

Kernel Pool Monitoring for Malware Forensics in a Cloud Computing Environment

Irfan Ahmed

Assistant Professor of Computer Science
University of New Orleans

iahmed4@uno.edu

Golden G. Richard III

Professor of Computer Science
University of New Orleans

Arcane Alloy, LLC

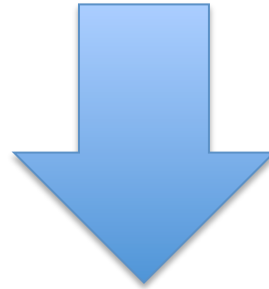
golden@cs.uno.edu

Disclosure / ACKs

- The talk mentions the following commercial software, historical and current:
 - Microsoft Windows
- We're users of Windows, but have no associated financial interest
- I'm not Irfan Ahmed, but he used to be my postdoc
- Finally, this work was partially supported by NSF grant CNS # 1016807.

Translation

“Kernel Pool Monitoring for
Malware Forensics in a Cloud
Computing Environment”



- Malicious software changes function pointers in the operating systems kernel to do bad stuff
- We'd love to detect those changes: Thus, HookLocator

Function Pointers: Example

```
for (int i = 0; i < arraysize(DriverObject->MajorFunction); ++i) {  
    DriverObject->MajorFunction[i] = DispatchAny;  
}
```

DriverObject->MajorFunction[IRP_MJ_POWER] =	DispatchPowerFilter;
DriverObject->MajorFunction[IRP_MJ_PNP] =	DispatchPnpFilter;
DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] =	DispatchInternal;
DriverObject->MajorFunction[IRP_MJ_CREATE] =	DispatchCreate;
DriverObject->MajorFunction[IRP_MJ_CLOSE] =	DispatchClose;
DriverObject->MajorFunction[IRP_MJ_CLEANUP] =	DispatchCleanup;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =	DispatchControl;

Source: <http://angry.harpy.ru/~geo/filter/sys/DriverEntry.cpp>

Monitoring Function Pointers

- Rootkits modify function pointers in kernel space to hijack or alter system control flow
- Monitoring and logging modifications can play a critical role in malware forensics
- Current practices only monitor well-known data structures in kernel space such as SSDT, IDT
- This leaves several dark areas in memory open for exploitation, particularly the OS kernel heap

Kernel Heap Exploitation

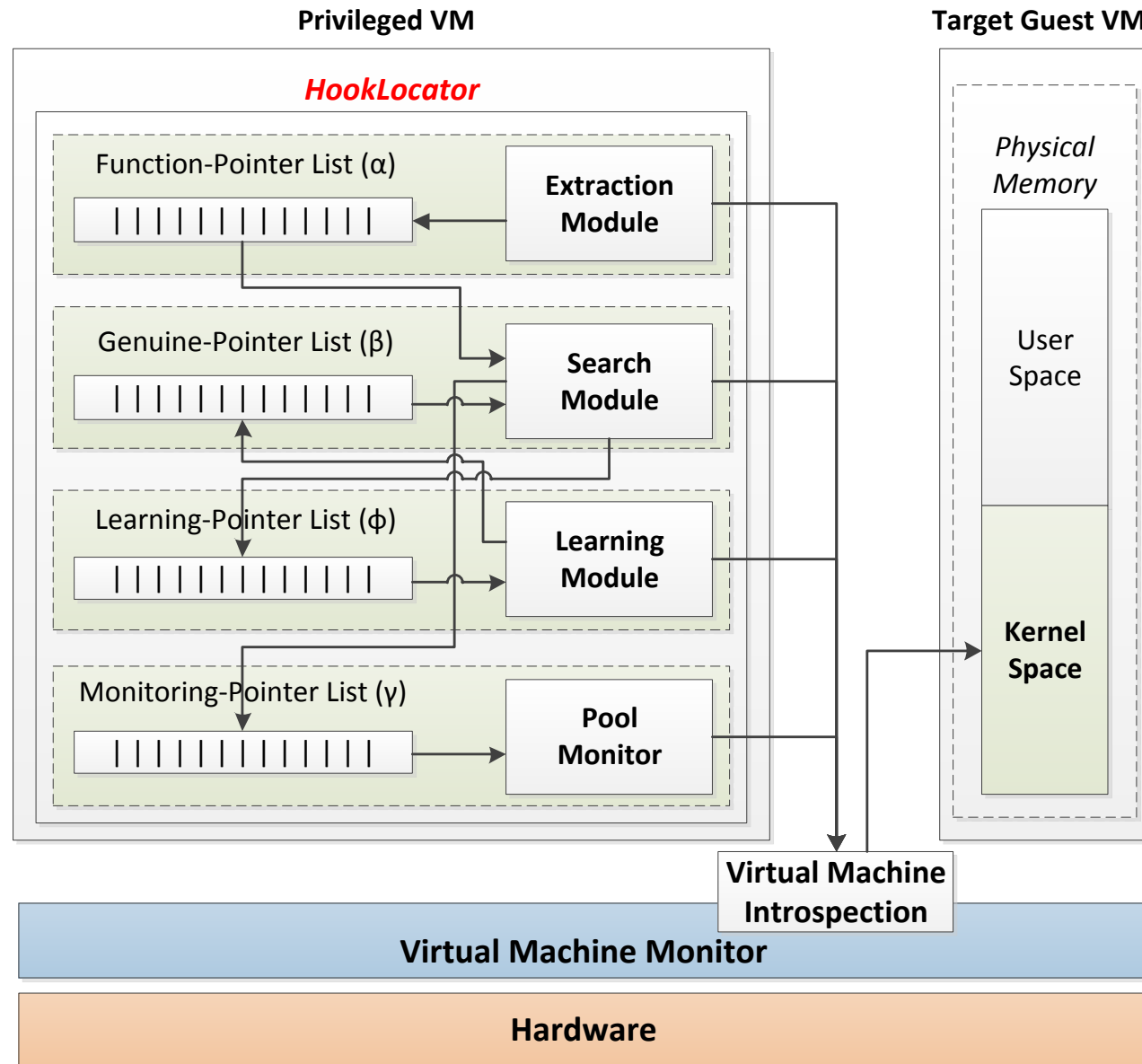
- Kernel heap contains chunks of dynamically allocated memory
 - Including complex data structures that may contain function pointers
- Function pointer exploitation is a real threat
 - Yin et al. demonstrated a keylogger by modifying just one function pointer in kernel space
- Windows kernel heap contains thousands of function pointers – providing a rich-attack surface

Yin et al., “HookScout: Proactive Binary-Centric Hook Detection”, in DIMVA 2010

Proposed Approach: HookLocator

- Current approaches
 - Most rely on source code—inappropriate for Windows
 - HookScout—state of the art for 32-bit Windows
 - No need for source
 - Extensive offline analysis
 - Won't work for 64-bit Windows (PatchGuard, relocation table)
- HookLocator for Windows
 - First solution for cloud to monitor function pointers in kernel heap
 - Virtual machine introspection provides resilience against in-VM OS compromise
 - No in-VM presence at all
 - No offline analysis – use information from target VM in real time to track modifications to pointers

HookLocator



Extraction Module

- Extract pointers from reliable sources in target system's memory
 - Kernel code
 - Kernel data structures
 - Import Address Table (IAT)
 - Export table
 - System Service Descriptor Table (SSDT)
 - Interrupt Descriptor Table (IDT)
- Data structures contain specific fields for pointers
- Kernel code is different – no specific locations

Extraction from Kernel Code

- Two approaches
 - Cross-comparison approach
 - Pattern matching approach
- Cross-comparison approach
 - Utilize relocation property of kernel code
 - 32-bit Windows Kernel contains absolute addresses, handled by relocation table
 - Relocation table is discarded after kernel is loaded into memory
 - HookScout uses relocation table
 - **If Kernel is loaded into different memory locations, the absolute addresses are different**

Extraction Module – Two Snapshots of Kernel Loaded at Different Locations

00000000	<u>bc 44 2b 86</u>	<u>76 44 2b 86</u>	<u>f6 44 2b 86</u>	<u>44 45 2b 86</u>	.D+.vD+..D+.DE+.
00000010	<u>7c 45 2b 86</u>	<u>f2 43 2b 86</u>	<u>64 44 2b 86</u>	<u>2c 44 2b 86</u>	E+..C+.dD+.,D+.
00000020	<u>ba 49 2b 86</u>	<u>24 b0 2b 86</u>	<u>bc 43 2b 86</u>	<u>62 48 2b 86</u>	.I+.\$.+..C+.bH+.
	.	.	.	.	
0006c710	30 01 83 f8	20 77 34 83	c1 04 3b 4d	fc 72 eb 89	0... w4...;M.r..
0006c720	55 10 ff 15	<u>64 20 60 82</u>	64 8b 0d 20	00 00 00 88	U...d`.d... ..
	.	.	.	.	
0011b8e0	4e 8b 30 5b	46 d6 59 41	b8 71 17 c7	11 e7 34 0c	N.0[F.YA.q....4.
0011b8f0	02 00 00 00	6e 74 6b 72	70 61 6d 70	2e 70 64 62	ntkrpamp.pdb



00000000	<u>bc 24 28 86</u>	<u>76 24 28 86</u>	<u>f6 24 28 86</u>	<u>44 25 28 86</u>	.\$(.v\$(..\$(.D%(.
00000010	<u>7c 25 28 86</u>	<u>f2 23 28 86</u>	<u>64 24 28 86</u>	<u>2c 24 28 86</u>	%(..#(.d\$(.,\$(.
00000020	<u>ba 29 28 86</u>	<u>24 90 28 86</u>	<u>bc 23 28 86</u>	<u>62 28 28 86</u>	.)(\$.(..#(.b((.
	.	.	.	.	
0006c710	30 01 83 f8	20 77 34 83	c1 04 3b 4d	fc 72 eb 89	0... w4...;M.r..
0006c720	55 10 ff 15	<u>64 00 65 82</u>	64 8b 0d 20	00 00 00 88	U...d.e.d...
	.	.	.	.	
0011b8e0	4e 8b 30 5b	46 d6 59 41	b8 71 17 c7	11 e7 34 0c	N.0[F.YA.q....4.
0011b8f0	02 00 00 00	6e 74 6b 72	70 61 6d 70	2e 70 64 62	ntkrpamp.pdb

Extraction: Cross-Comparison Approach

00000000	<u>bc 24 28 86</u>	<u>76 24 28 86</u>	<u>f6 24 28 86</u>	<u>44 25 28 86</u>	. \$(.v\$(..\$(.D%(.
00000010	<u>7c 25 28 86</u>	<u>f2 23 28 86</u>	<u>64 24 28 86</u>	<u>2c 24 28 86</u>	%(..#(.d\$(.,\$(.
00000020	<u>ba 29 28 86</u>	<u>24 90 28 86</u>	<u>bc 23 28 86</u>	<u>62 28 28 86</u>	.) (\$.(..#(.b((.
0006c710	30 01 83 f8	20 77 34 83	c1 04 3b 4d	fc 72 eb 89	0... w4...;M.r..
0006c720	55 10 ff 15	<u>64 00 65 82</u>	64 8b 0d 20	00 00 00 88	U...d.e.d..
0011b8e0	4e 8b 30 5b	46 d6 59 41	b8 71 17 c7	11 e7 34 0c	N.0[F.YA.q....4.
0011b8f0	02 00 00 00	6e 74 6b 72	70 61 6d 70	2e 70 64 62	ntkrpamp.pdb

=

Different Absolute
Addresses

00000000	<u>bc 44 2b 86</u>	<u>76 44 2b 86</u>	<u>f6 44 2b 86</u>	<u>44 45 2b 86</u>	.D+.vD+..D+.DE+.
00000010	<u>7c 45 2b 86</u>	<u>f2 43 2b 86</u>	<u>64 44 2b 86</u>	<u>2c 44 2b 86</u>	E+..C+.dD+.,D+.
00000020	<u>ba 49 2b 86</u>	<u>24 b0 2b 86</u>	<u>bc 43 2b 86</u>	<u>62 48 2b 86</u>	.I+.\$.+..C+.bH+.
0006c710	30 01 83 f8	20 77 34 83	c1 04 3b 4d	fc 72 eb 89	0... w4...;M.r..
0006c720	55 10 ff 15	<u>64 20 60 82</u>	64 8b 0d 20	00 00 00 88	U...d `d.. ...
0011b8e0	4e 8b 30 5b	46 d6 59 41	b8 71 17 c7	11 e7 34 0c	N.0[F.YA.q....4.
0011b8f0	02 00 00 00	6e 74 6b 72	70 61 6d 70	2e 70 64 62	ntkrpamp.pdb



00000000	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	.D+.vD+..D+.DE+.
00000010	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	E+..C+.dD+.,D+.
00000020	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	<u>T A G 0</u>	T+ \$ + C+ bH+
0006c710	30 01 83 f8	20 77 34 83	c1 04 3b 4d	fc 72 eb 89	0... w4...;M.r..
0006c720	55 10 ff 15	<u>T A G 0</u>	64 8b 0d 20	00 00 00 88	U...d `d.. ...
0011b8e0	4e 8b 30 5b	46 d6 59 41	b8 71 17 c7	11 e7 34 0c	N.0[F.YA.q....4.
0011b8f0	02 00 00 00	6e 74 6b 72	70 61 6d 70	2e 70 64 62	ntkrpamp.pdb

Address is replaced
with unique Tag

Extraction: Pattern Matching Approach

- Specifically for 64-bit Windows kernel
- Does not depend on the relocation property of the Kernel
- Analyzed collection of functions and found a pattern: 0xC390
 - After a return instruction, there are NOP instructions before next function starts
 - Search for return instructions, and then follow the NOP until other instructions are found – pointing to the start of a function

Extraction: Pattern Matching Approach (2)

```

ffffff800`02dd660b 4c8923
ffffff800`02dd660e 4883c428
ffffff800`02dd6612 415f
ffffff800`02dd6614 415c
ffffff800`02dd6616 5d
ffffff800`02dd6617 5b
ffffff800`02dd6618 c3
ffffff800`02dd6619 90
ffffff800`02dd661a 90
ffffff800`02dd661b 90
ffffff800`02dd661c 90
ffffff800`02dd661d 90
ffffff800`02dd661e 90
ffffff800`02dd661f 90

```

nt!IopSynchronousServiceTail:

```

ffffff800`02dd6620 48895c2408
ffffff800`02dd6625 4c89442418
ffffff800`02dd662a 55
ffffff800`02dd662b 56
ffffff800`02dd662c 57
ffffff800`02dd662d 4154
ffffff800`02dd662f 4155
ffffff800`02dd6631 4883ec40

```

Beginning of a
function

```

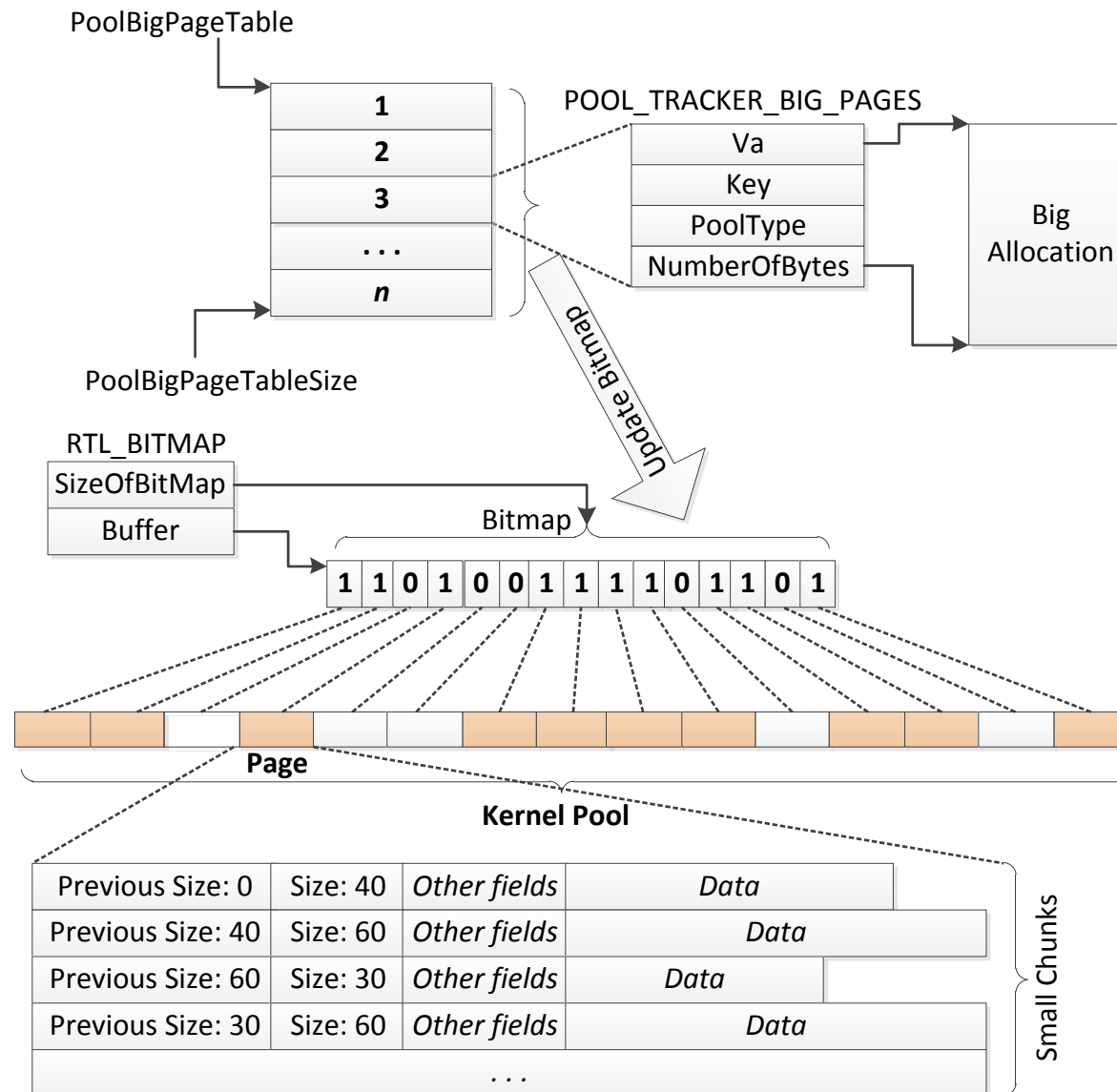
. . .
mov    qword ptr [rbx],r12
add    rsp,28h
pop     r15
pop     r12
pop     rbp
pop     rbx
ret
nop
nop
nop
nop
nop
nop
nop
mov     qword ptr [rsp+8],rbx
mov     qword ptr [rsp+18h],r8
push    rbp
push    rsi
push    rdi
push    r12
push    r13
sub     rsp,40h
. . .

```

Return instruction

NOPs

Search Module: Kernel Pool Allocations



Search: Small Chunks in Page

Use as a pattern to find small allocations

```
lkd> !pool fffffa80`127995d0
```

Pool page fffffa80127995d0 region is Nonpaged pool

Address	Size	Previous Size	State	Tag
fffffa8012799000	230	0	(All)	
fffffa8012799230	50	230	(Allocated)	PnpI
fffffa8012799280	40	50	(Allocated)	PnpI
fffffa80127992c0	50	40	(Allocated)	PnpI
fffffa8012799310	40	50	(Allocated)	PnpI
fffffa8012799350	40	40	(Allocated)	PnpI
fffffa8012799390	40	40	(Allocated)	PnpI
fffffa80127993d0	50	40	(Allocated)	PnpI
fffffa8012799420	280	50	(Allocated)	Dnod
fffffa80127996a0	230	280	(Allocated)	Devi
fffffa80127998d0	280	230	(Allocated)	Dnod
fffffa8012799b50	230	280	(Allocated)	Devi
fffffa8012799d80	280	230	(Allocated)	Dnod

Previous size of first allocation

Learning Module

- Identify candidate pointers as genuine
- Observed: 97% of function pointers in Microsoft Windows do not change over their entire life span
- Criteria: Function pointers that do not change throughout their lifetime are genuine pointers
- Rationale: Attractive targets for attack are the pointers that:
 - Have a long life span
 - Do not change throughout their life span
 - These are useful for **consistently** changing control flow

Learning Module (2)

- When Learning Module decides candidate pointer is genuine:
 - Stores contextual information in genuine pointer list
 - All the values in the header of the allocation:
 - PoolTag
 - PoolType,
 - EPROCESS (ProcessBilled field) etc.
 - Offset of the pointer from the start of the allocation
 - Name of the module
 - Offset of the pointer from the start of the module

Pool Monitor

- Observe the validity of “genuine” pointers
- Ideally, the pointer being monitored should not change
- If it changes, should be come NULL...
- ...or address of some other function within kernel space

Evaluation

- Guest VM running Windows 7
- Hypervisor: Xen
- Installed some common user applications:
 - Skype, Google Chrome, MS Office 2010, Acrobat Reader etc.
- ...to understand the effects of user processes on kernel heap data

of Function Pointers in .text Section

	Windows 7				
	32-bit		64-bit		
Module Name	H	C	H	C	P
ntoskrnl.exe	5,388	5,390	401	200	2,960
hal.dll	537	537	0	0	537
ntfs.sys	147	147	0	0	416
i8042prt.sys	68	68	0	0	90
http.sys	212	212	0	0	518
disk.sys	11	11	0	0	65
volmgr.sys	18	18	0	0	35
kdcom.dll	21	21	0	0	20

- HookScout's IDA plugin (H)
- Cross-comparison approach (C)
- Pattern matching approach (P)

Function Pointers from Different Sources

Including all
modules, etc.



Function Pointer Source Name	Total Number of Function Pointers
Kernel Code	28,518
IDT	271
SSDT	401
IAT	12,142
Export Table	4,887
<i>TOTAL</i>	<i>46,219</i>
<i>Unique Pointers</i>	<i>30,875</i>

- 33.2% of function pointers are duplicates
 - Same functions imported by multiple modules
 - Common function pointers

Correlation: Size of Allocation and # of Function Pointers

		Total size of allocations (MB)	Total number of Pointers	Number of pointers Per MB
Big allocations	Non paged pool	34.09	0	0
	Paged pool	79.13	1,665	12
Small Chunks	Non paged pool	4.25	3,201	753
	Paged pool	172.62	2,297	13

- Small chunks (non-paged pool) is the most attractive target
 - High density of function pointers
 - Page swapping is not an issue

Effects of User Processes

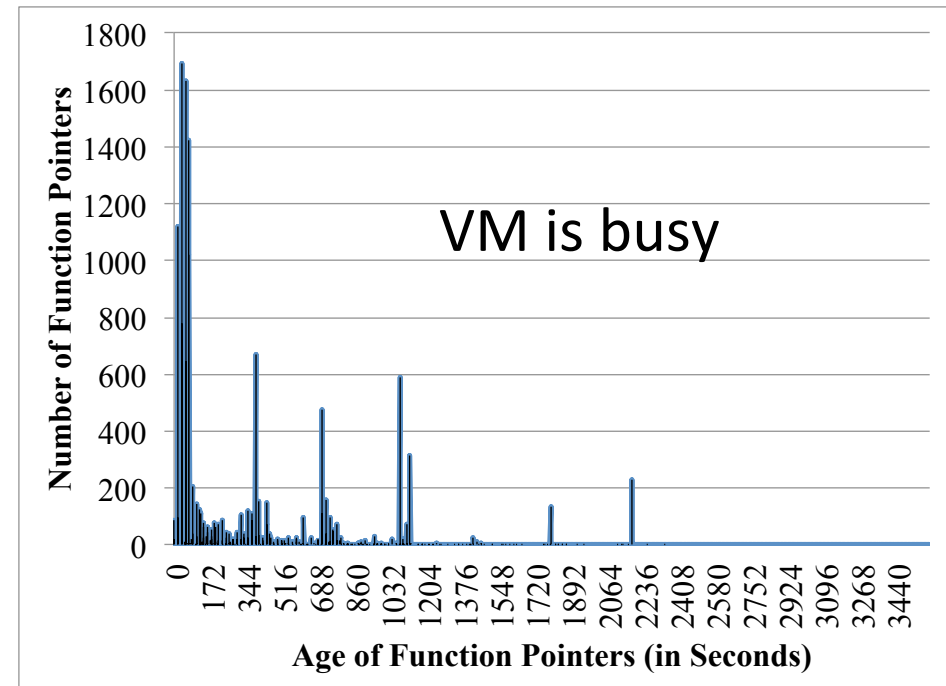
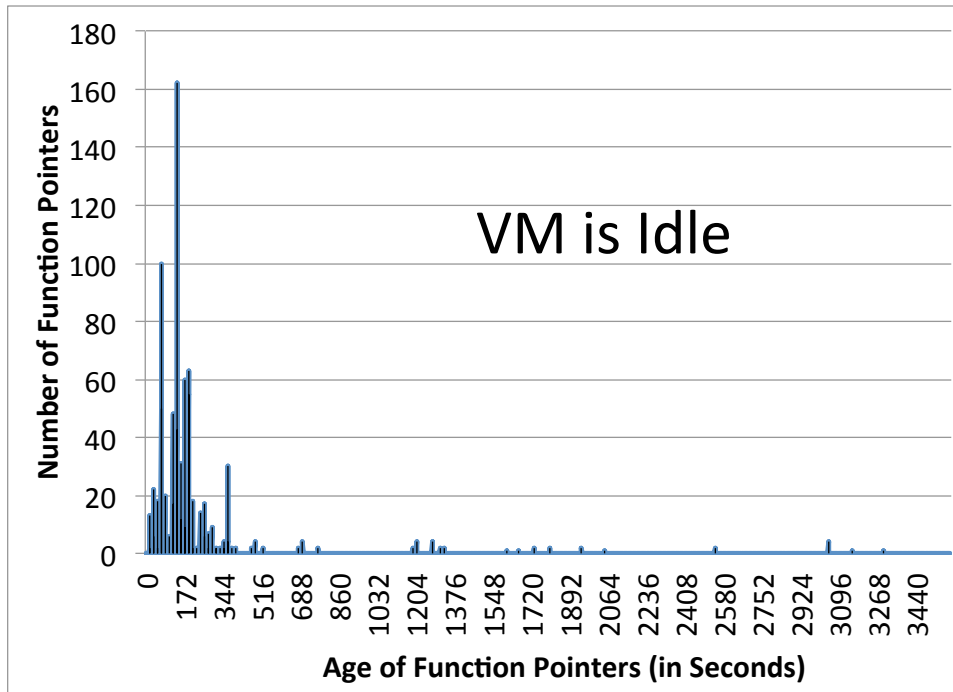
Small chunks, non-paged pool

	Total size of Allocations (in Kilo Bytes)			Total Number of Pointers		
	Before process initiated	After process initiated	Percent Increase (%)	Before process initiated	After process initiated	Percent Increase (%)
Skype	4,717.00	4,770.83	1.14	3,456	3,499	1.25
MS Word	4,751.25	4,793.33	0.89	3,491	3,512	0.61
MS PowerPoint	4,769.25	4,789.33	0.42	3,503	3,515	0.34
Acrobat Reader	4,767.25	4,840.67	1.54	3,507	3,535	0.79
Windows Media Player	4,820.75	4,928.00	2.22	3,481	3,556	2.16
Chrome	4,789.75	4,863.33	1.54	3,472	3,533	1.76

1–2% increase in total allocation size

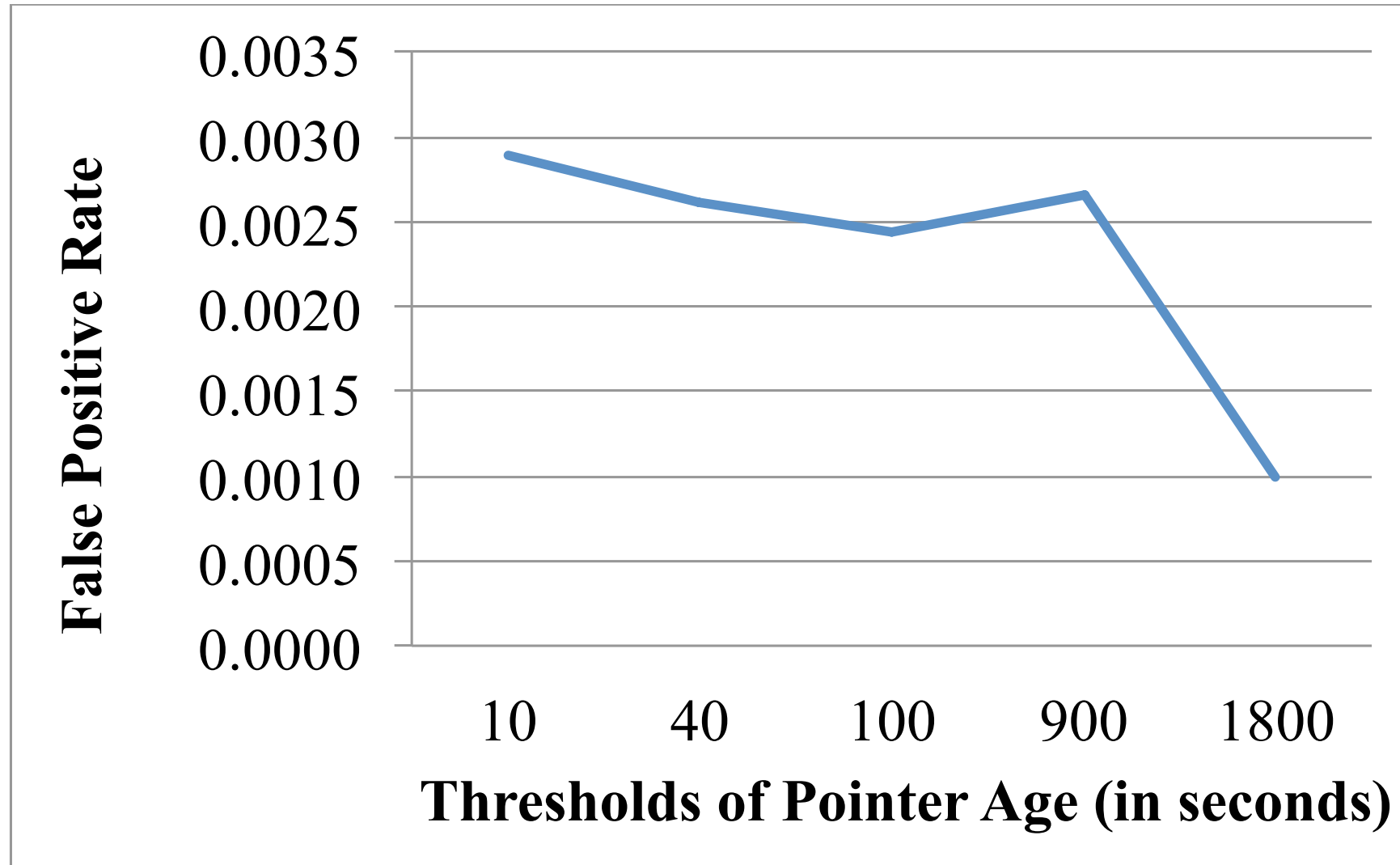
1–2% Increase in function pointers

Pointer Lifespan



- PCMark 7 – performance-benchmarking tool
- “Idle”: 8% of pointers gone within 6 minutes
- “Busy”: 69% of pointers gone within 20 minutes

False Positive Rate: Different Thresholds



Pool Monitor: Effectiveness

- Created a tool that synthetically mutates function pointers in the kernel pools at random
- HookLocator raises alerts if pointer value is changed to some unexpected value
 - Not NULL
 - Not within address range of kernel code
- 100% detection

Performance Evaluation

- Experimental settings
 - Intel Core 2 Quad CPU (4 × 2.83GHz) with 4GB RAM
 - Xen (4.1.2),
 - Guest VM: 64-bit Windows 7
 - VM has 1 GB RAM & 1 core
 - Privileged VM: Fedora 16 (3.1.0-7.fc16.x86_64 kernel)
- Evaluation on different types of memory allocations
 - big allocations
 - small chunks
 - paged
 - non-paged
- Using two extreme test cases when
 - VM is idle
 - VM is busy

Performance Evaluation

VM is Idle

Overhead is in **privileged VM**

	Memory Overhead		Number of Executions	
	MB	%age	per minute	per second
Big allocations	60	2.38	31	0.52
Small chunks (Paged Pool)	44	1.74	4	0.07
Small chunks (Non-paged pool)	24	0.94	115	1.91

VM is Busy

Overhead is in **privileged VM**

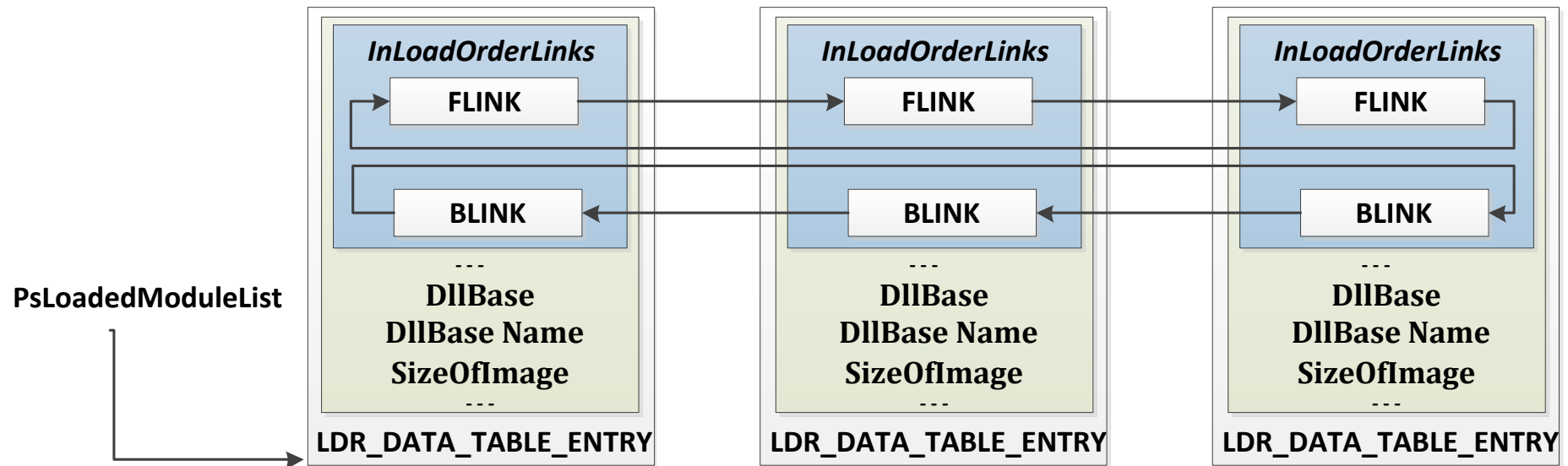
	Memory Overhead		Number of Executions	
	MB	%age	per minute	per second
Big allocations	65	2.58	31	0.52
Small chunks (Paged Pool)	47	1.87	4	0.07
Small chunks (Non-paged pool)	24	0.96	101	1.68

Conclusions

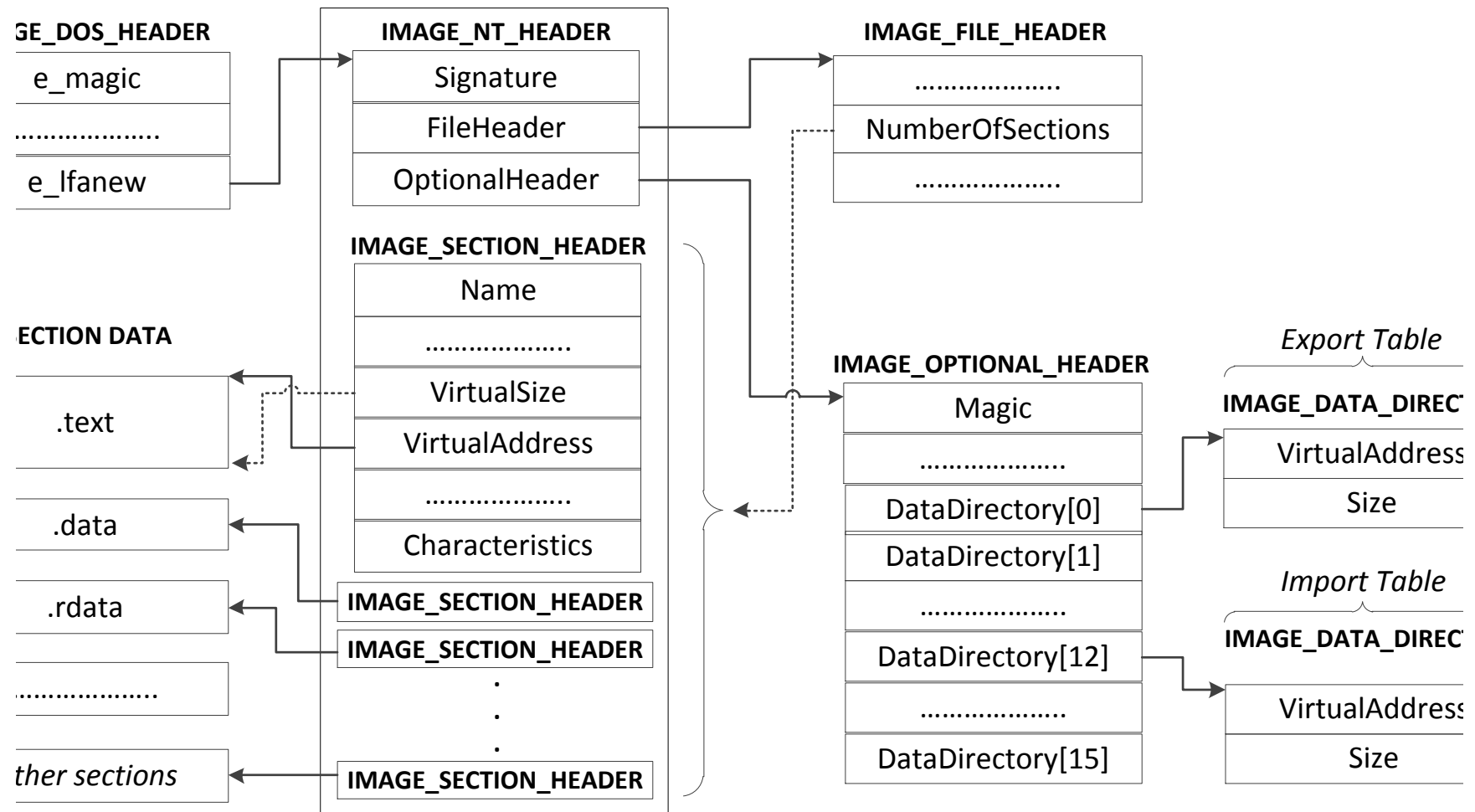
- State-of the art solution cannot be seamlessly ported to 64-bit Windows
- Our solution works for both 32-bit and 64-bit Windows versions
- We identify the heap region that provides the most favorable attack surface – non-paged small chunks
- HookLocator monitors this heap region with low overhead and in close to “real time” fashion
- Runs at a higher privilege level than the potentially compromised VM
- No in-VM components, so difficult to subvert

Questions?

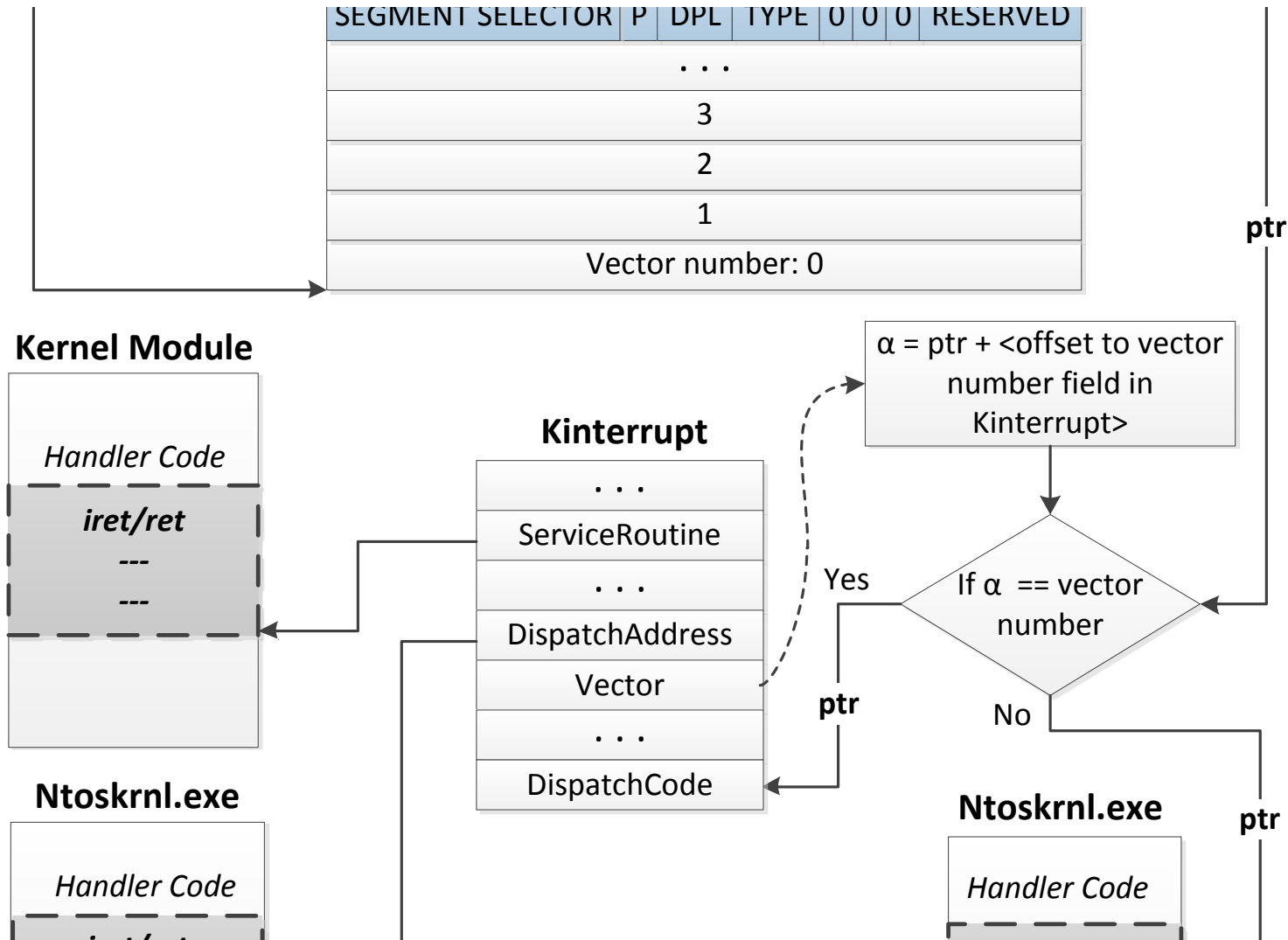
Extraction Module – module traversing



Extraction Module – data structures



Extraction Module – data structures



Extraction Module – data structures

- System Service Descriptor Table
 - KeServiceDescriptorTable contains pointer to SSDT
 - In 32-bit Windows XP
 - KeServiceDescriptorTable is present in export table
 - KTHREAD data structure also contains a ServiceTable field, which is initialized with KeServiceDescriptorTable
 - In Windows 7, export table and KTHREAD do not contain KeServiceDescriptorTable
 - Extract the variable value from KeAddSystemServiceTable function