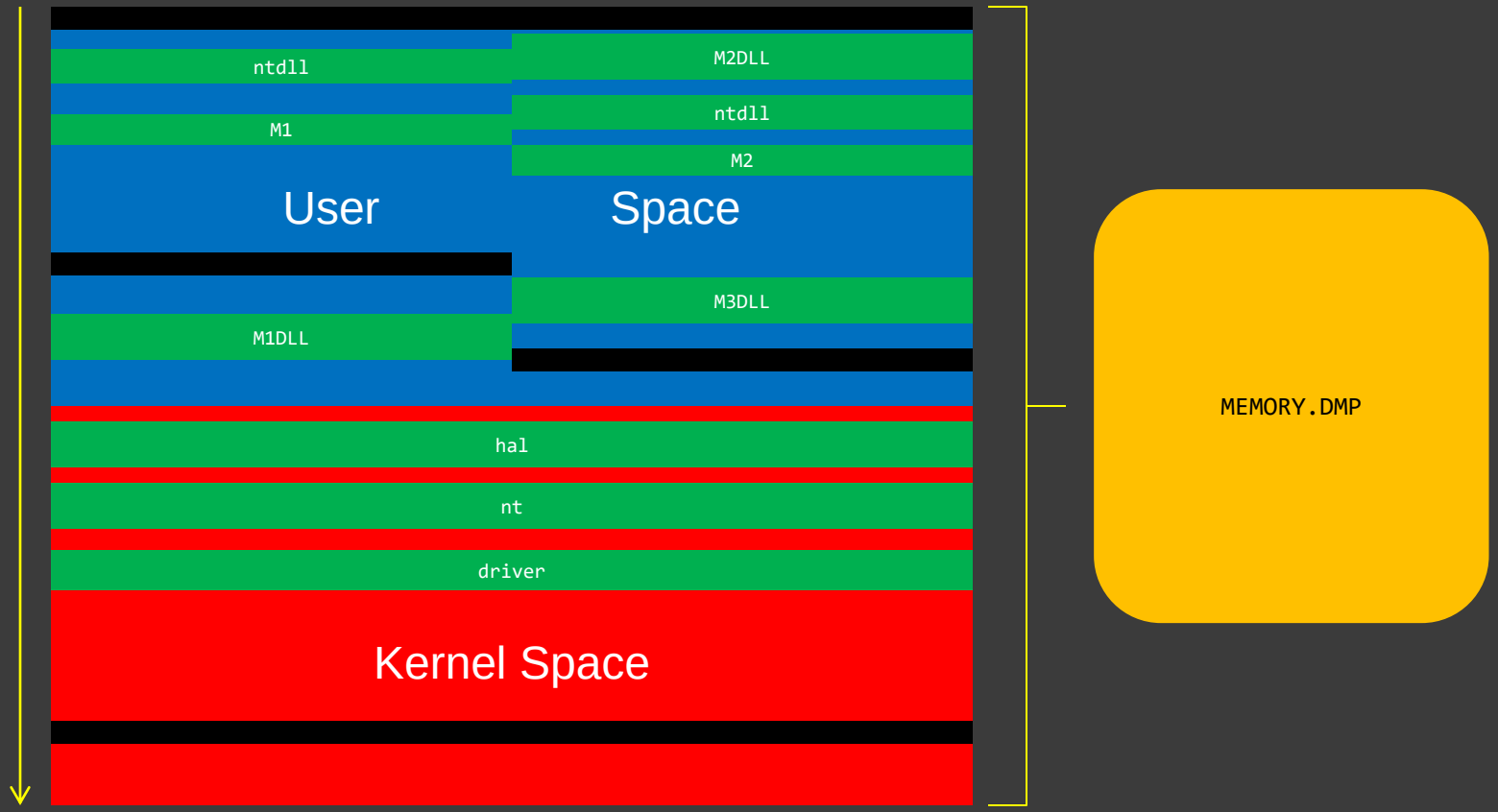
Direct Dump Manipulation

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

Physical Space Memory

Space Review

```
0: kd> !process <address> 3f

0: kd> .process /r /p <address>

0: kd> !thread <address> 3f

0: kd> .thread /r /p <address>

0: kd> .thread /w <address>
```



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

https://www.patterndiagnostics.com/files/LegacyWindowsDebugging.pdf

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)
- Windows Internals, 6th ed., 7th ed.
- 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!