



## **A Hypervisor IPS based on Hardware Assisted Virtualization Technology**

**Fourteenforty Research Institute, Inc.**  
<http://www.fourteenforty.jp>

Senior Research Engineer  
Junichi Murakami



## Presentation Outline

1. Review of subversive techniques in kernel space
2. Review of Virtualization Technology
3. Viton, Hypervisor IPS
4. Conclusion



- 1. Review of subversive techniques in kernel space

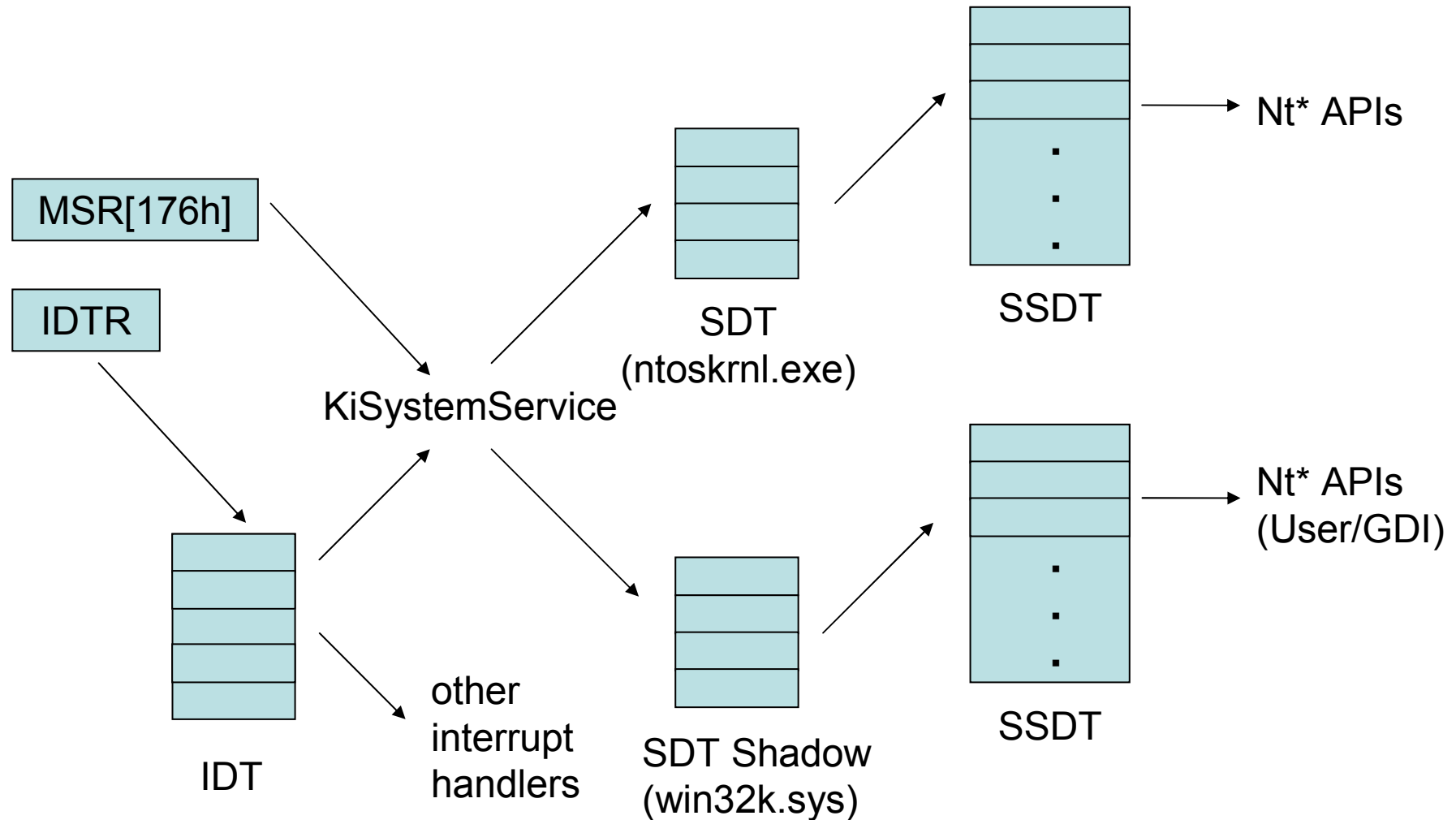


## Remember Joanna's classification

- Joanna Rutkowska proposed stealth malware taxonomy in November, 2006.  
<http://invisiblethings.org/papers/malware-taxonomy.pdf>
- Type 0
  - standalone malware, which never changes any system resources
- Type I
  - changes the persistent system resources
- Type II
  - changes the non-persistent system resources
- Type III
  - malware runs outside the system



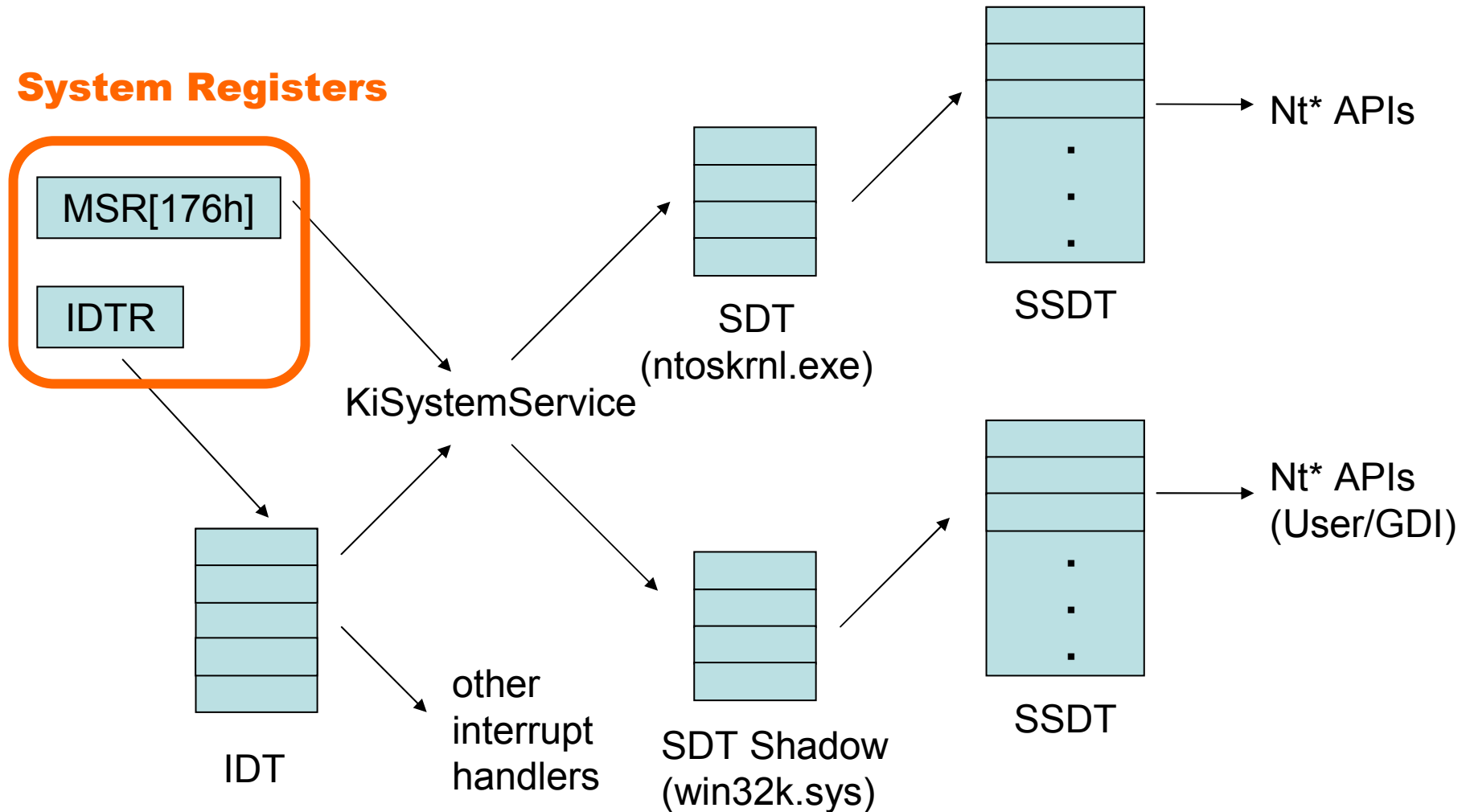
# Type I: Overview of Hooking Points





# Type I: Overview of Hooking Points

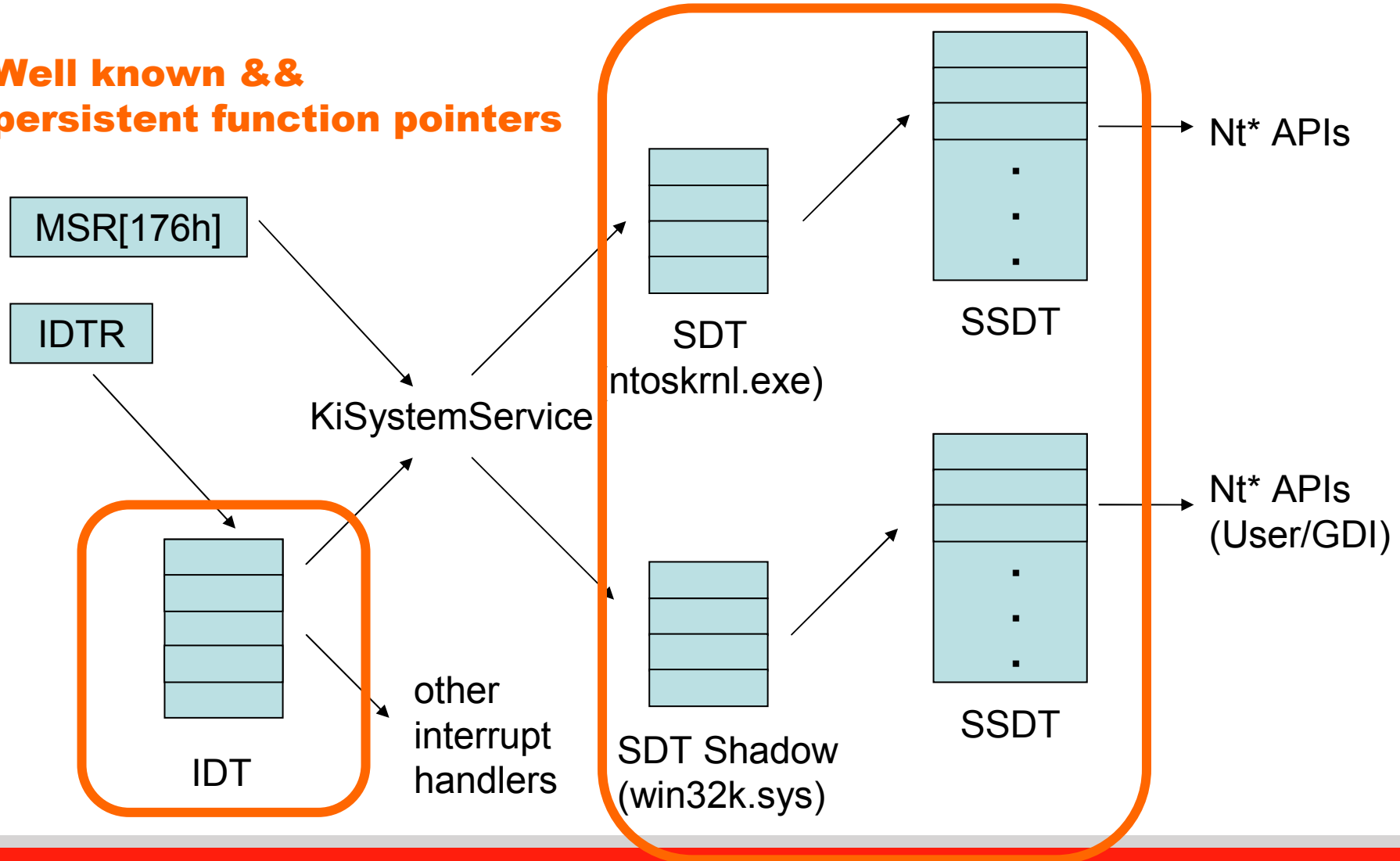
## System Registers





# Type I: Overview of Hooking Points

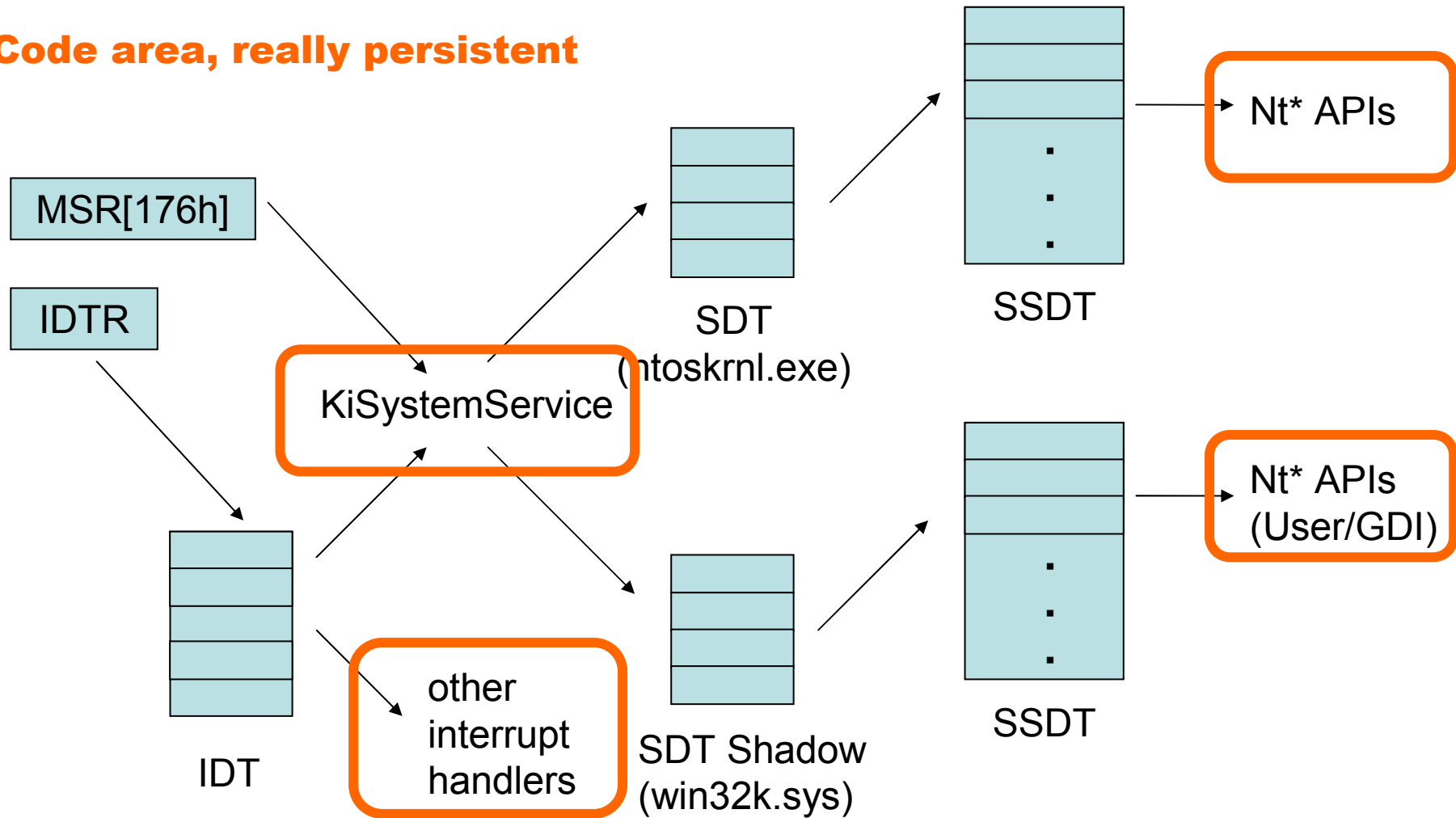
**Well known &&  
persistent function pointers**





# Type I: Overview of Hooking Points

Code area, really persistent







## Type I

- It is easy to **detect**
- PatchGuard in Vista(x64) is a countermeasure for this type
- Many **rootkit detectors** have been released for this type



## Type II

- Malware changes the non-persistent system resources
- Hooking point might be modified by the regular execution path
- DKOM(Direct Kernel Object Manipulation)
  - by <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf>
- KOH(Kernel Object Hooking)
  - by Greg Hogle in Jan, 2006  
<http://www.rootkit.com/newsread.php?newsid=501>



## DKOM(Direct Kernel Object Manipulation)

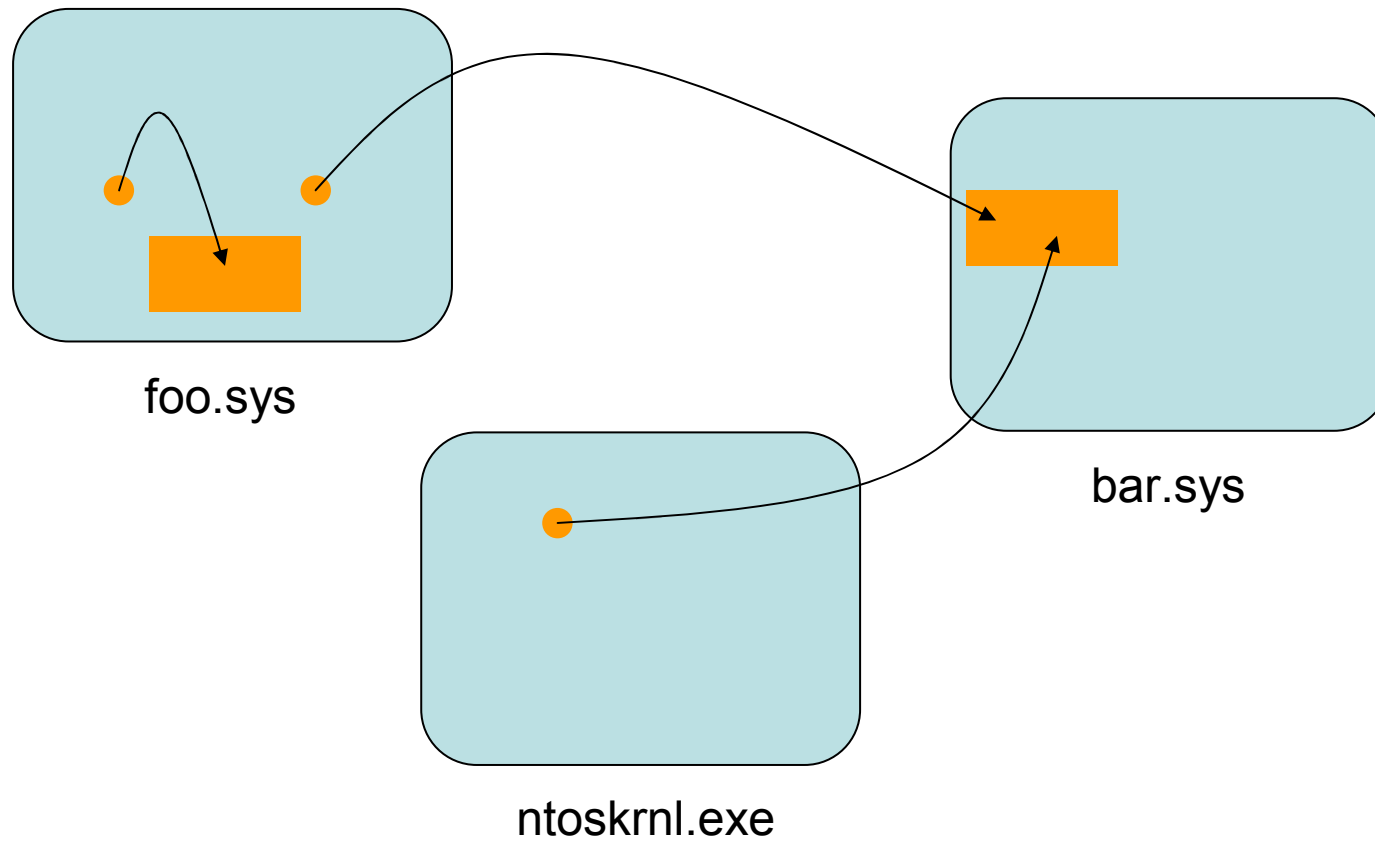
- Malware manipulates the process list, tokens and other kernel objects directly
- For example:
  - Unlink target process from process list
  - Add/remove priviledges to tokens
- DKOM's possibilities are limited
  - Whether information hiding can be done depends on the implementation of process that deals with the data



## KOH(Kernel Object Hooking)

- Remember the SDT, SSDT and other well known && persistent function pointers?
- Do you know how many such patching points are there in kernel space?
  - They might or might not be persistent
  - It depends on each kernel object
- Detector has to understand all function pointers
- `is_within_own_memory_range(PVOID Address)` is useful, but not enough

# is\_within\_own\_memory\_range(PVOID Addr)





## Type III

- No malware exists in the system(guest)
- Malware (ab)uses Virtualization Technology
- SMM Rootkit and Firmware Rootkit might also fall into this category (a problem of taxonomy that is not important for our cause)
  
- BluePill
  - Original BP was presented by Joanna Rutkowska in BH-US-2006.
  - (Current) New BP supports both Intel VT and AMD-v technologies, and is also capable of on the fly loading and unloading
  - BP doesn't modify any system resources on the guest
    - From a technical view, BP patches the guest's PTE to hide its loaded virtual memory from the guest
    - However this doesn't really help detecting it



## Type III (cont.)

- Vitriol
  - Presented by Dino Dai Zovi, Black Hat US 2006
  - VT-x rootkit, closed source
- VMM Rootkit Framework
  - Posted by Shawn Embleton, Aug, 2007  
<http://www.rootkit.com/newsread.php?newsid=758>
  - This is really good start point for learning for how to create VMM



## Case Study: Storm Worm

- The Storm Worm first appeared in Fall, 2006
- Some variants have rootkit functions to hide from AV products
- As of Jan 2008 we can see "Happy New Year 2008" variants
- When a user clicks onto the executable,



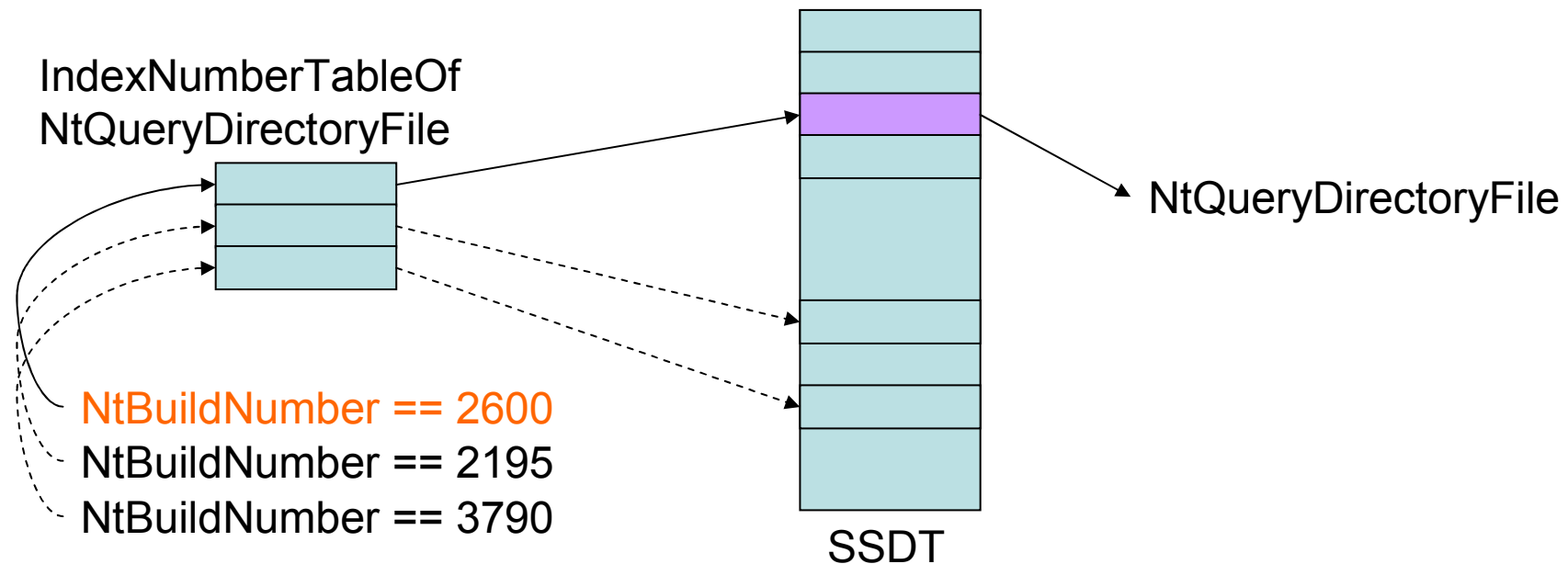


## Storm Worm

1. Executable drops the system driver (.sys), and loads it into the kernel using Service Control Manager (SCM)
  
2. Driver has two functions shown below
  - Rootkit functions
    - Hide files, registry entries and connections using SSDT and IRP hooking
  - Code Injection function
    - Inject malicious code (not DLL) into process context of services.exe and execute it
  
3. Injected code starts P2P communication

## Rootkit functions

- Storm Worm hooks three Native APIs
  - NtQueryDirectoryFile, NtEnumerateKey, NtEnumerateValueKey
- API Index of SSDT is different for each NtBuildNumber
- Storm Worm has index number tables for build 2195(2k), 2600(XP) and 3790(2k3)





## Rootkit functions (cont.)

- It hooks the IRP\_DEVICE\_CONTROL routine by patching the TCP DriverObject's IRP table ("¥¥Device¥¥Tcp")
- Hide connections from netstat

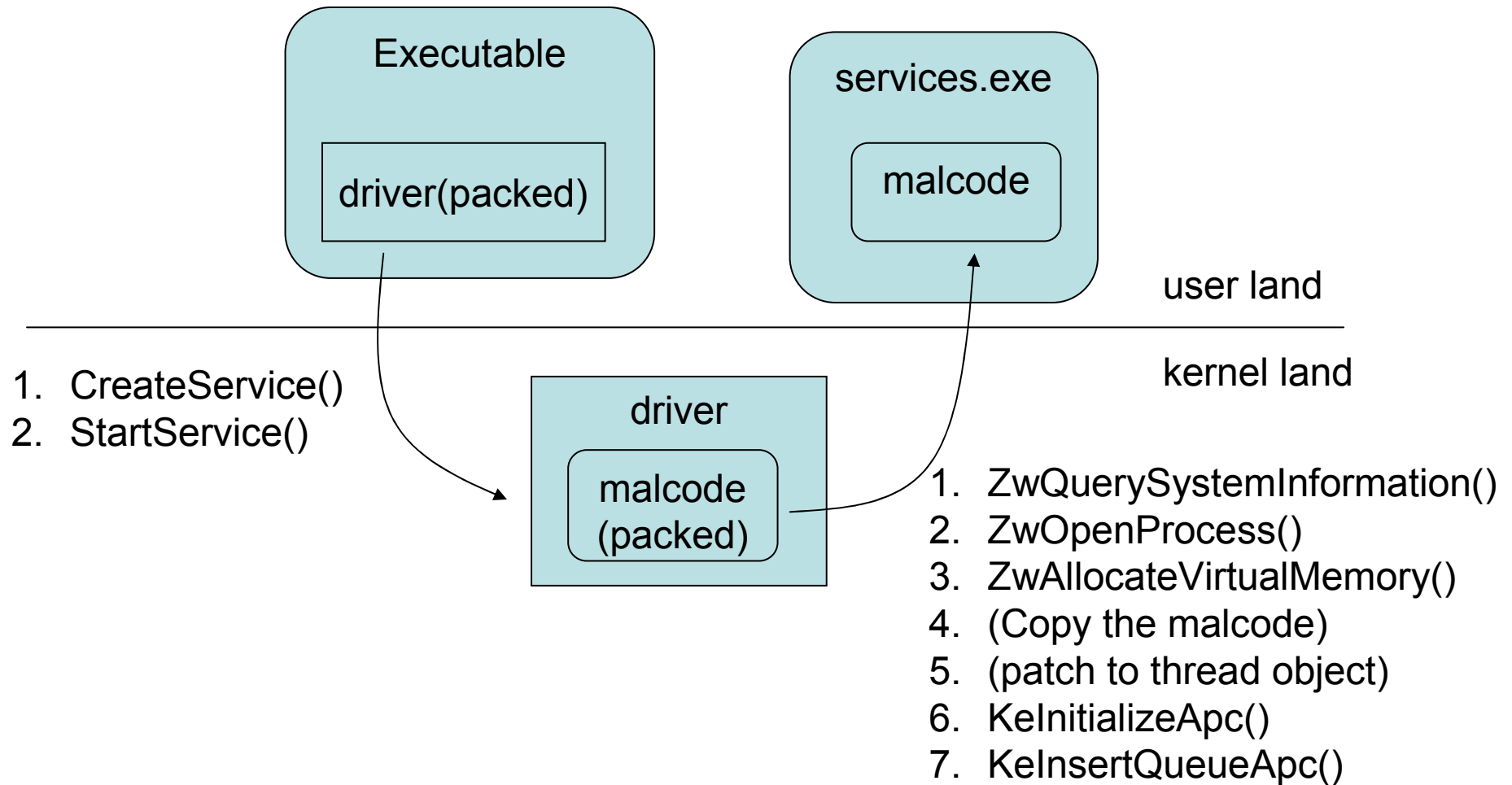
But is this KOH?

YES: It modifies the IRP Table contained within the DriverObject

NO: Many people know about the existence of IRP tables



# Code injection function





## 2. Review of subversive techniques in kernel space



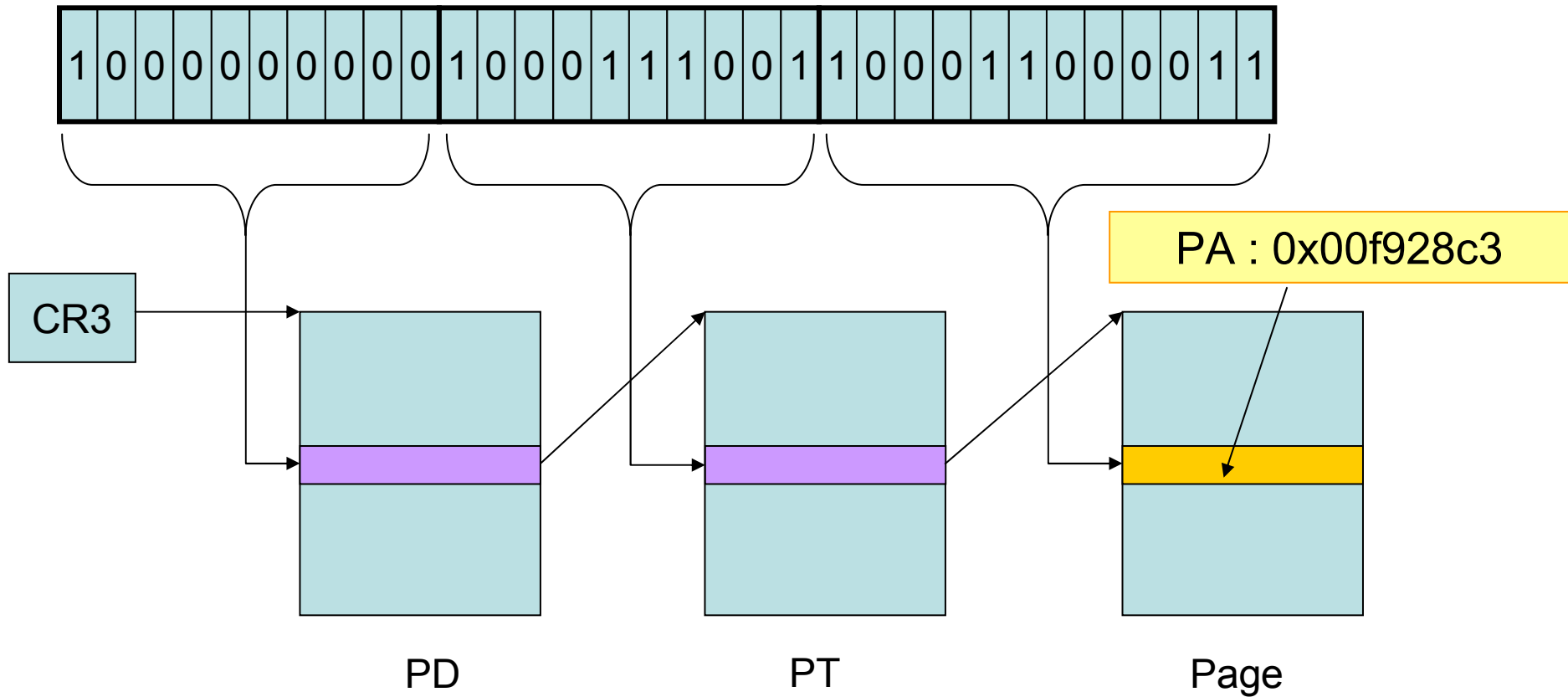
## What we have to consider “Virtualization”

- CPU Virtualization
  - Some registers should be reserved for VMM and each VM.  
GDTR, LDTR, IDTR, CR0–4, DR0–7, MSR, Segment Register, etc
  - Exceptions
- Memory Virtualization
  - should separate VMM memory space and each VM’s memory space
- Device Virtualization
  - Interrupt, I/O instructions, MMIO, DMA access



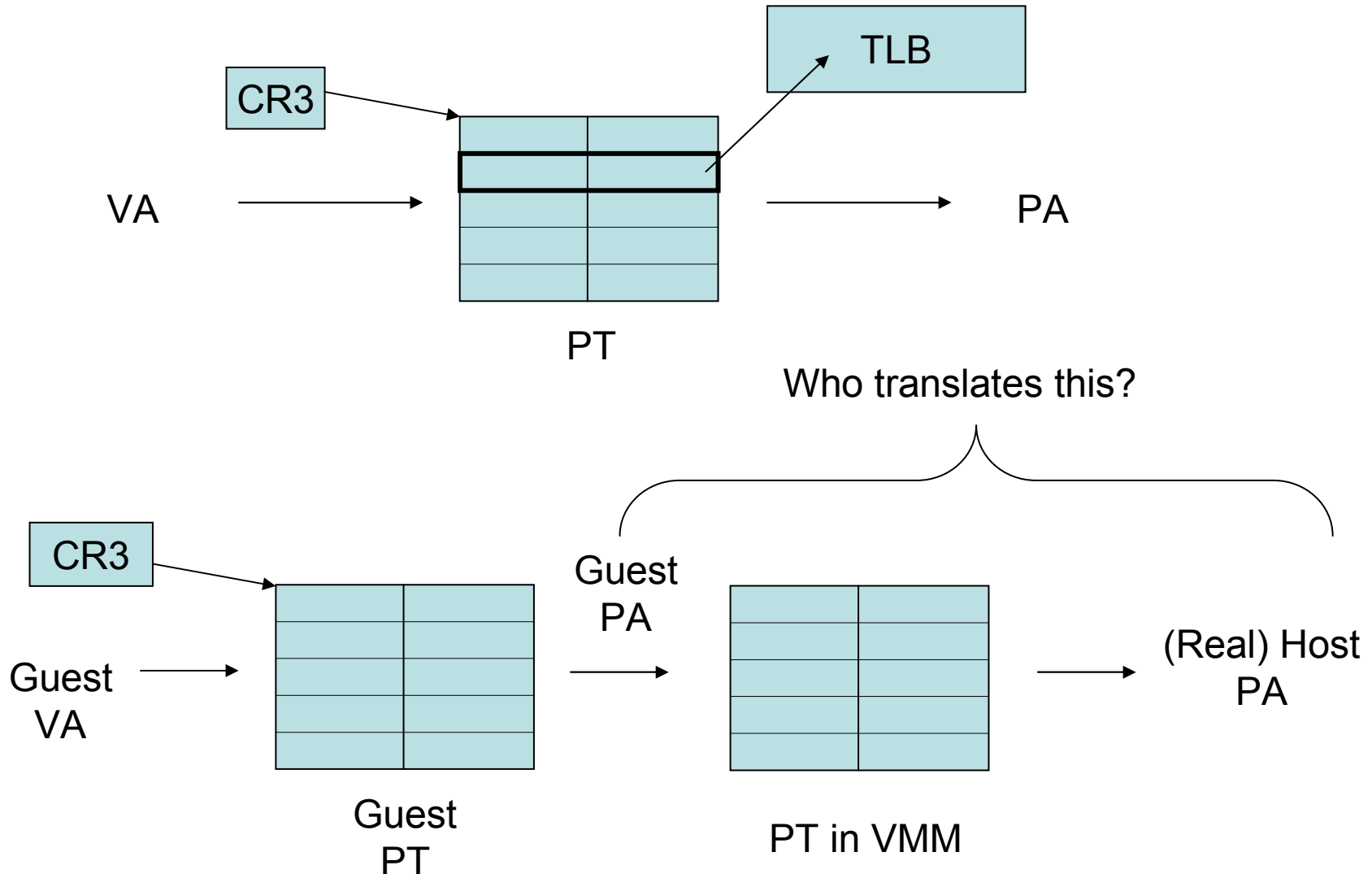
# Virtual Address to Physical Address

VA : 0x802398c3





To simplify...







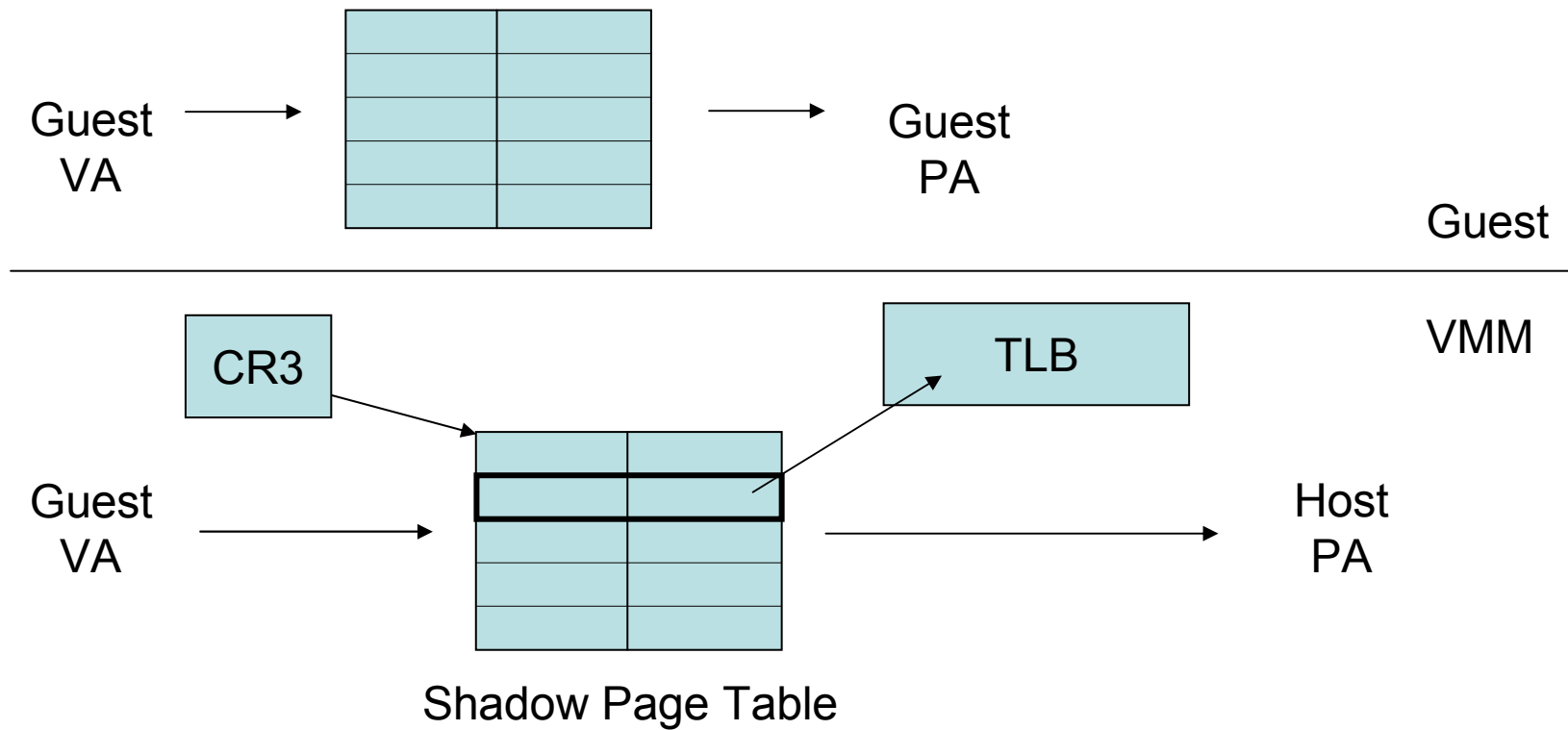
## Memory virtualization

- If the processor supports EPT (Extended Page Table), this 2-stages translation is automatically done by the MMU
  - EPT is not implemented yet
- VMM should implement this translation as software using Shadow Paging



# Shadow Paging

- VMM updates SPT on #PF in the guest
  - and also emulates TLB flush caused by MOV to CR3 and INVLPG



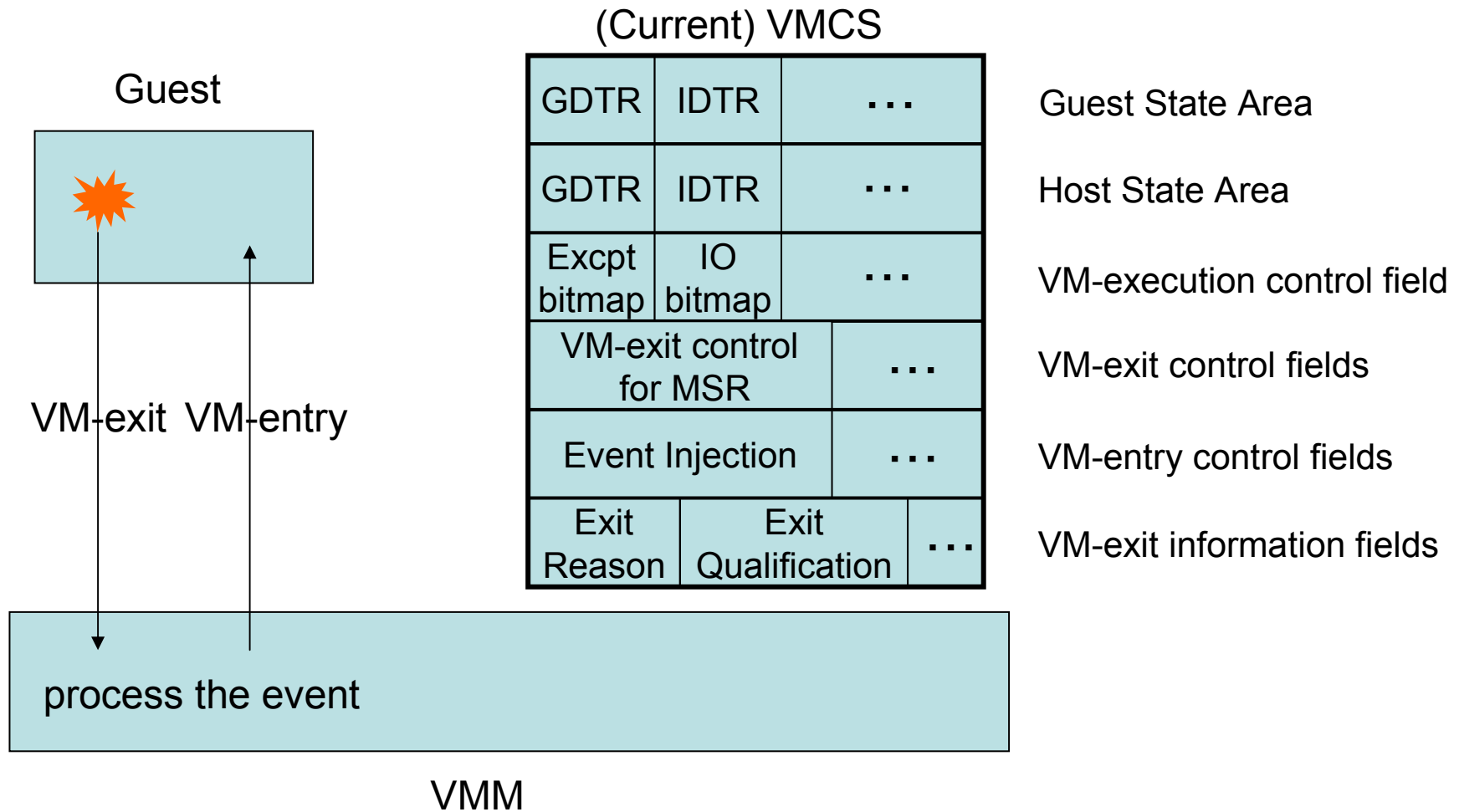


## Intel VT

- Intel VT is the Intel VT-\* family's generic name
  - VT-x, virtualization for x86/64
  - VT-d, virtualization for device (Directed I/O)
  - VT-i, virtualization for Itanium
- Key factors
  - VMX mode
    - VMX root-operations(ring0-3)
    - VMX non-root-operations(ring0-3)
  - VMCS (Virtual Machine Control Structure)
  - VMX Instructions set
    - VMXON, VMXOFF, VMLAUNCH, VMRESUME, VMCALL, VMWRITE, VMREAD, VMCLEAR, VMPTRLD, VMPTRST



# How Intel VT works:





```
enum EXIT_REASON {
```

- Specific instructions
  - CPUID, INVD, INVLPG, RDTSC, RDPMC, HLT, etc.
  - All VMX Instructions
- I/O Instructions
  - IN, OUT, etc.
- Exceptions
- Access to CR0–CR4, DR0–DR7, MSR
- etc.

```
};
```



## Steps to launch the VMM and VM

- Confirm that the processor supports VMX operations
  - CPUID
- Confirm that VMX operations are not disabled in the BIOS
  - MSR\_IA32\_FEATURE\_CONTROL
- Set the CR4.VMXE bit
- Allocate and Initialize VMXON region
  - Write lower 32 bits value of VMX\_BASIC\_MSR to VMXON region
- Execute VMXON
  - CR0.PE, CR0.PG, and CR4.VME must be set.



## Steps to launch the VM and VMM (cont.)

- Allocate VMCS regions
- Execute VMPTRLD to set Current VMCS
- Initialize Current VMCS using VMREAD and VMWRITE
  - VMCS contains the EP of VMM, and Guest IP after VMLAUNCH
- Execute VMLAUNCH
  - Continue to execute the guest from IP is contained in VMCS
- When VM-exit occurred, IP and other registers are switched to VMM ones.



### 3. Viton, Hypervisor IPS





## Viton

- IPS, which runs outside the guest
- Just a PoC, tested on Windows XP SP2 only
- Force immutability to persistent system resources
- Observe control/system registers modification, and VMX instructions are raised in the guest
- Offer the extensibility for monitoring the guest activity
- It is based on Bitvisor

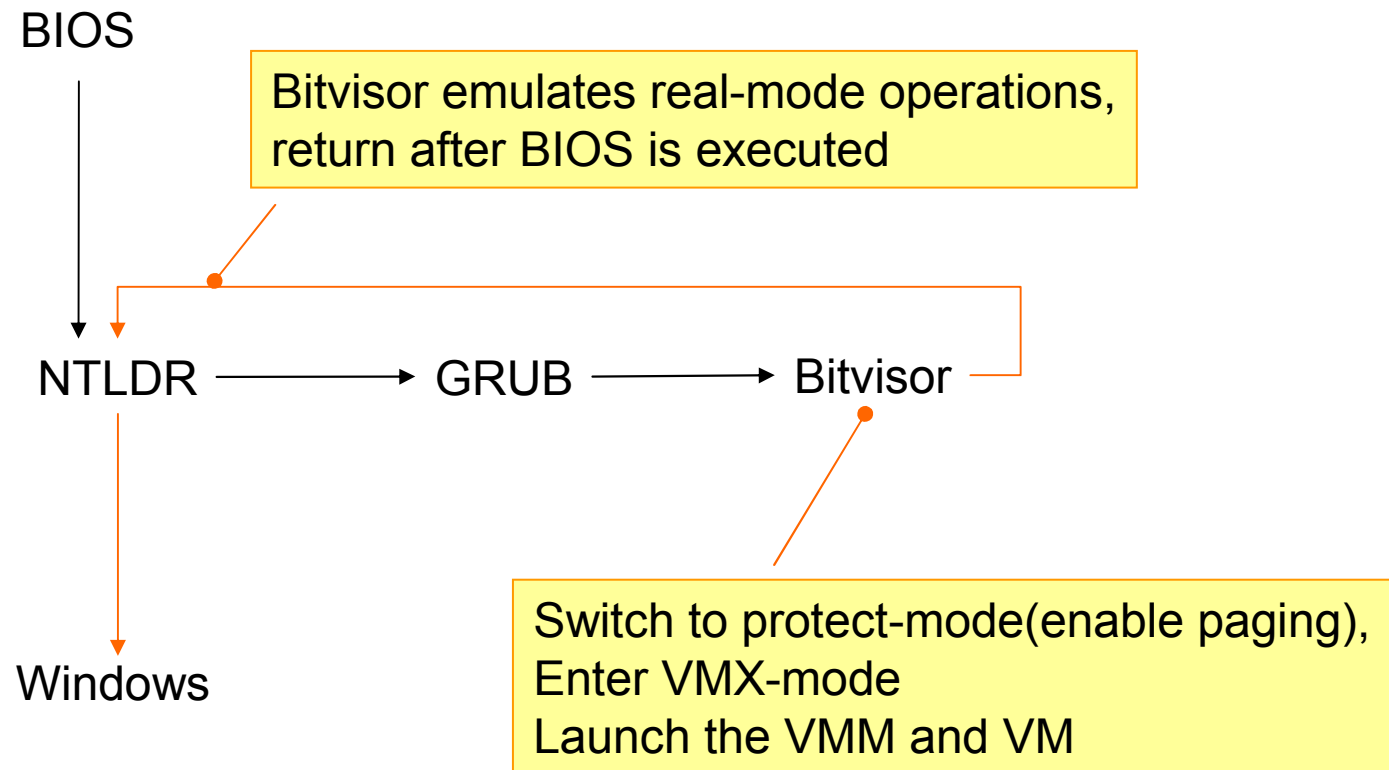


## Bitvisor – <http://www.securevm.org>

- The Bitvisor VMM software is developed by the Secure VM project centered around Tsukuba Univ. in Japan
  
- Features:
  - Open source, BSD License
  - Semi-path through model
  - Type I VMM (Hypervisor model, like Xen)
  - Full scratched, pure domestic production
  - Support for 32/64 bits architecture in VMM
  - Support for Multi-core/processor in VMM and Guest
  - Can run Windows XP/Vista as Guests without modification
  - Support for PAE in the Guest
  - Support for Real-mode emulation



## How Bitvisor works: Launch process





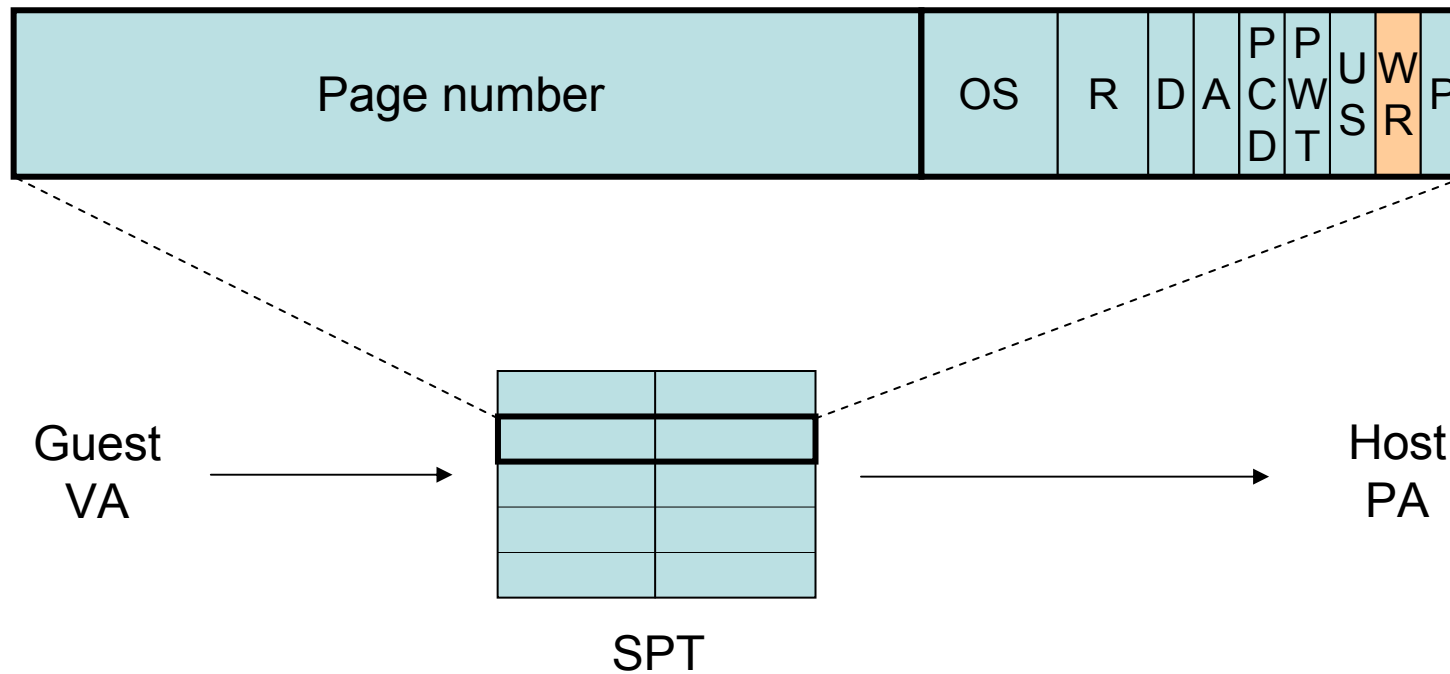
## What Viton protects/detects:

- Instructions
  - Detect and block all VMX Instructions
- Registers
  - Watchdog for IDTR
  - Locking the MSR[SYSTEMR\_EIP]
  - Locking the CR0.WP Bit
- Memory
  - Protect from modification
    - All code sections (R-X) in ntoskrnl.exe
    - IDT
    - SDT
    - SDT.ST (SSDT)



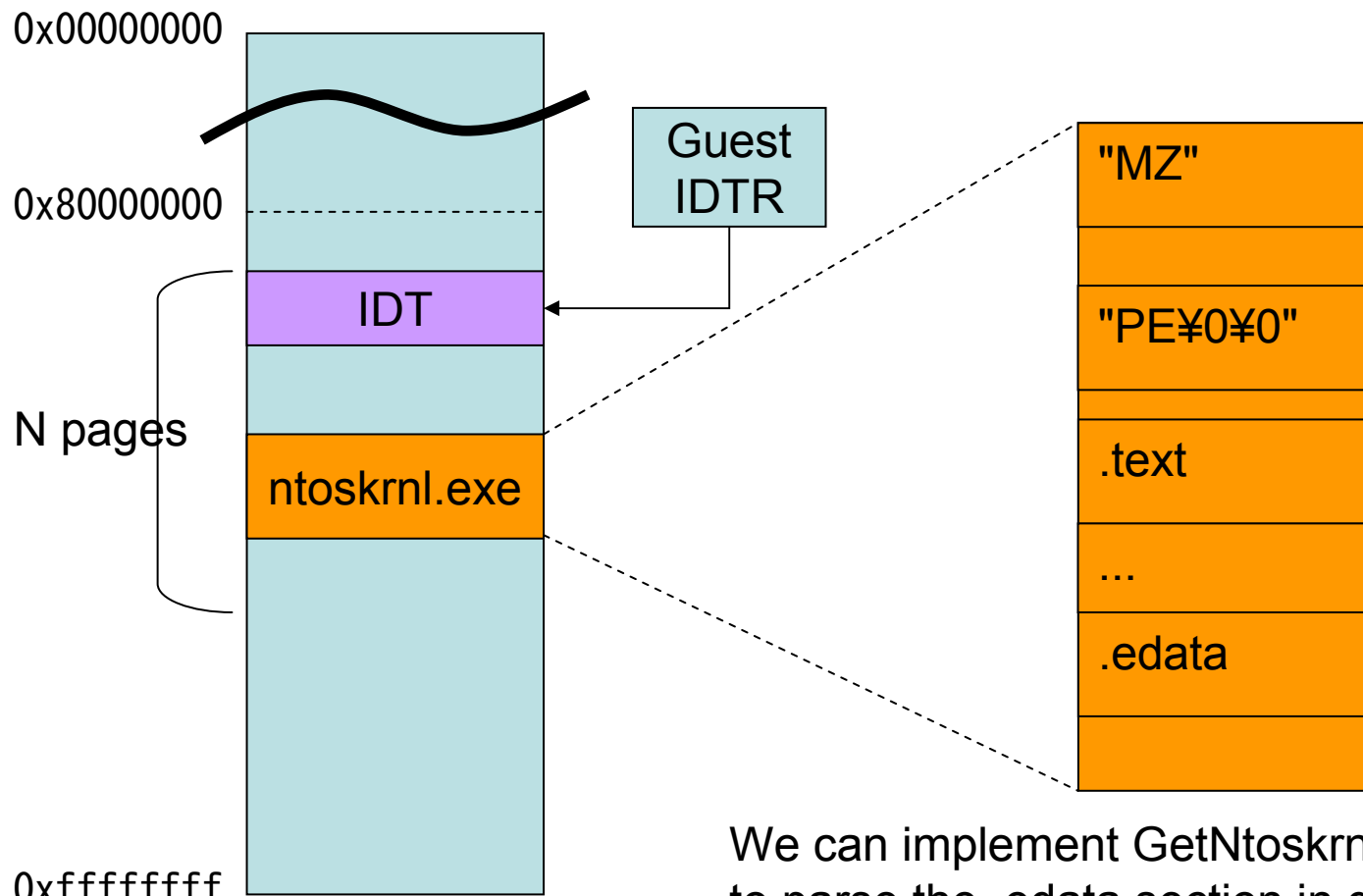
# How to protect the guest memory modification

- Viton clears the WR bit in a SPT entry
  - If CR0.WP is set, even the kernel cannot modify the page





# How to recognize the guest memory layout



We can implement GetNtoskrnlSymbolAddr() to parse the .edata section in guest VA space.

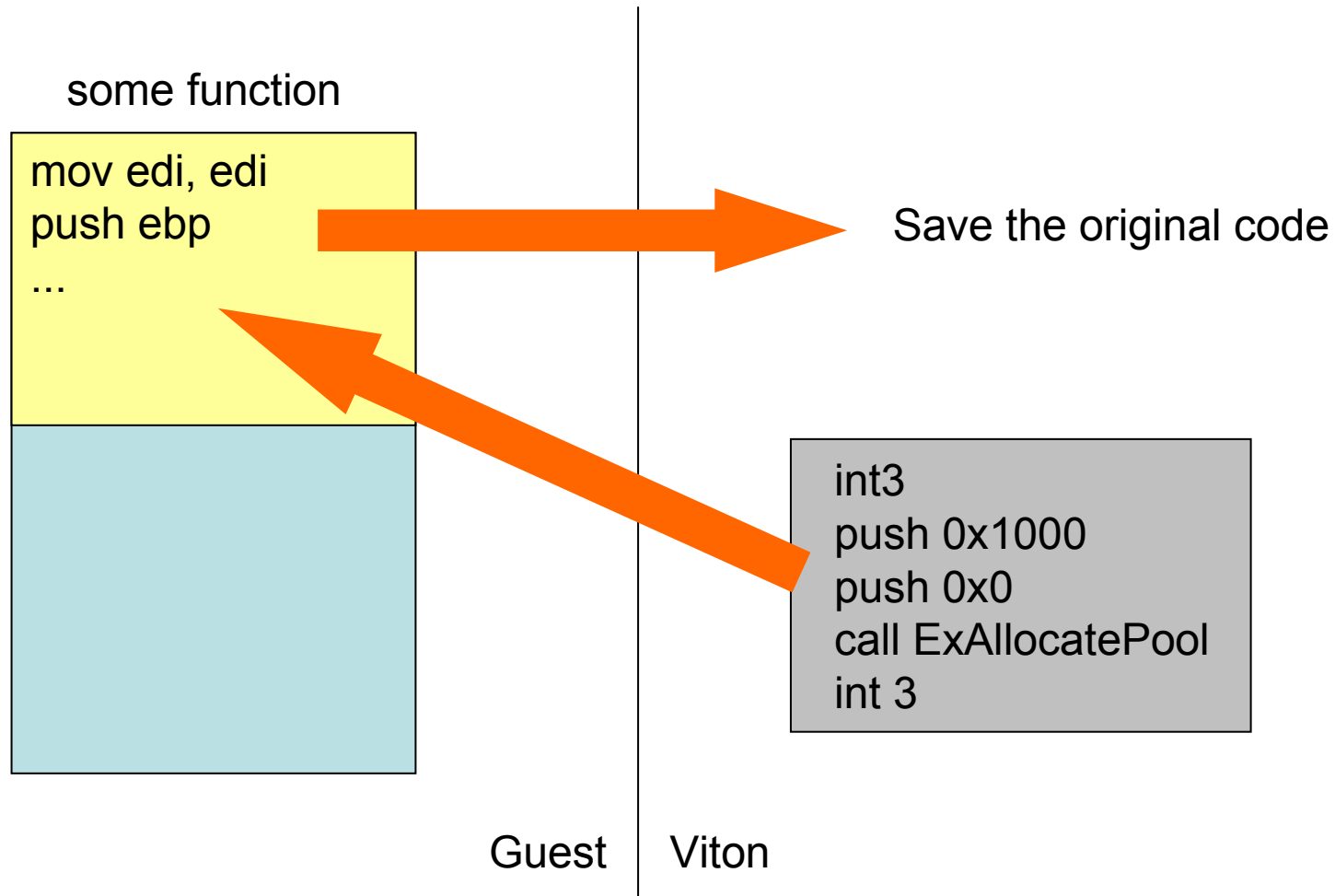


## Guest activity monitoring

- When we use the Viton, no one can modify the kernel code, excluding the Viton.
  
- Viton can monitor the guest's activity by hooking the code
  1. Allocate memory for detours in the guest VA space
  2. Setup the detours buffer
  3. Hook the target function



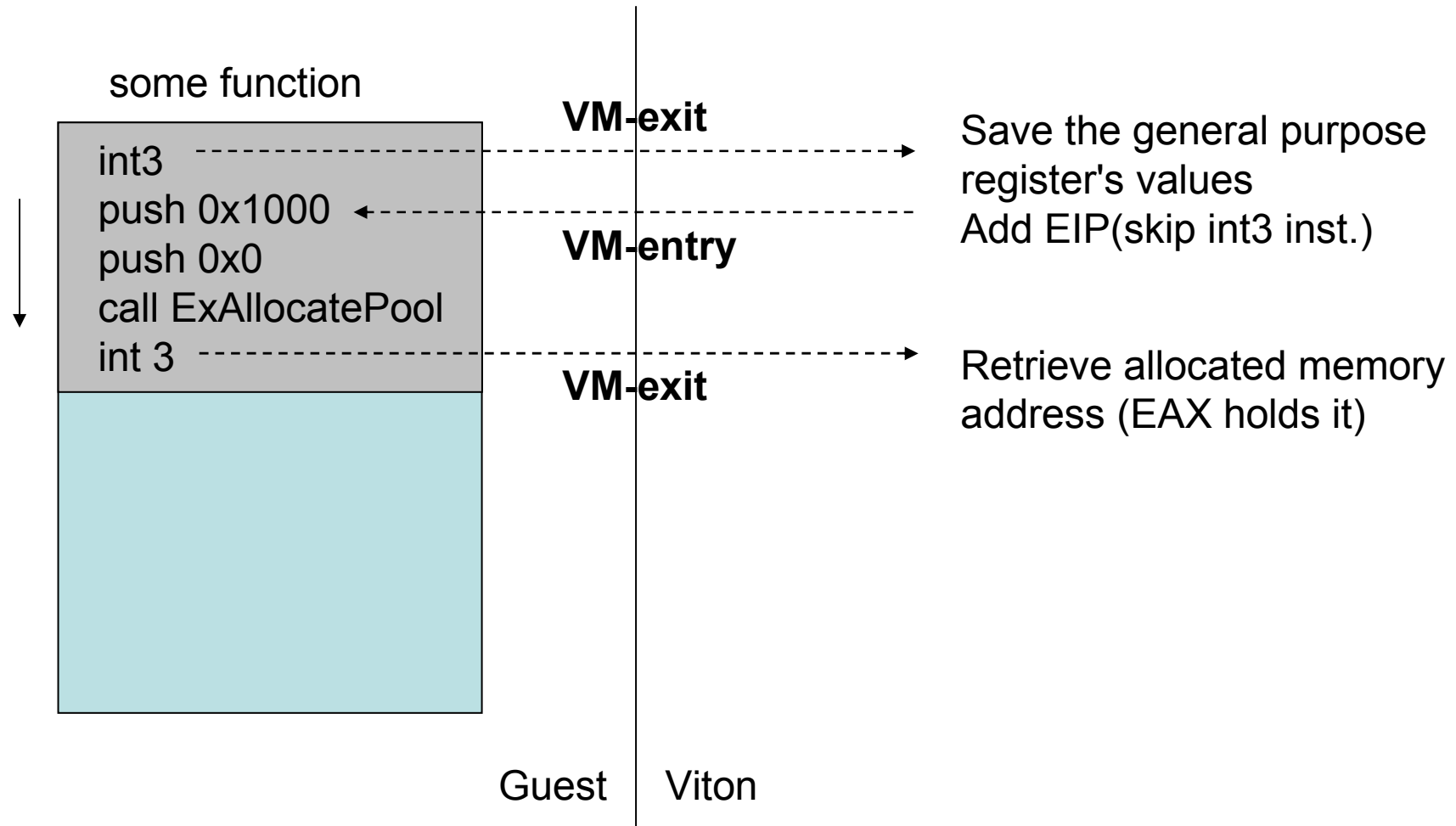
## How to allocate memory in guest VA space





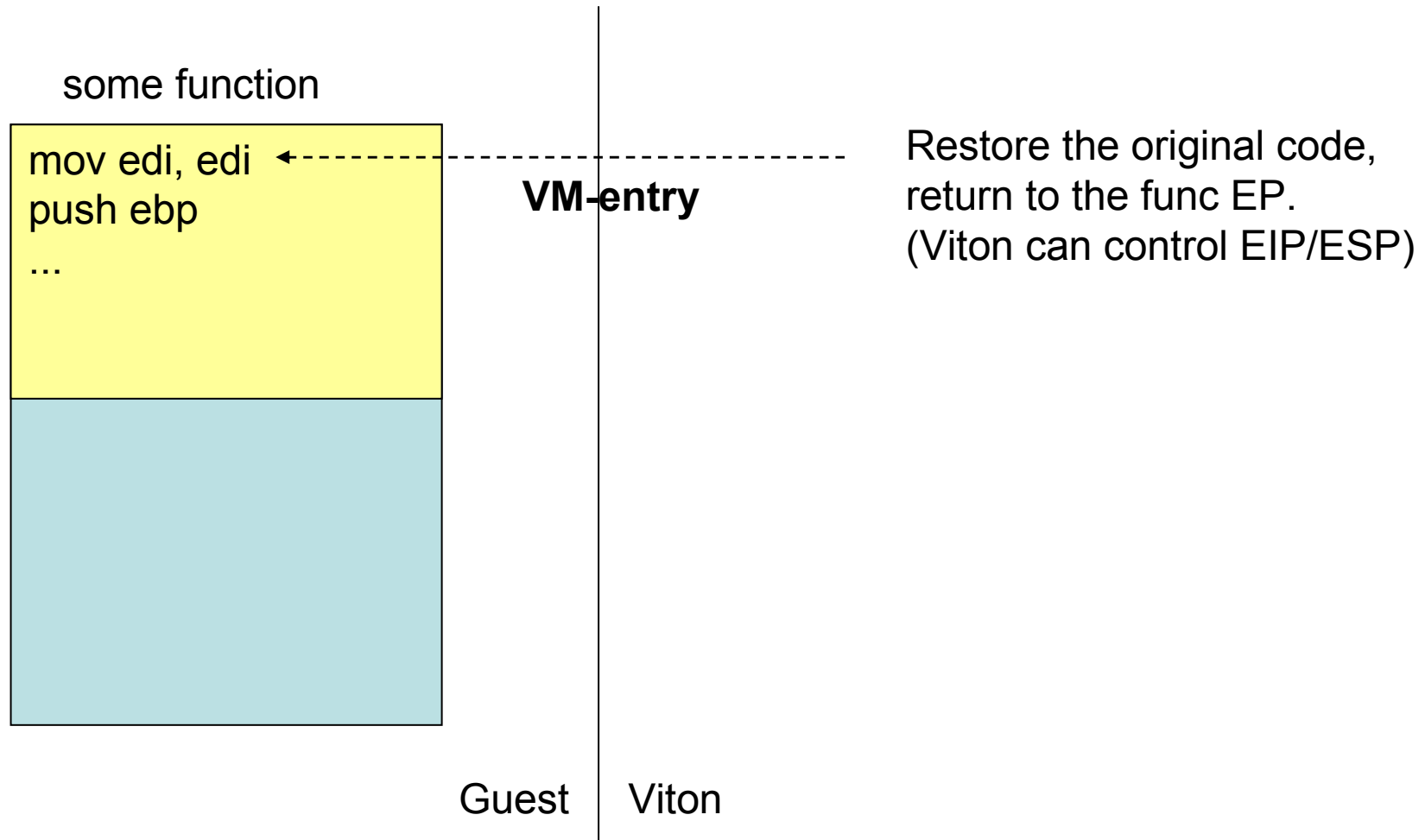


## How to allocate memory in guest VA space



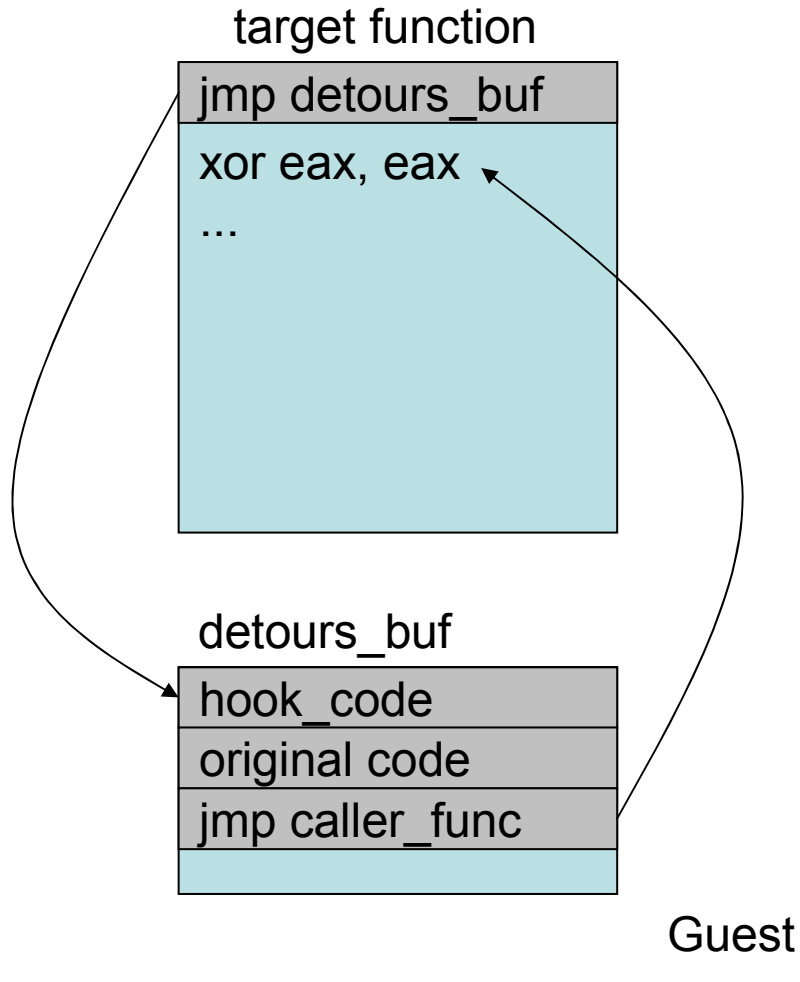


## How to allocate memory in guest VA space





## How to hook the guest code



When the target function is called,

1. jump to the detours\_buf
2. Execute our hook\_code
3. Execute original code which is overwritten by "jmp detours\_buf"
4. jump to the next code of overwritten one

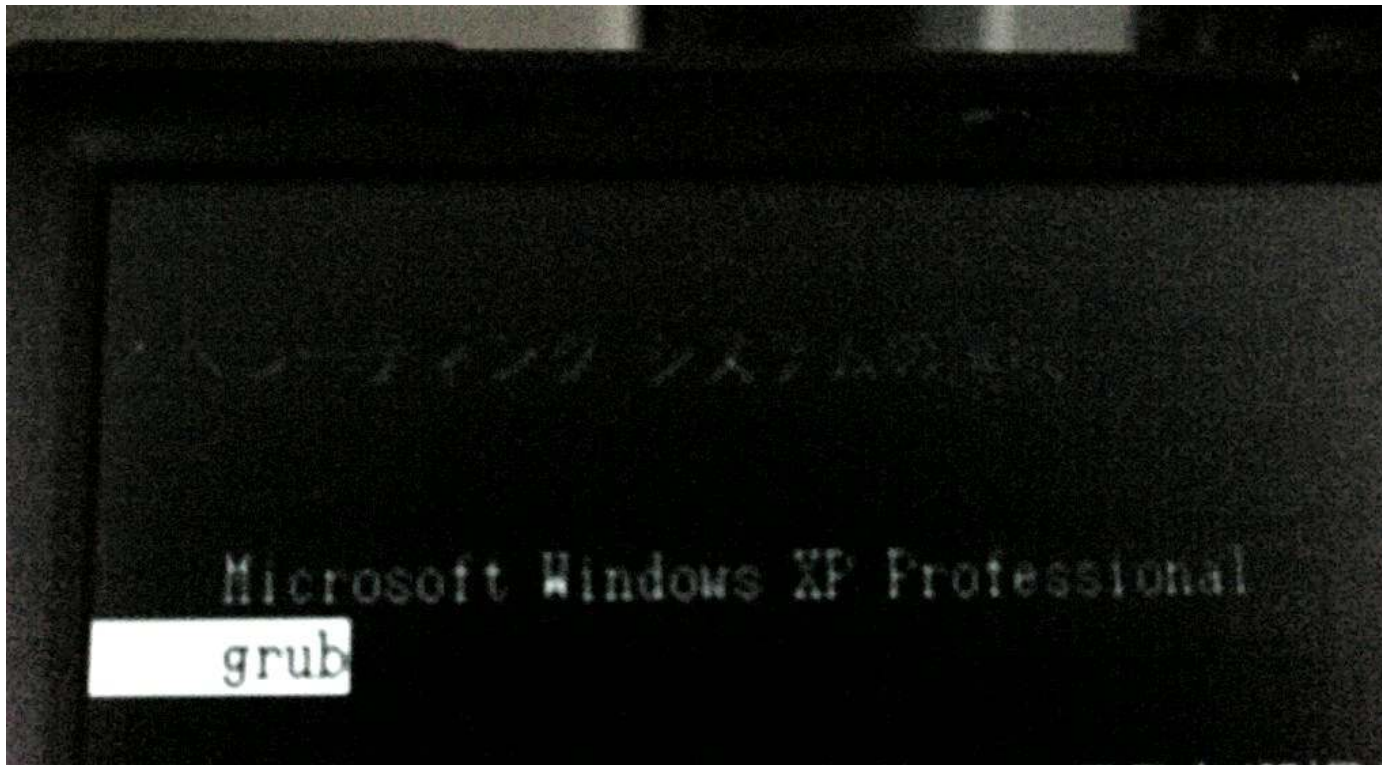


## What can Viton do hooking the guest code ?

- Viton can retrieve the guest information in hook\_code
  - int3 and other inst. that cause VM-exit are useful
- So, Wouldn't you hook below functions ?
  - ZwCreateProcess/ZwTerminateProcess
  - ZwLoadDriver
- Then, Viton understands process, driver and other guest system resource information.

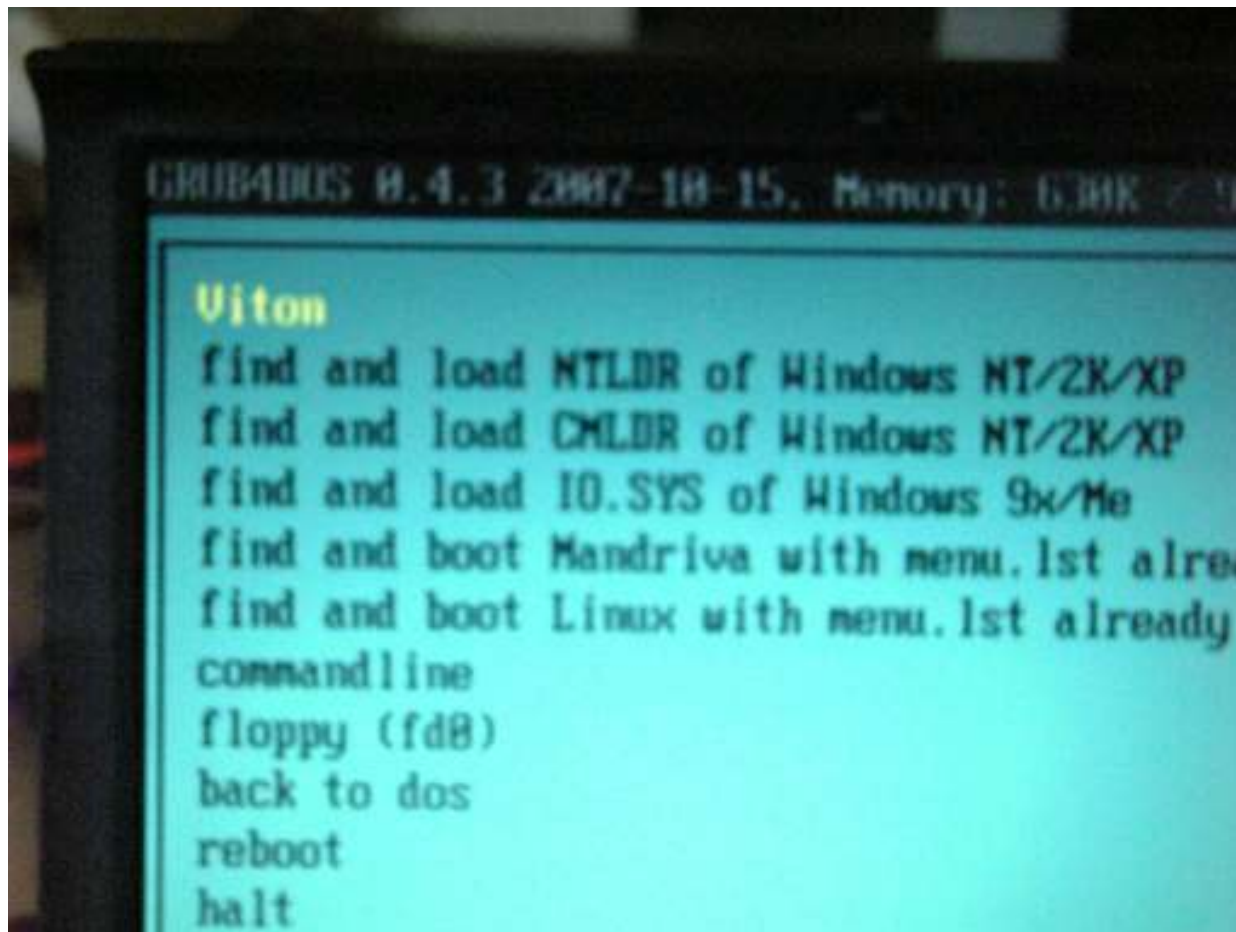


## Demo

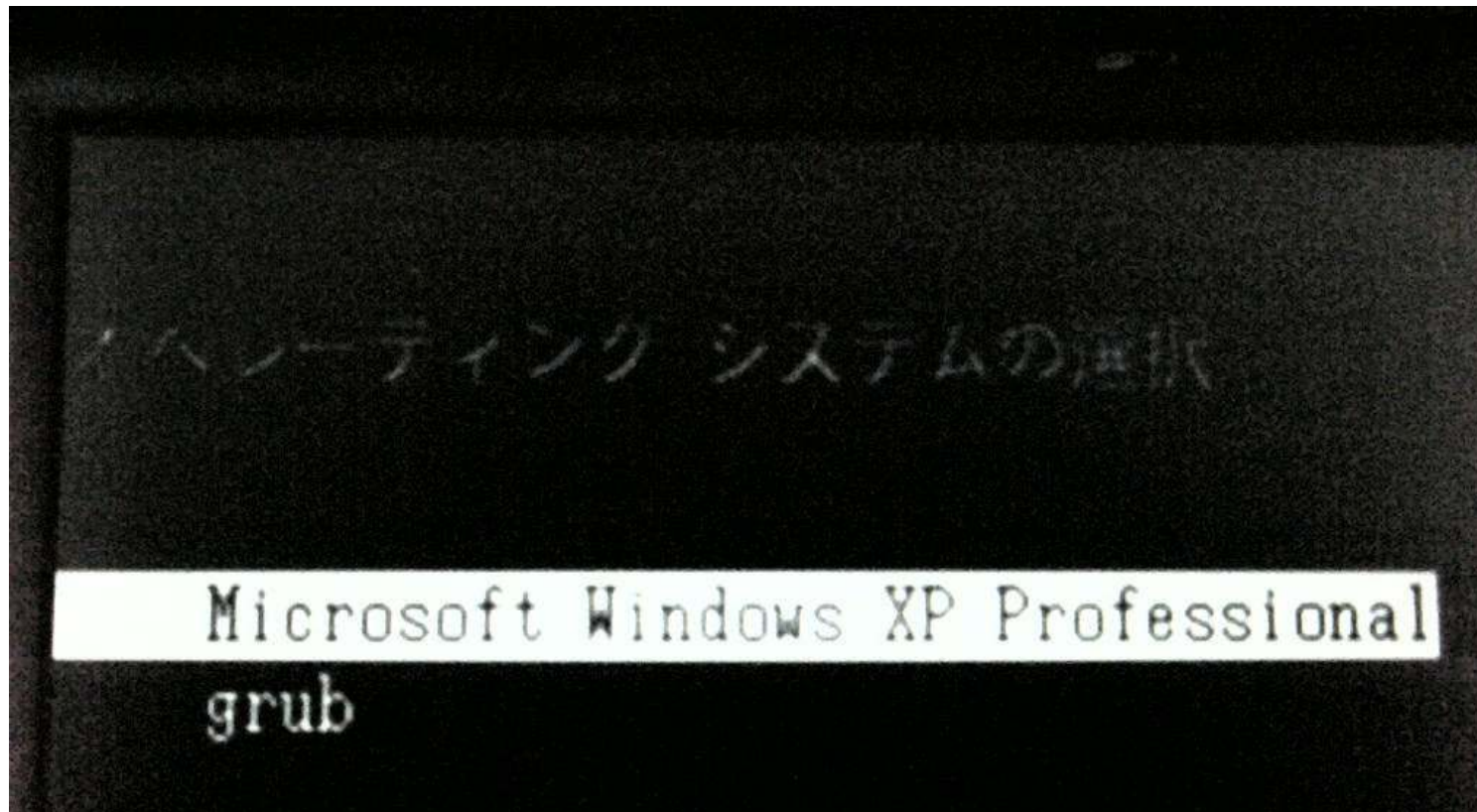




## Demo

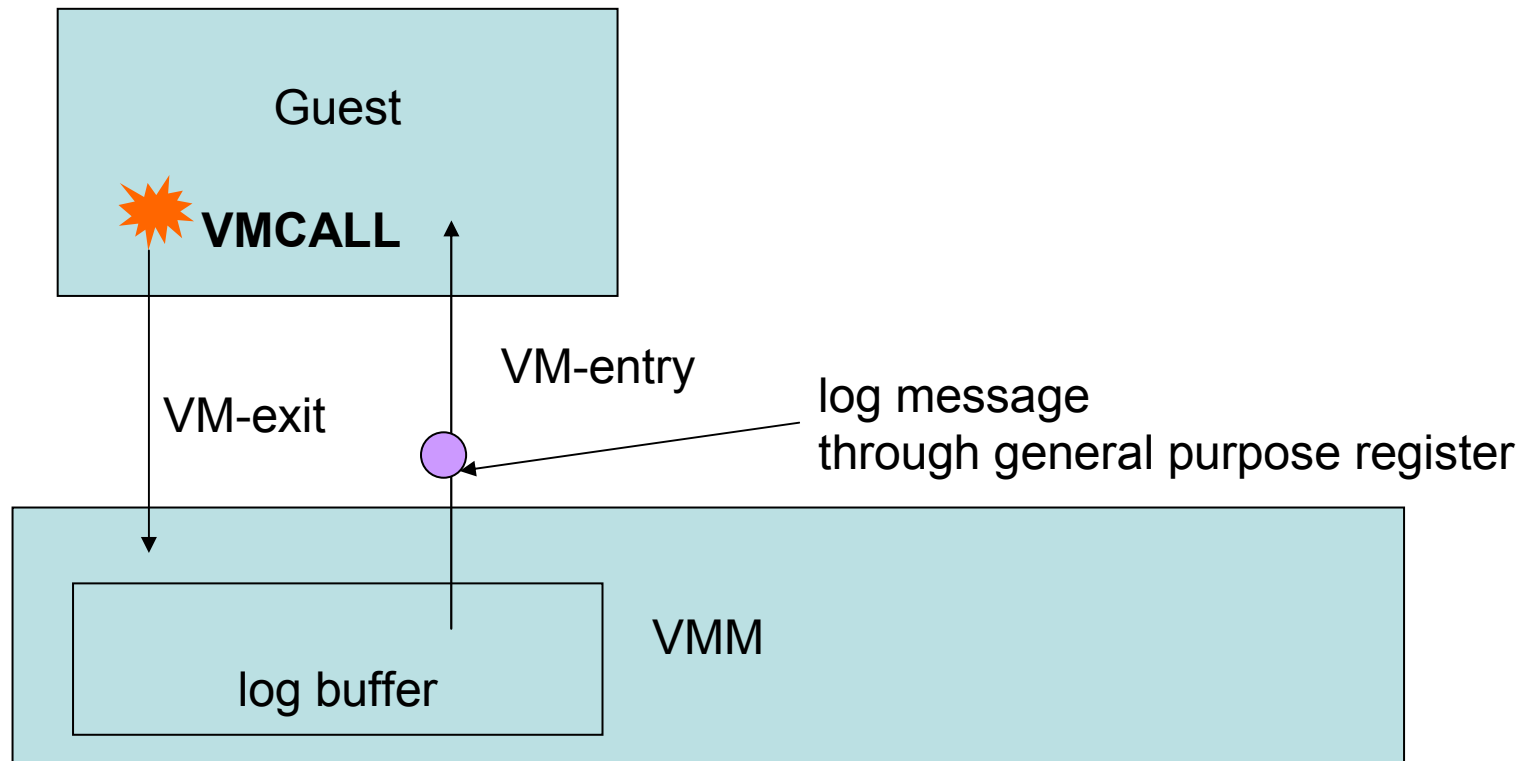


## Demo





## dbgsh (Bitvisor's debugging function)







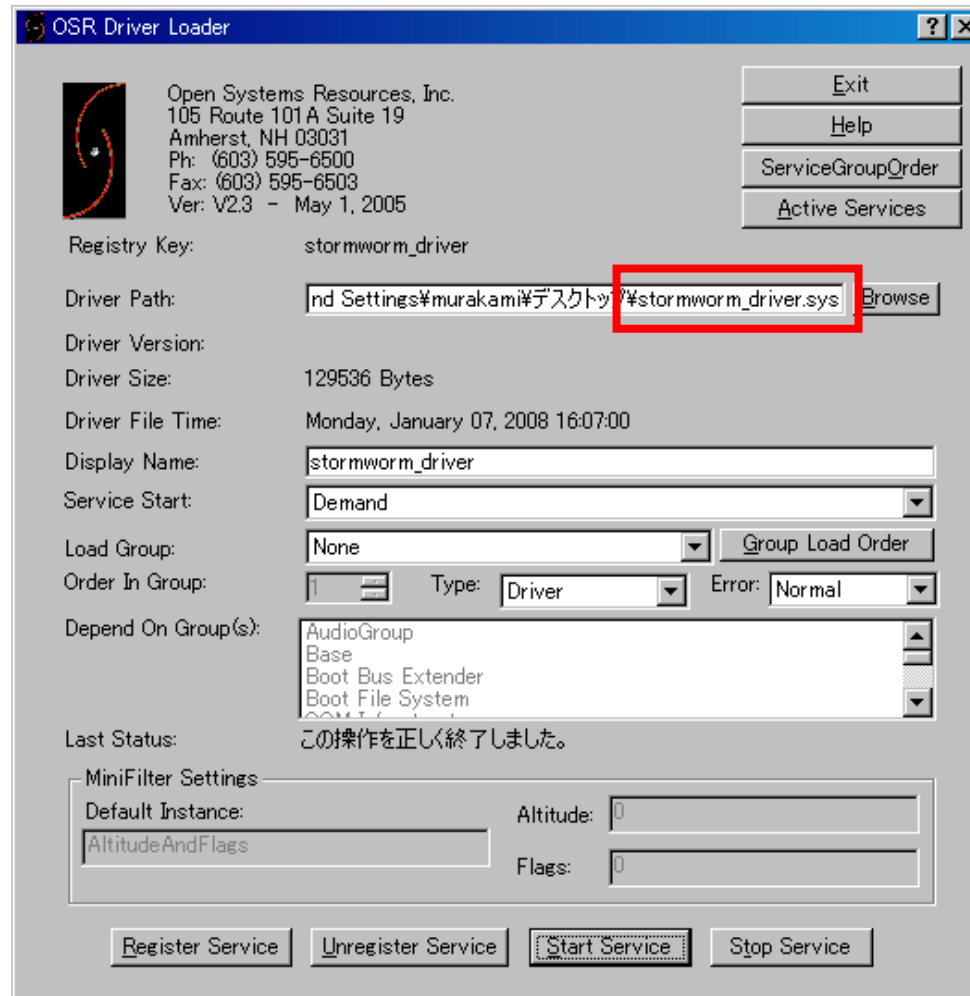
## Demo

```
C:\Documents and Settings\murakami\Desktop\dbgsh.exe
[viton] CRO.WP is enabled
[viton] VMCS_GUEST_IDTR_BASE: 0x8003F400
[viton] Guest IDT[0x2E].handler: 0x80542000
[viton] Start address for searching kernel base: 0x80442000
[viton] Kernel base: 0x804D9000
[viton] 0:      .text 0x00001000 0x00006E46C 0x68000020
[viton] add to ro_list: 0x804DA168 - 0x8054846C
[viton] 1:      POOLMI 0x00070000 0x000011F9 0x68000020
[viton] add to ro_list: 0x80549000 - 0x8054A1F9
[viton] 2:      MISYSPT 0x00072000 0x000006CB 0x68000020
[viton] add to ro_list: 0x8054B000 - 0x8054B6CB
[viton] 3:      POOLCODE 0x00073000 0x000012AE 0x68000020
[viton] add to ro_list: 0x8054C000 - 0x8054D2AE
[viton] 4:      .data 0x00075000 0x00018CE8 0xC8000040
[viton] 5:      INITDATA8 0x0008E000 0x00000038 0xC8000040
[viton] 6:      INITCONSe 0x0008F000 0x00001A65 0x48000040
[viton] 7:      PAGE 0x00091000 0x000DECDF 0x60000020
[viton] add to ro_list: 0x8056A000 - 0x80648CDF
[viton] 8:      PAGELK 0x00170000 0x0000E520 0x60000020
[viton] add to ro_list: 0x80649000 - 0x80657520
[viton] 9:      PAGEVRFY 0x0017F000 0x0000EAA6 0x60000020
[viton] add to ro_list: 0x80658000 - 0x80666AA6
[viton] 10:     PAGEWMI 0x0018E000 0x00001703 0x60000020
[viton] add to ro_list: 0x80667000 - 0x80668703
[viton] 11:     PAGEKD 0x00190000 0x00003D93 0x60000020
```

ro\_list: read only list



# Demo



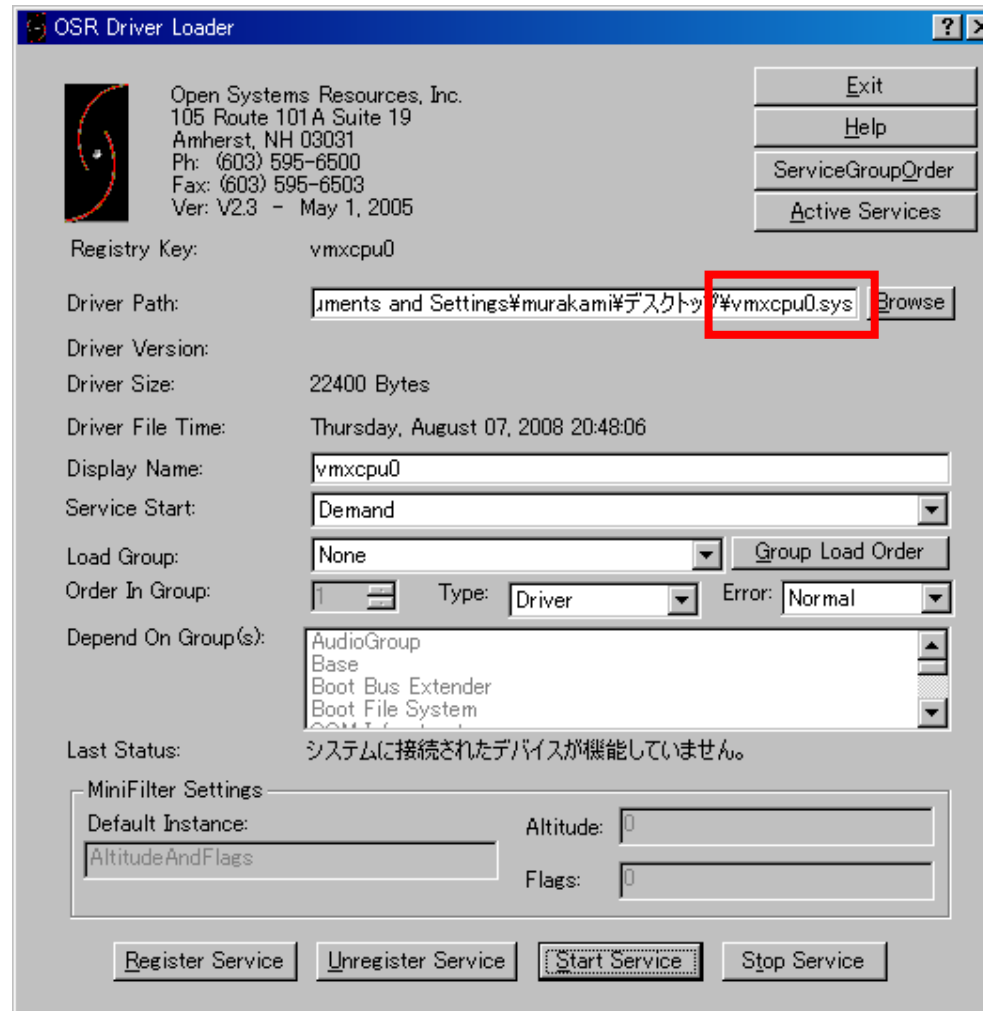


## Demo

```
C:\Documents and Settings\murakami\Desktop\dbgsh.exe
[viton] 12: PAGESPEC! 0x00194000 0x00000E21 0x60000020
[viton] add to ro_list: 0x8066D000 - 0x8066DE21
[viton] 13: PAGEHDLS 0x00195000 0x00001DB8 0x60000020
[viton] add to ro_list: 0x8066E000 - 0x8066FDB8
[viton] 14: .edata 0x00197000 0x0000B57D 0x40000040
[viton] 15: PAGEDATA 0x001A3000 0x000015B8 0xC0000040
[viton] 16: PAGECONS@0 0x001A5000 0x00003040 0x40000040
[viton] 17: PAGEKD 0x001A9000 0x0000C021 0xC0000040
[viton] 18: PAGECONS 0x001B6000 0x0000018C 0xC0000040
[viton] 19: PAGELKCO 0x001B7000 0x00000088 0x40000040
[viton] 20: PAGEVRFC14 0x001B8000 0x00003449 0x40000040
[viton] 21: PAGEVRFDH 0x001BC000 0x00000648 0xC0000040
[viton] 22: INIT 0x001BD000 0x0002BB18 0xE2000020
[viton] 23: .rsrc 0x001E9000 0x00010740 0x40000040
[viton] 24: .reloc 0x001FA000 0x000108F4 0x42000040
[viton] CRO.WP is enabled
[viton] add to ro_list: 0xF77B0590 - 0xF77B0D8F
[viton] get_kernel_symbol("KeAddSystemServiceTable"): 0x805A20F4
[viton] get_kernel_symbol("KeServiceDescriptorTable"): 0x8055D6E0
[viton] add to ro_list: 0x8055D6E0 - 0x8055D6F0
[viton] add to ro_list: 0x80505A70 - 0x80506C30
[viton] CRO.WP modification is detected
[viton] malicious patching: 0x80505CB4
[viton] replace malicious code to infinite loop code(¥xeb¥xfe)
end
```



# Demo





## Demo

DebugView on %NOFUTURE (local)

#	Time	Debug Print
0	7:08:58	[vmm] Driver Routines [00000000]
1	7:08:58	[vmm] ----- [00000000]
2	7:08:58	[vmm] Driver Entry [A605F4A0]
3	7:08:58	[vmm] Driver Unload [A605F3B0]
4	7:08:58	[vmm] StartVMX [A605D640]
5	7:08:58	[vmm] VMMEnterPoint [A605F830]
6	7:08:58	[vmm] VMXONRegion virtual address [A29B1000]
7	7:08:58	[vmm] VMXONRegion physical address [20405000]
8	7:08:58	[vmm] VMCSRegion virtual address [A29AD000]
9	7:08:58	[vmm] VMCSRegion physical address [185C6000]
10	7:08:58	[vmm] FakeStack [8475B000]
11	7:08:58	[vmm] Guest Return EIP [A605F76E]
12	7:08:58	[vmm] Enabling VMX mode on CPU 0 [00000000]
13	7:08:58	[vmm] Running on Processor [00000000]
14	7:08:58	[vmm] GDT Base [8003F000]
15	7:08:58	[vmm] GDT Limit [000003FF]
16	7:08:58	[vmm] IDT Base [8003F400]
17	7:08:58	[vmm] IDT Limit [000007FF]
18	7:08:58	[vmm] VMX Support Not Present. [0000E39D]
19	7:08:58	[vmm] Running on Processor [00000000]
20	7:08:58	[vmm] ERROR: Launch aborted. [00000000]



## Viton vs.

- Type I
  - Easy
- Type II
  - DKOM: Difficult, but possible
  - KOH: Difficult, we need more research, and breakthrough
- Type III
  - Easy (First come, first served)



## 4. Conclusions

- Virtualization Technology becomes a help to protect the kernel
- However, it is not a silver bullet
  - Foundation for existing security solutions

**Thank you!**



**Fourteenforty Research Institute, Inc.**

<http://www.fourteenforty.jp>

Senior Research Engineer

Junichi Murakami <[murakami@fourteenforty.jp](mailto:murakami@fourteenforty.jp)>