Public Preview

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

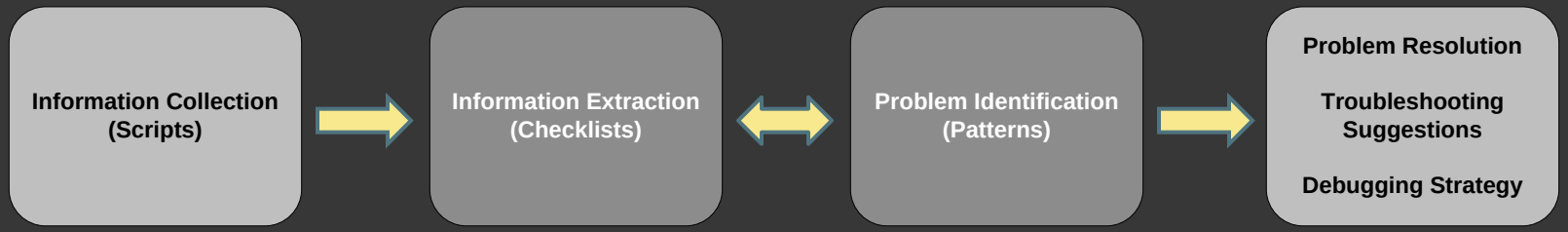
Pattern-Oriented Diagnostic Analysis

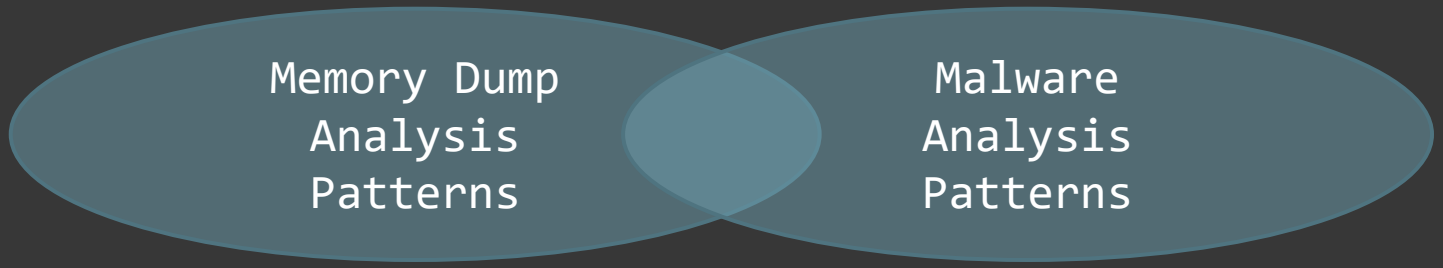
Diagnostic Pattern: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

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, ...





Practice Exercises

Links

- Memory Dumps

NOT IN THE PUBLIC PREVIEW VERSION

- Exercise Transcripts

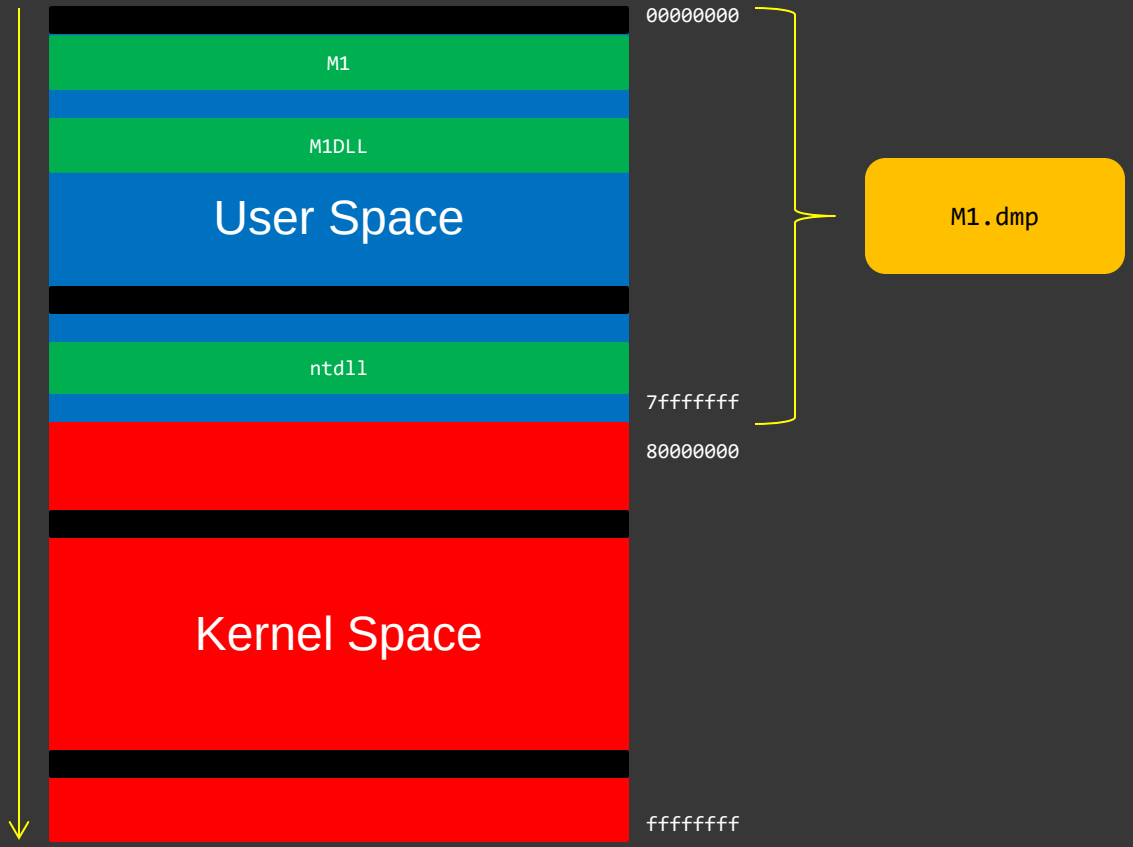
NOT IN THE PUBLIC PREVIEW VERSION

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

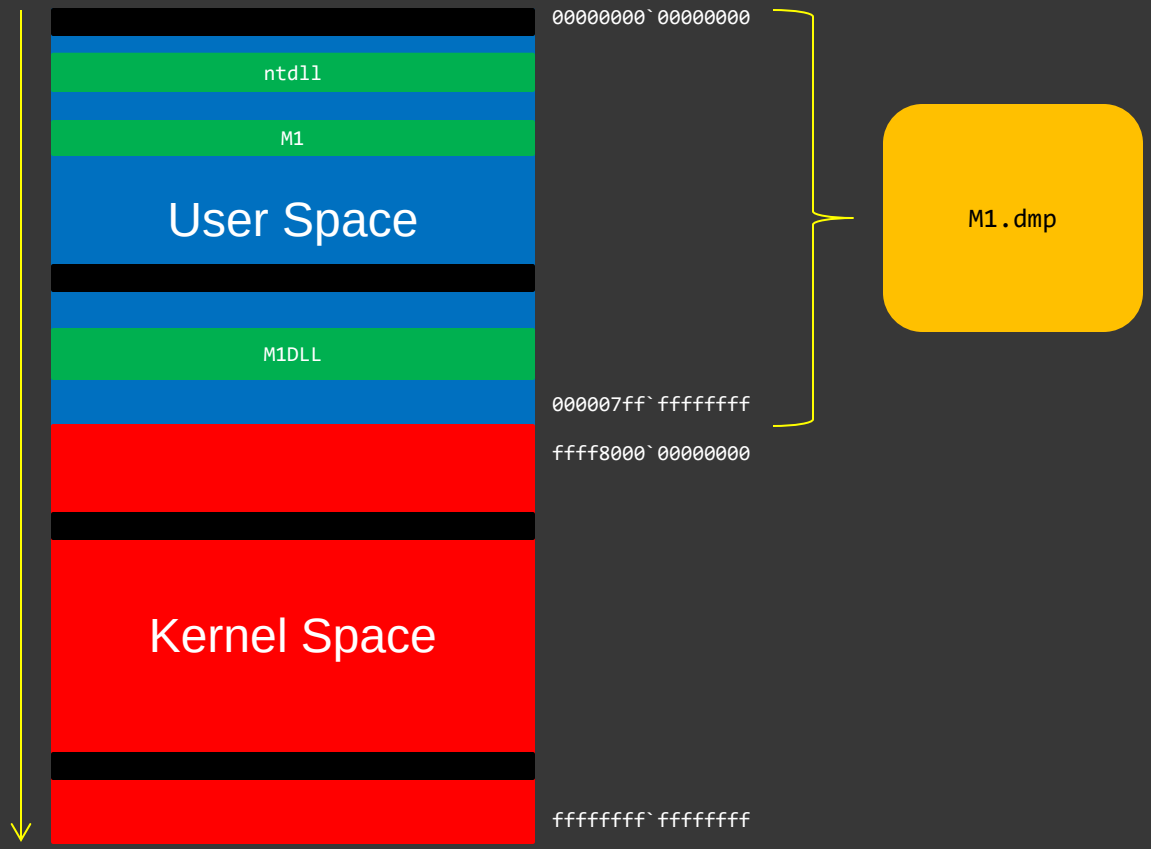
User Space Memory

Space Review (x86)




```
0:000> lm
start    end        module name
013e0000 013f5000   M1
10000000 10039000   WinCRT
70120000 70134000   M1DLL
72a40000 72a91000   winspool
738b0000 73930000   uxtheme
73ec0000 73ed3000   dwmapi
746b0000 746bc000   CRYPTBASE
746c0000 74720000   sspicli
74a00000 74a83000   clbcatq
74a90000 74b20000   gdi32
74b20000 74baf000   oleaut32
74bb0000 74bba000   lpk
74bc0000 74c6c000   msvcrt
74ca0000 74d6c000   msctf
750d0000 75170000   advapi32
751e0000 752f0000   kernel32
752f0000 75309000   sechost
75390000 753d7000   KERNELBASE
753e0000 75437000   shlwapi
760d0000 7622c000   ole32
76230000 762cd000   usp10
764d0000 765c0000   rpcrt4
765c0000 766c0000   user32
766c0000 76720000   imm32
76fe0000 77160000   ntdll
```

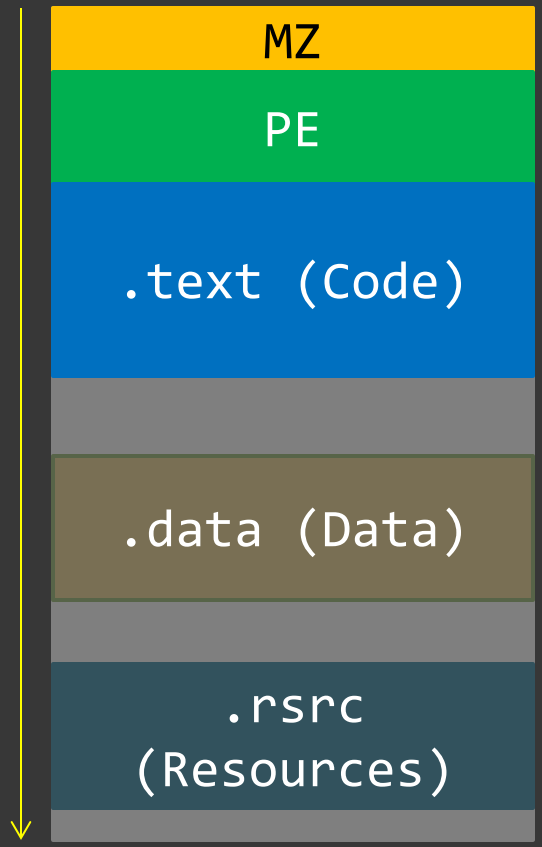
Space Review (x64)




```
0:000> lm
start             end                 module name
00000000`76be0000 00000000`76cff000   kernel32
00000000`76d00000 00000000`76dfa000   user32
00000000`76e00000 00000000`76fa9000   ntdll
00000001`3f8a0000 00000001`3f8b8000   M1
000007fe`f9d20000 000007fe`f9d37000   M1DLL
000007fe`fb700000 000007fe`fb718000   dwmapi
000007fe`fb990000 000007fe`fb9e6000   uxtheme
000007fe`fd190000 000007fe`fd19f000   CRYPTBASE
000007fe`fd4e0000 000007fe`fd54c000   KERNELBASE
000007fe`fd6b0000 000007fe`fd749000   clbcatq
000007fe`fd940000 000007fe`fda1b000   advapi32
000007fe`fda20000 000007fe`fdb4d000   rpcrt4
000007fe`fdb50000 000007fe`fdbb7000   gdi32
000007fe`fdbc0000 000007fe`fddc3000   ole32
000007fe`fece0000 000007fe`fed0e000   imm32
000007fe`fed10000 000007fe`fed1e000   lpk
000007fe`fed20000 000007fe`fee29000   msctf
000007fe`feea0000 000007fe`feebf000   sechost
000007fe`feec0000 000007fe`fef89000   usp10
000007fe`fef90000 000007fe`ff02f000   msvcrt
000007fe`ff030000 000007fe`ff107000   oleaut32
```

EXE/DLL/SYS

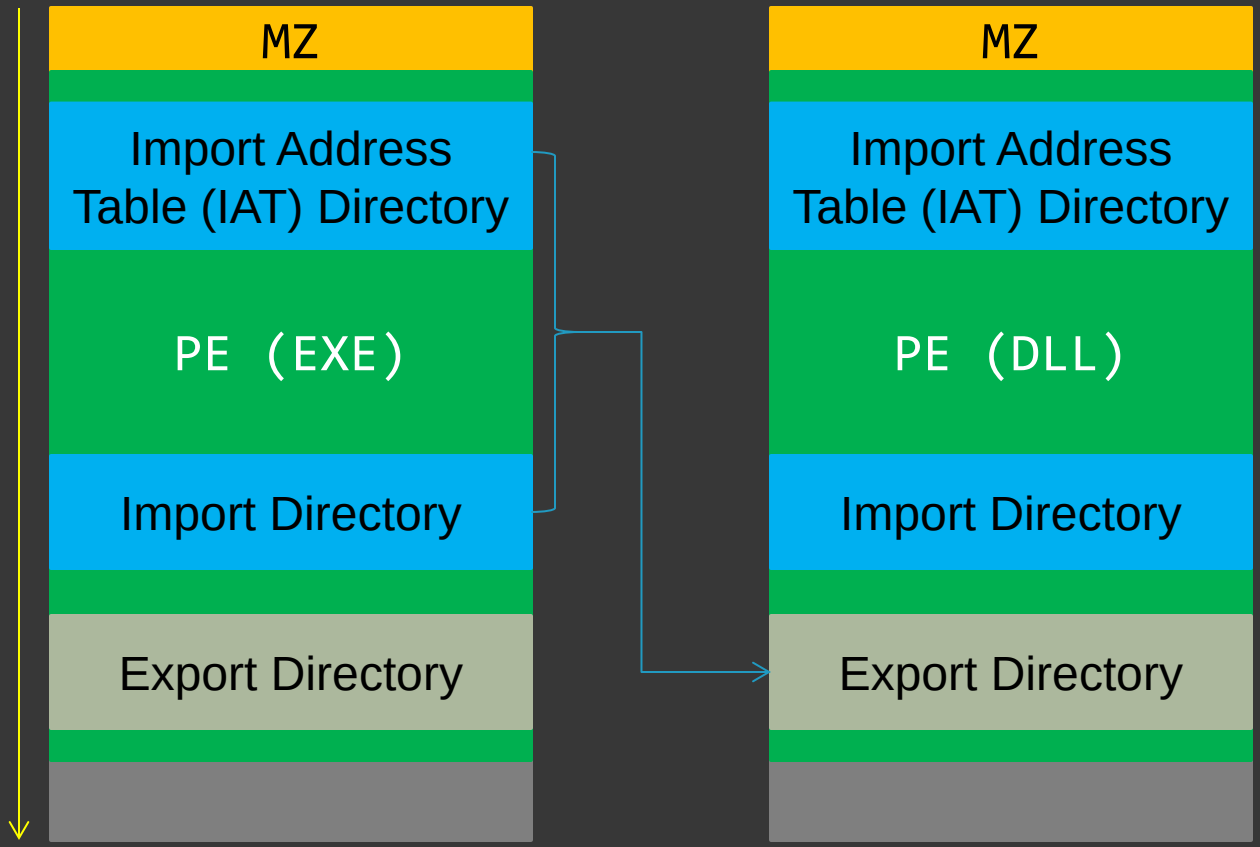
```
0:000> lm
start             end                 module name
00000001`80000000 00000001`80017000   M1DLL
[…]

0:000> dc 00000001`80000000 l40
00000001`80000000  00905a4d 00000003 00000004 0000ffff  MZ..............
[…]

0:000> !dh 00000001`80000000
[…]
```



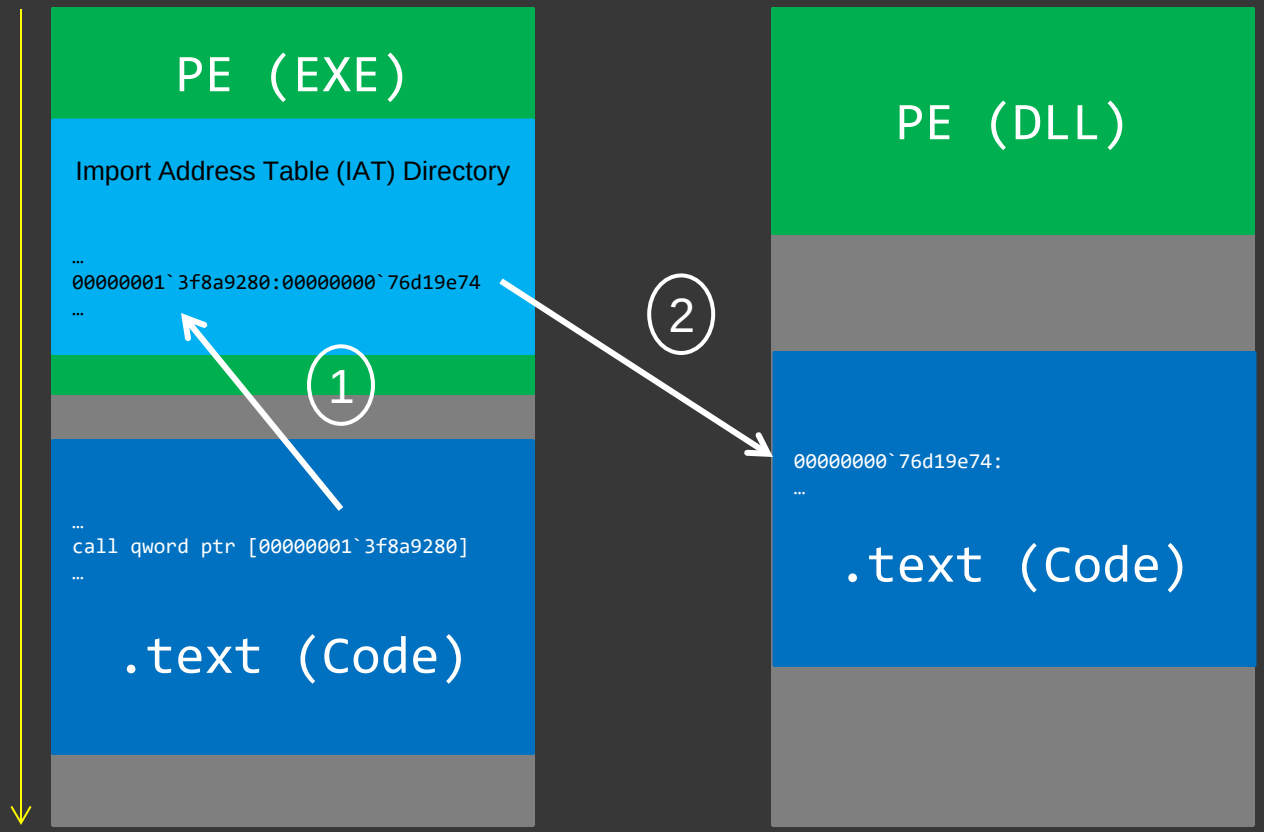
Exercise M1A

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

Dynamic Linking Design



After Dynamic Linking



Exercise M1B

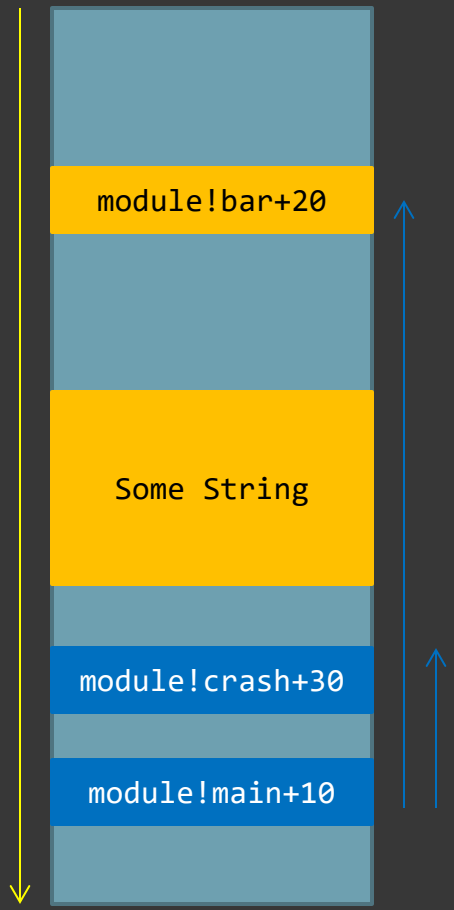
- **Goal:** Look at address map, module headers and version information after load, check IAT, check import library calls, check module integrity
- **Patterns:** Unknown Module
- \AWMA-Dumps\Exercise-M1B.pdf

Packed Code and Data

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

Example: UPX

Thread Raw Stack Data



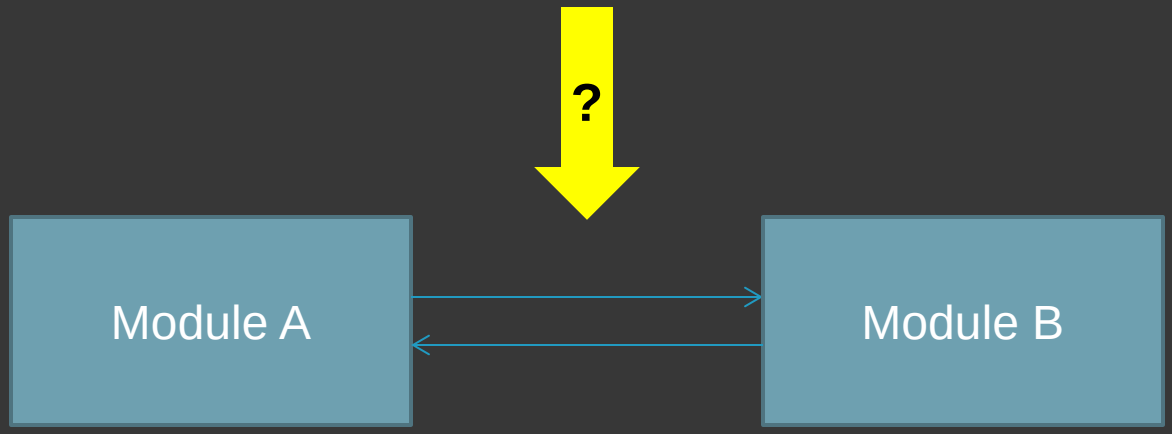

```
void main()
{
     foo();
     crash();
}

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

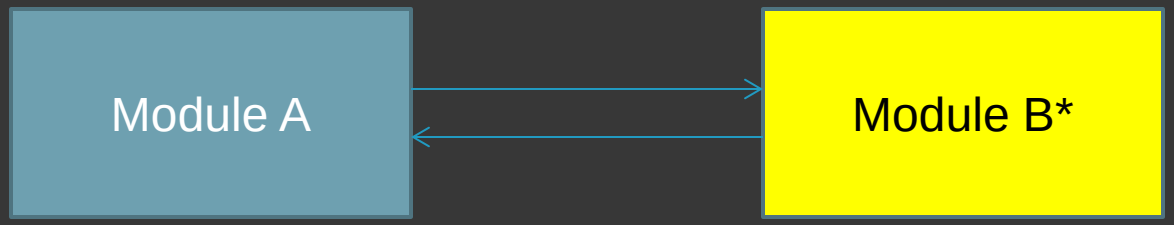
- **Goal:** Diagnose packed and hidden modules and their execution residues
- **Patterns:** Packed Code, Hidden Module, Pre-Obfuscation Residue, Execution Residue, String Hint
- \AWMA-Dumps\Exercise-M2.pdf

Malware Requirements



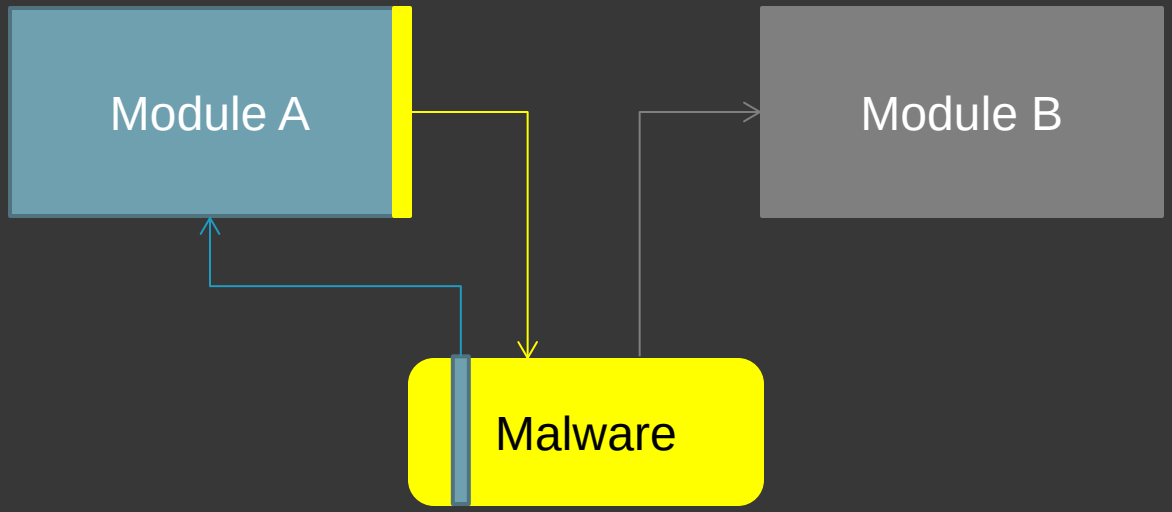
Malware Architecture

- 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

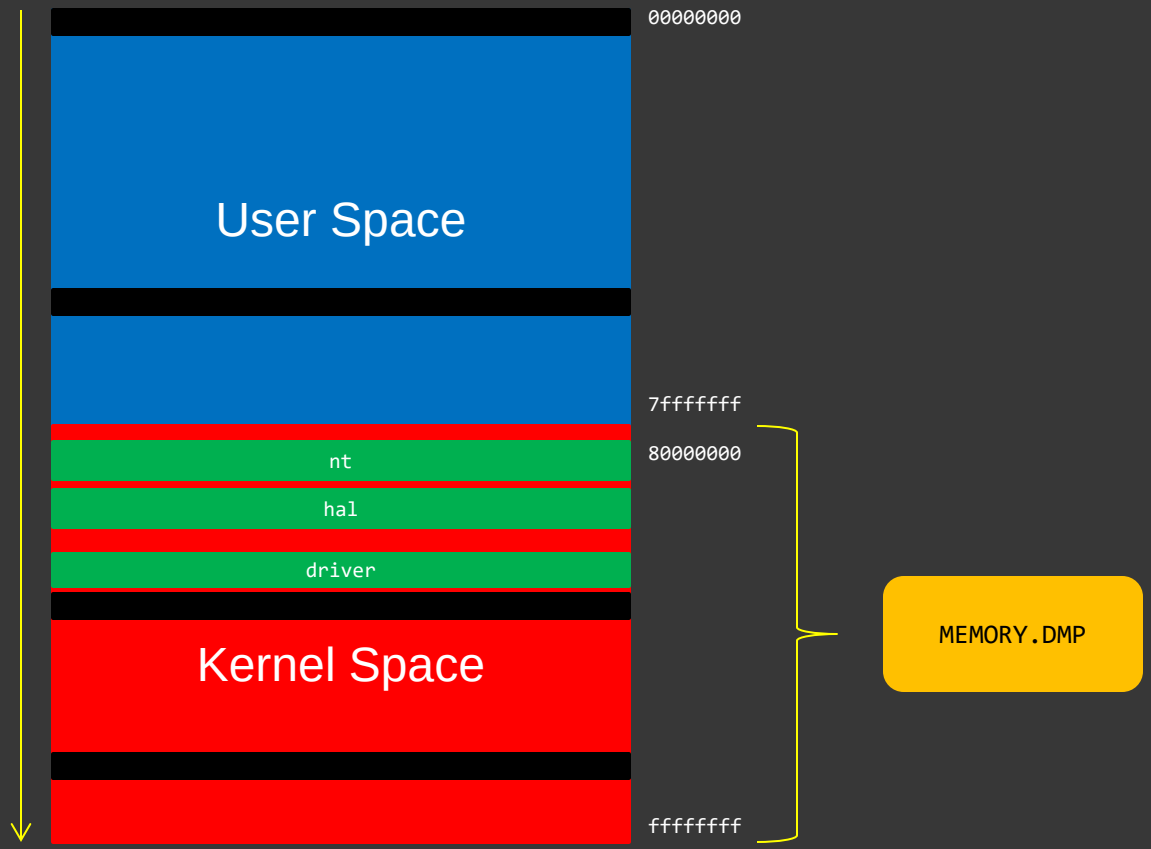
- Import Address Table
- System call dispatch
- Exception handling

Pattern Links

Stack Trace Collection
RIP Stack Trace
Hooksware
Hidden Module
String Hint
Fake Module
Patched Code
Packed Code
No Component Symbols
Pre-Obfuscation Residue
Deviant Module
Unknown Module
Execution Residue
Namespace

Kernel Space Memory

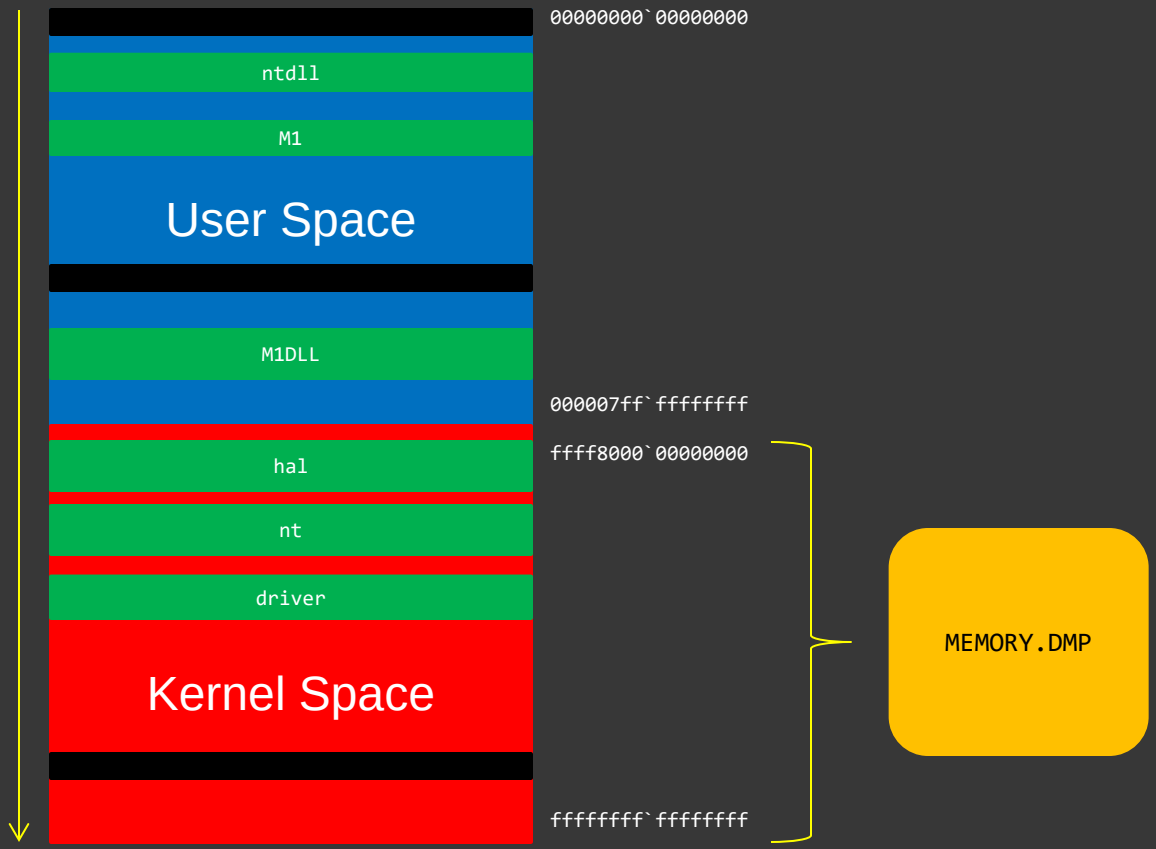
Space Review (x86)




```
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
fffff802`b309f000 fffff802`b30a8000   kd
fffff802`b3a1d000 fffff802`b3a89000   hal
fffff802`b3a89000 fffff802`b41d2000   nt
[…]
fffff880`01a2f000 fffff880`01a4b000   disk
fffff880`01a53000 fffff880`01c36000   Ntfs
fffff880`01c36000 fffff880`01c51000   ksecdd
[…]
fffff960`0007a000 fffff960`0046f000   win32k
fffff960`006d1000 fffff960`006da000   TSDDD
fffff960`008a4000 fffff960`008da000   cdd
```



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`
- `!sprocess <session> 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

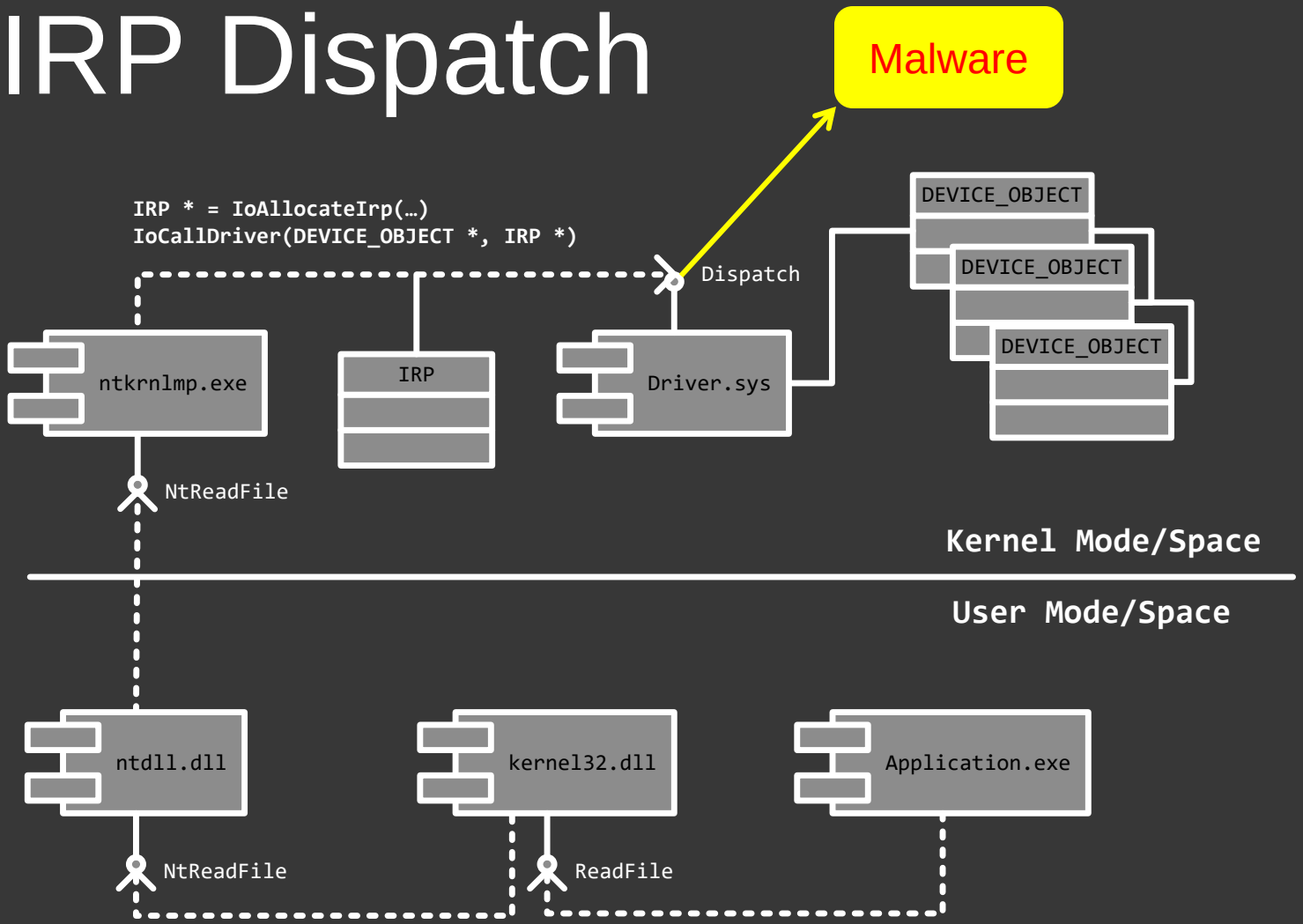
User Space/Mode

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

Kernel Space/Mode

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

IRP Dispatch

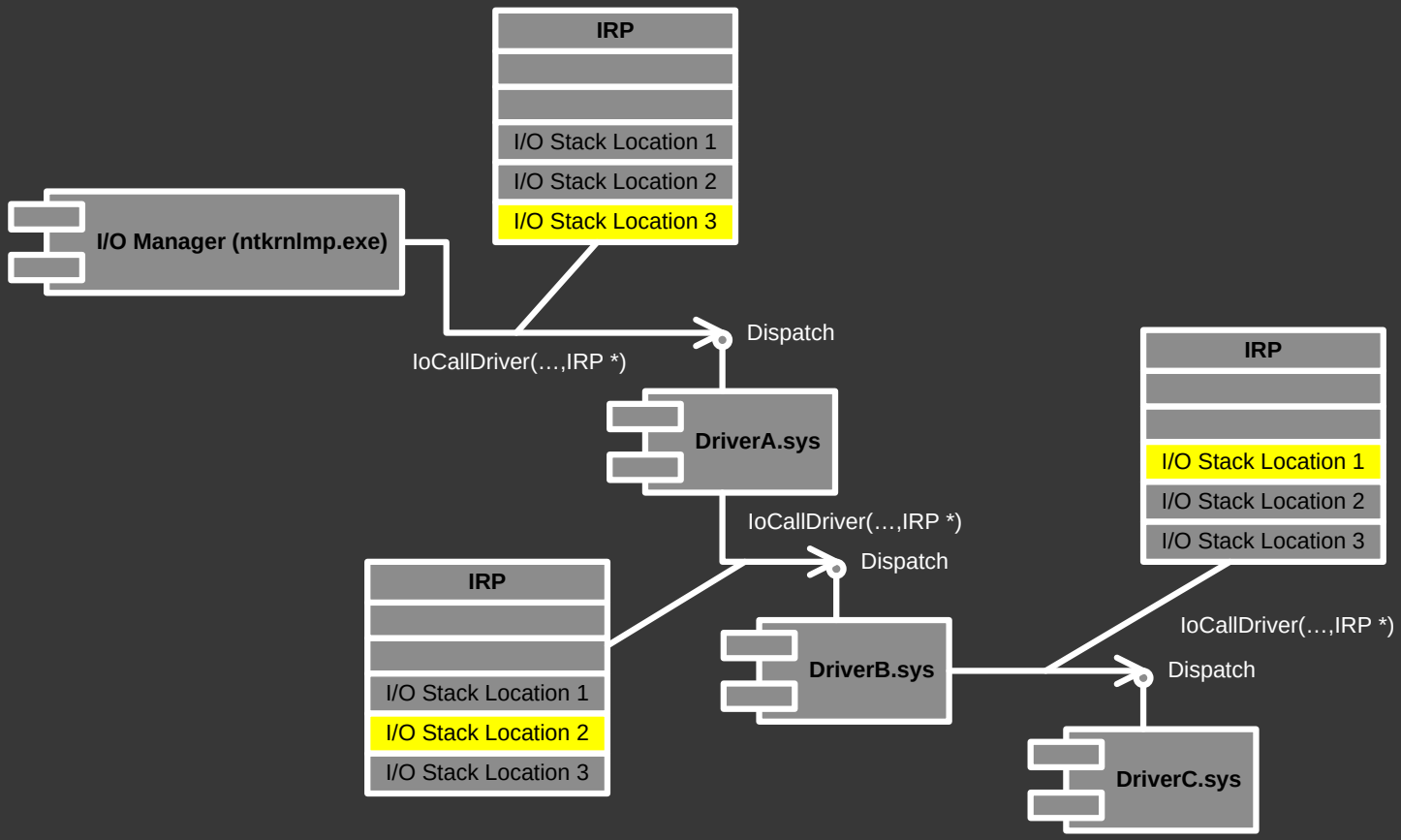


Device Driver Example

```
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_NAMED_PIPE         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
[…]
```

```
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

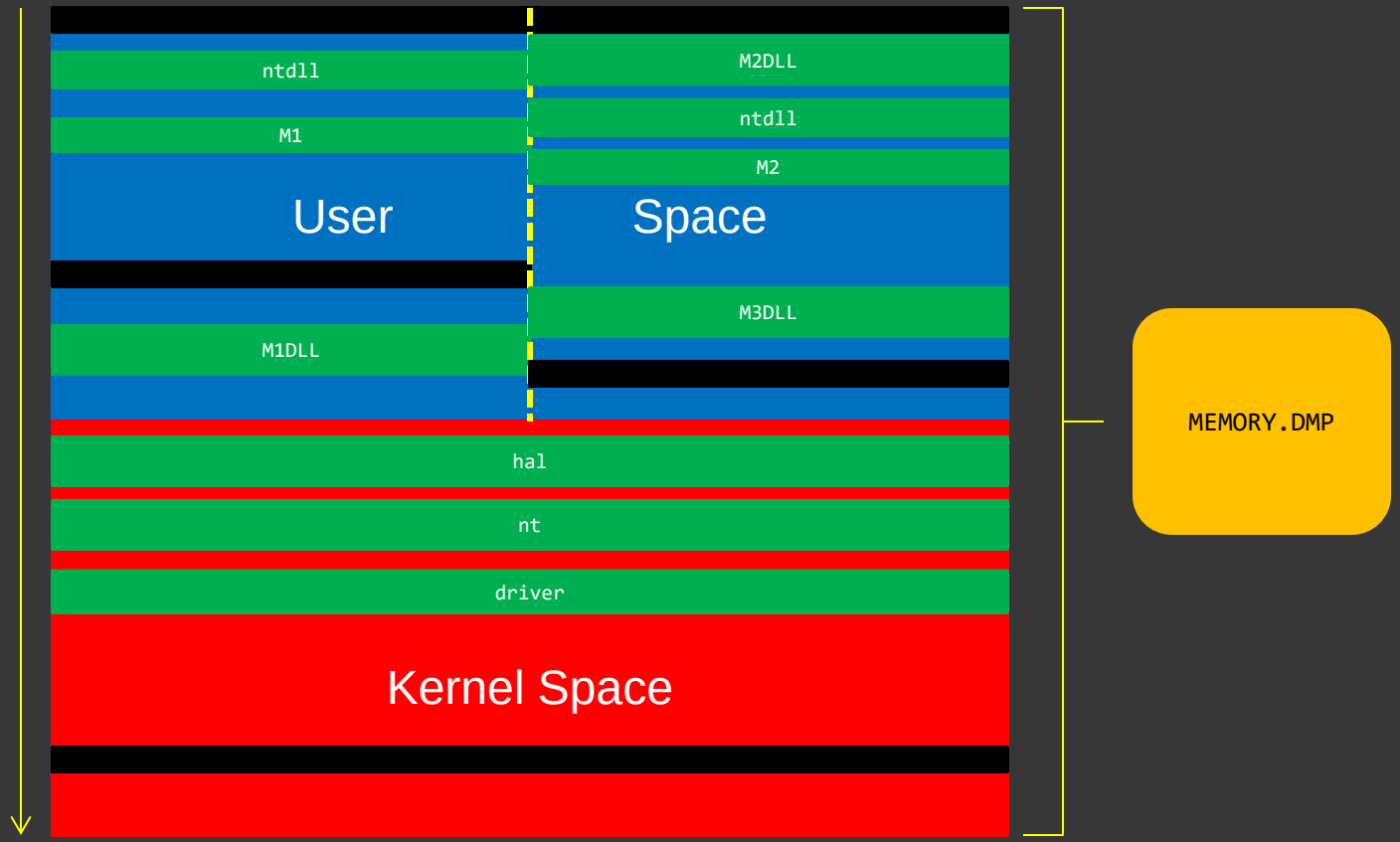
Direct Dump Manipulation

- 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, check IRP stacks
- **Patterns:** Deviant Token, Hidden Process, Hidden Module, Stack Trace Collection (I/O)
- \AWMA-Dumps\Exercise-M6.pdf

Memory Acquisition

http://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
- The Rootkit Arsenal (2nd edition)
- Windows Internals, 6th ed.
- Practical Foundations of Windows Debugging, Disassembling, Reversing
- Memory Dump Analysis Anthology (Volumes 1 – 10)

Q&A

Please send your feedback using the contact form on PatternDiagnostics.com

Thank you for attendance!