The Story of NtVDM Subsystem

wang yu

SyScan(+)360, 2013

Part One

Introduction

Introduction

- About me （wangyu@360.cn）
- Background

NtDVM subsystem is an interesting but neglected Windows component. When we enjoy playing DOS game under windows platform with the help of NtDVM, maybe we should also focus on its security properties. The exposure of CVE-2004-0208 and CVE-2012-2553 etc. indicates that the security of NtDVM deserves a further research.

This talks discusses the implementation of NtVDM in both user and kernel mode, and how the emulator works. Also we will talk the root cause of previously discovered vulnerabilities and the lessons we have learnt.

Introduction

- Outline

  - Intel SDM-3B 8086 Emulation

  - SoftPC / NtVDM architecture - the implementation of NtDVM in both user and kernel mode

  - DOS simulator's details - DOS emulation, Virtual 8086 Mode switch, BOP mechanism and event callbacks

  - Introduction to NtDVM subsystem security

  - Lessons we have learnt from the vulnerabilities

- Disclaimer

Part Two

NtVDM Subsystem Design and Implementation

Intel SDM 8086 Emulation Overview

- Compare IA-32 Real Address Mode with 8086 Processor's execution environment
  - Support FS, GS segment registers
  - Support 32-bits operands and 32-bits addressing
  - Support more instructions (LIDT / SIDT)
- Compare Virtual 8086 Mode with IA-32 Real Address Mode's execution environment
  - reuses the mechanisms of interrupts and exception handling
  - reuses the paging mechanisms of protected mode

"Virtual-8086 mode always executes at CPL 3."

"Use the U/S flag of page-table entries to protect the virtual-8086 monitor and other system software in the virtual-8086 mode task space."

Intel SDM 8086 Emulation Overview

A virtual-8086-mode task consists of the following items:

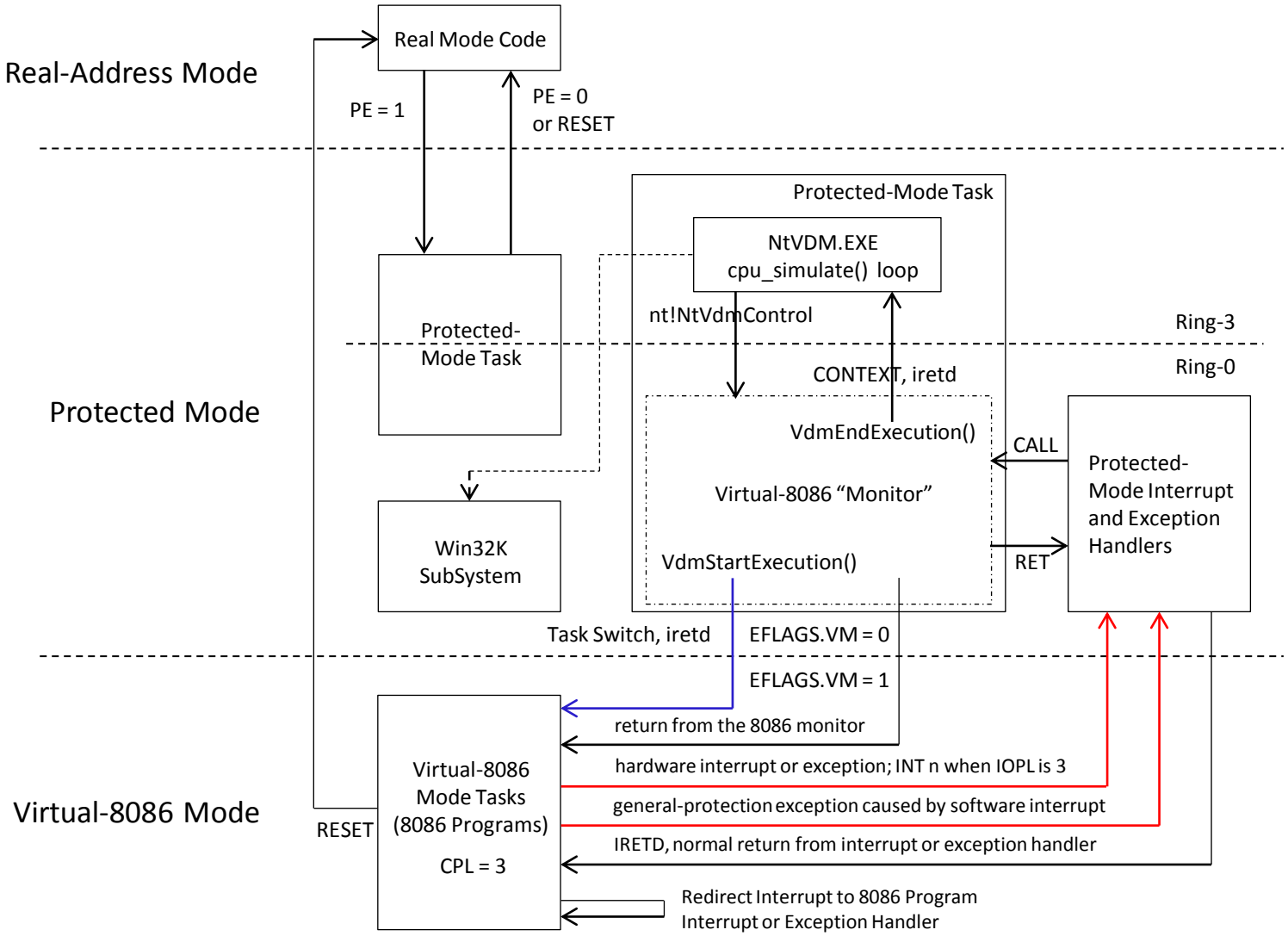
- A 32-bit TSS for the task.
- The 8086 program.
- 8086 operating-system services.
- A virtual-8086 monitor.

The processor enters virtual-8086 mode to run the 8086 program and returns to protected mode to run the virtual-8086 monitor.

The virtual-8086 monitor is a 32-bit protected-mode code module that runs at a CPL of 0. The monitor consists of initialization, interrupt- and exception-handling, and I/O emulation procedures that emulate a personal computer or other 8086-based platform.

The processor can leave the virtual-8086 mode only through an interrupt or exception.

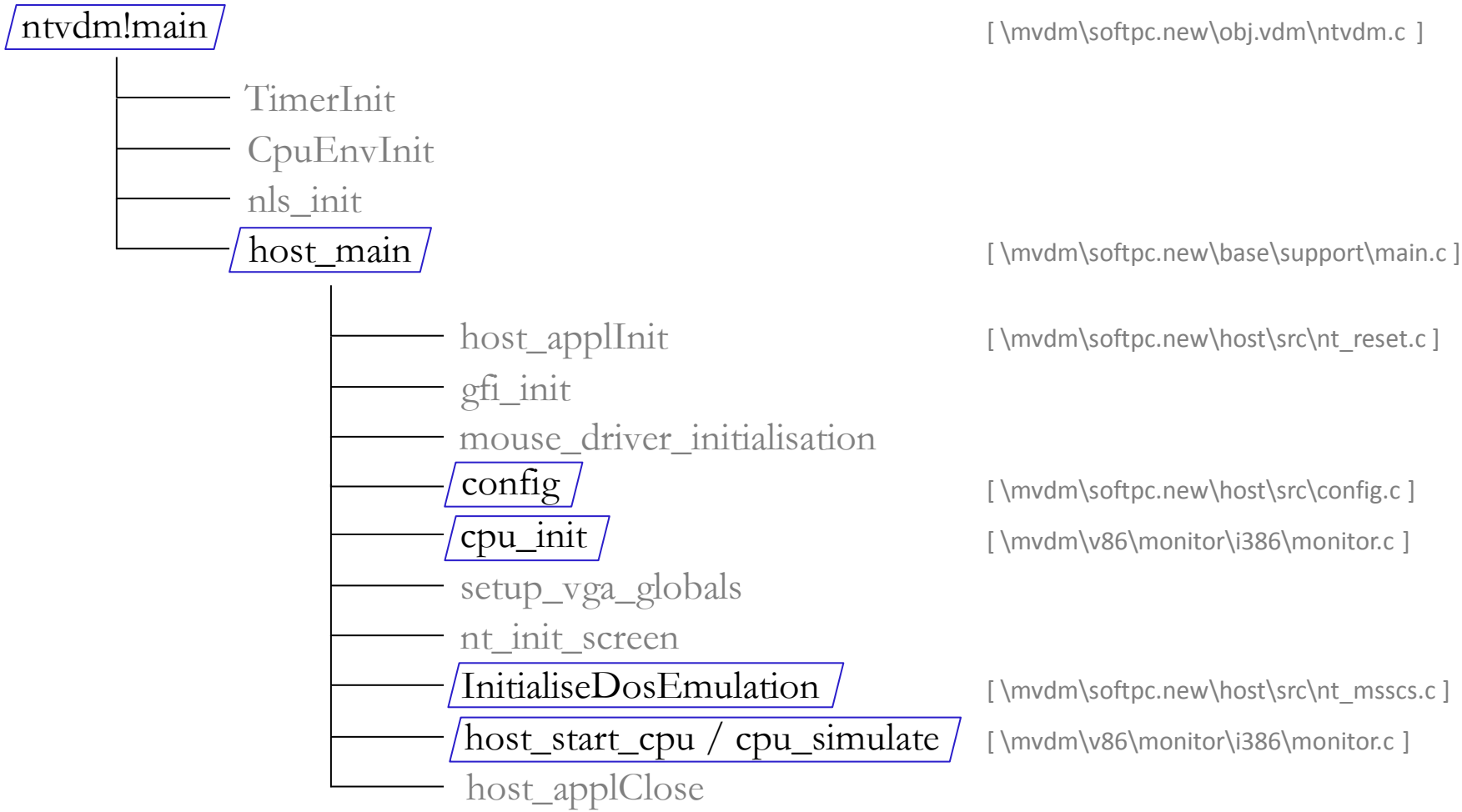
Intel SDM 8086 Emulation Overview



Reference : Figure 20-3. Entering and Leaving Virtual-8086 Mode

Simulator's User Mode — NtVDM Process Logic

NtVDM Process Sequence Overview



Simulator's User Mode — NtVDM Process Logic

host_main

- host_applInit
  - video_funcs、keybd_funcs、mouse_funcs initialization
  - Query EPROCESS's PS_PROCESS_FLAGS_VDM_ALLOWED flag
  - Parse command line, "-i" means DosSessionId
  - Create thread - ConsoleEventThread
- gfi_init
- mouse_driver_initialisation
- config
  - Set DOS window title twice
    the first format is "ntvdm - ProcessId . ThreadId . ConsoleHandle"
  - Input VDMINFO, and asynchronous call GetNextVDMCommand routine, and then query PIF infomation
    -> ntdll!CsrClientCallServer -> nt!ALPC -> SERVER CSRSS
    -> CSRSRV!CsrApiRequestThread -> basesrv!BaseSrvGetNextVDMCommand
  - Calculate memory usage, call GetROMsMapped - NtVdmControl(VdmInitialize)
    create VdmObjects(VDM_PROCESS_OBJECTS) based on _VDM_INITIALIZE_DATA
    initialize zreo memory address based on "\Device\PhysicalMemory"

```
0: kd> da 011ff6f4
011ff6f4  "ntvdm-148.808.320001"
```

```
0: kd> db 0
00000000  53 ff 00 f0 53 ff 00 f0-c3 e2 00 f0 53 ff 00 f0  S...S.......S...
00000010  53 ff 00 f0 54 ff 00 f0-4a 82 00 f0 53 ff 00 f0  S...T...J...S...
00000020  a5 fe 00 f0 87 e9 00 f0-01 0b 00 f0 01 0b 00 f0  ................
```

Simulator's User Mode — NtVDM Process Logic

host_main

- cpu_init
  - Call NtVdmControl(VdmFeatures) to query KeI386VirtualIntExtensions's attribute
  - Initialize FIXED_NTVDMSTATE_LINEAR (0x714)
  - fninit initialize FPU
  - Allocate and initialize _TEB.Vdm in _VDM_TIB

```
ntvdm!VDM_TIB
   Size                : Uint4B
   VdmInterruptTable   : Ptr32
   VdmFaultTable       : Ptr32
   MonitorContext      : _CONTEXT
   VdmContext          : _CONTEXT
   EventInfo           : _VdmEventInfo
   ......
   ContinueExecution   : UChar
```

- setup_vga_globals
- nt_init_screen
- InitialiseDosEmulation
  - Prepare DOS execution environment
  - scs_init asyn-call GetNextVDMCommand query IsFirstVDM information
    -> ntdll!CsrClientCallServer -> nt!ALPC -> basesrv!BaseSrvIsClientVdm
  - Loading ntio***.sys to NTIO_LOAD_SEGMENT : NTIO_LOAD_OFFSET

```
0: kd> da 011ff524
011ff524  "C:\Windows\system32\ntio804.sys"
```

  - setCS(NTIO_LOAD_SEGMENT); setIP(NTIO_LOAD_OFFSET);
    Set VdmTib.VdmContext as an entry point, this is Virtual 8086 Mode 's entry point !

```
0: kd> dt _CONTEXT 0x02f32f30+0x2d8
   +0x0b8 Eip              : 0
   +0x0bc SegCs            : 0x70
   +0x0c0 EFlags           : 0x202
```

Simulator's User Mode — NtVDM Process Logic

host_main

host_start_cpu / cpu_simulate

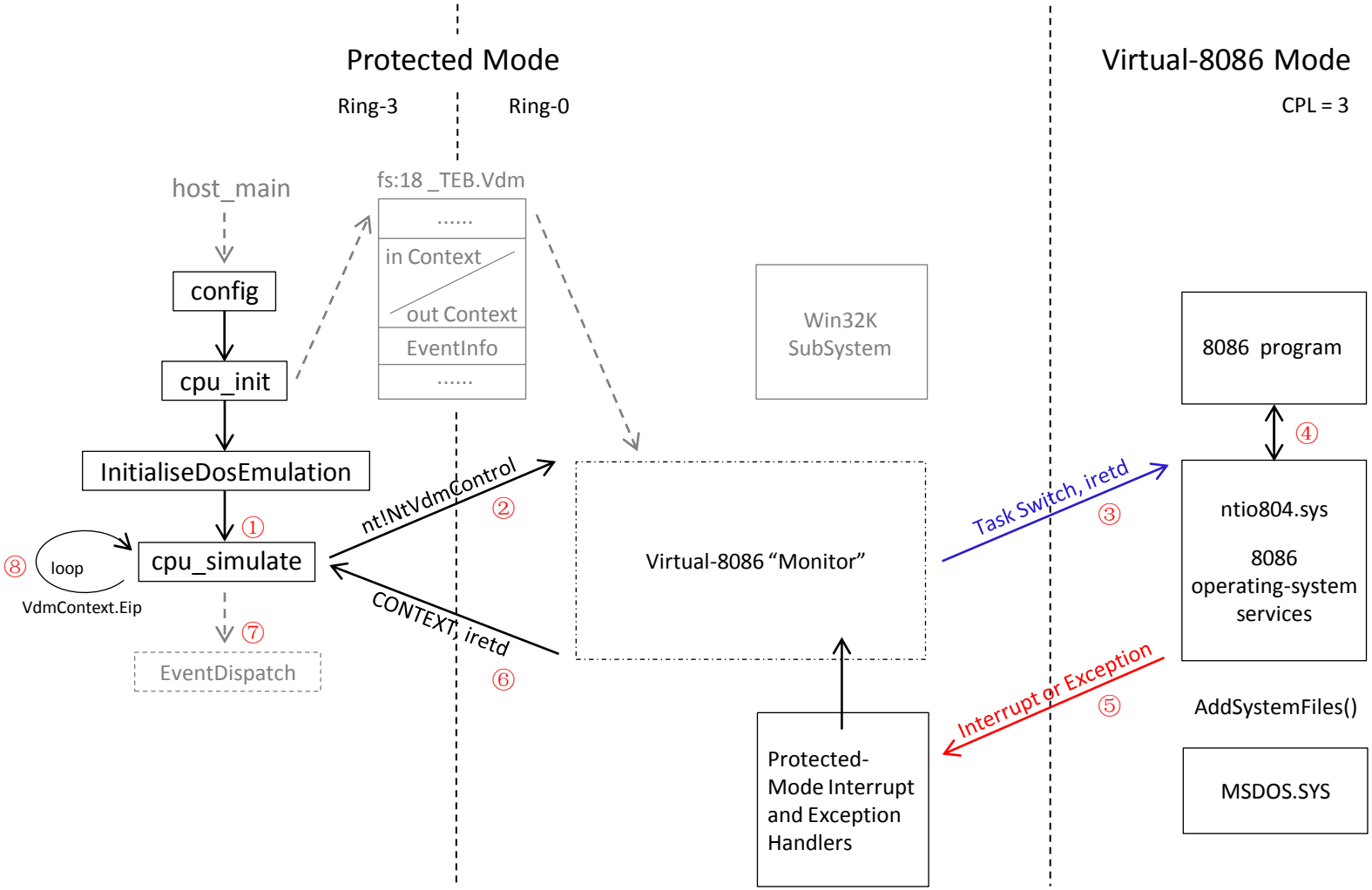
- Set flag, like ContinueExecution = TRUE; etc.
- Call NtVdmControl(VdmStartExecution) enter kernel mode, and then switch to Virtual 8086 Mode by iretd instruction
- The processor can leave the Virtual-8086 Mode only through an interrupt or exception. Monitor sets VDM_TIB.EventInfo field, and then back to user mode
- Modify VdmTib.VdmContext.Eip, prepare for next loop
- Check event type and invoke corresponding event handler

```
while (ContinueExecution) {
    ......
    if (*pNtVDMState & VDM_INTERRUPT_PENDING) {
            DispatchInterrupts();
        }
    ......
    Status = NtVdmControl(VdmStartExecution,NULL);
    ......
    if (!NT_SUCCESS(Status)) {
#if DBG
            DbgPrint("NTVDM: Could not start execution\n");
#endif
            return;
        }
 }
```

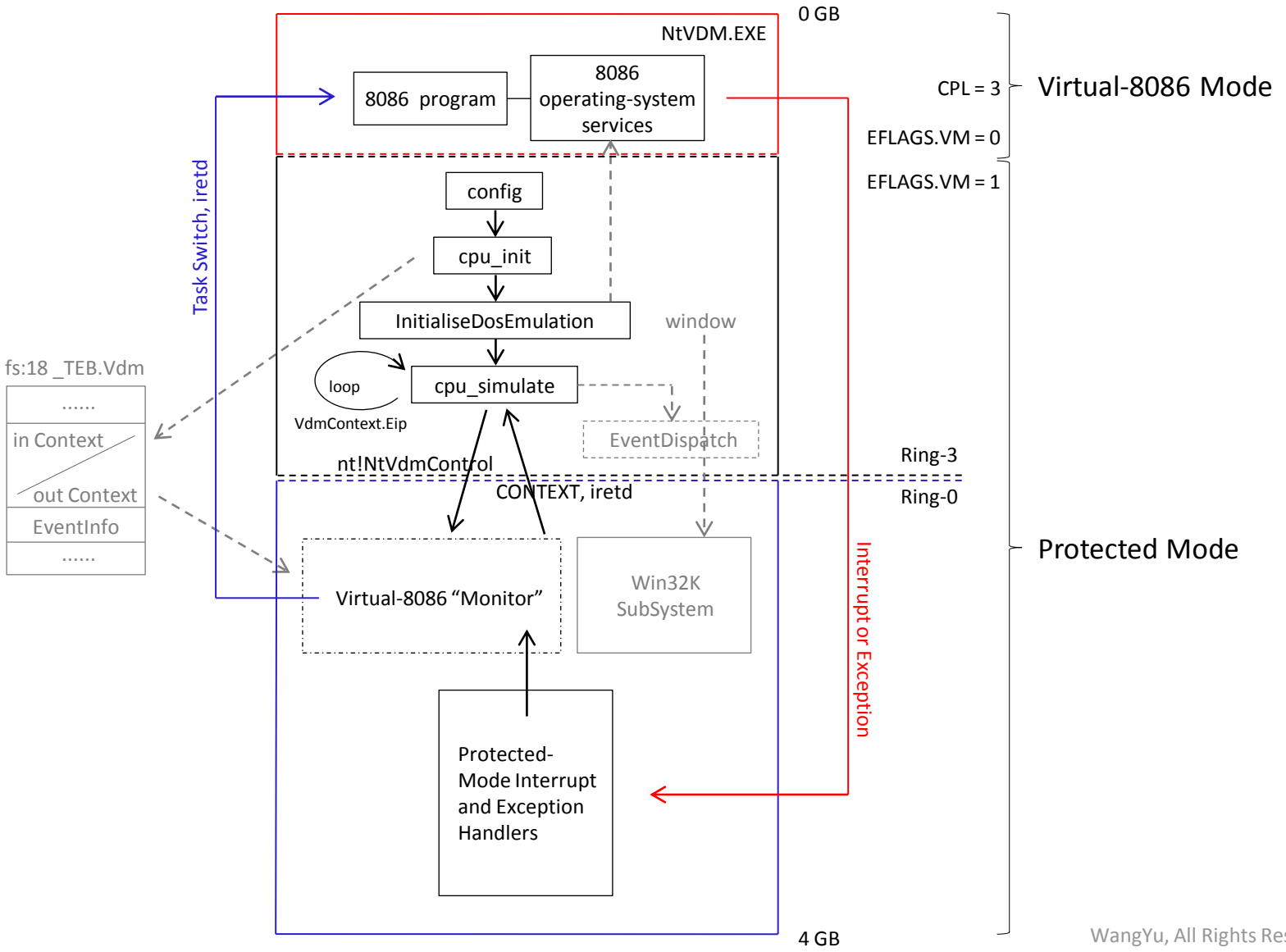
```
//
// Event Dispatch table
//

VOID (*EventDispatch[VdmMaxEvent])(VOID) = {
       ntvdm!EventVdmInterceptDiv
       ntvdm!EventVdmIo
       ntvdm!EventVdmStringIo
       ntvdm!EventVdmMemAccess
       ntvdm!EventVdmIntAck
       ntvdm!EventVdmHandShakeAck
       ntvdm!EventVdmBop
       ntvdm!EventVdmError
       ntvdm!EventVdmIrq13
       };
```

From User Mode Aspect to View Working Flow



Let's Think Different

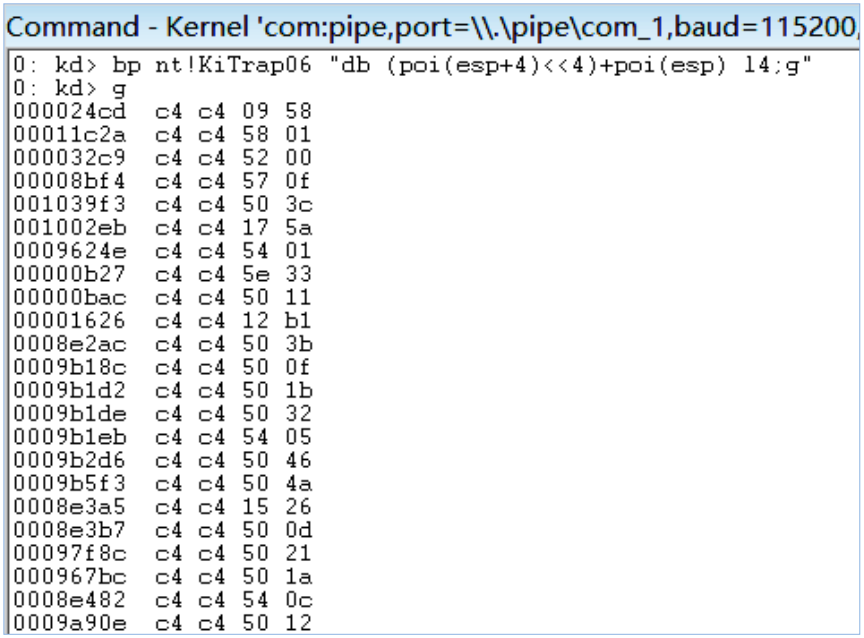


Simulator's User Mode — BOP Mechanism

DEMO : Calling Win32 from DOS

BOP, for BIOS Operation

0xc4, 0xc4 - sort of LEA but with register-register operands which is invalid form



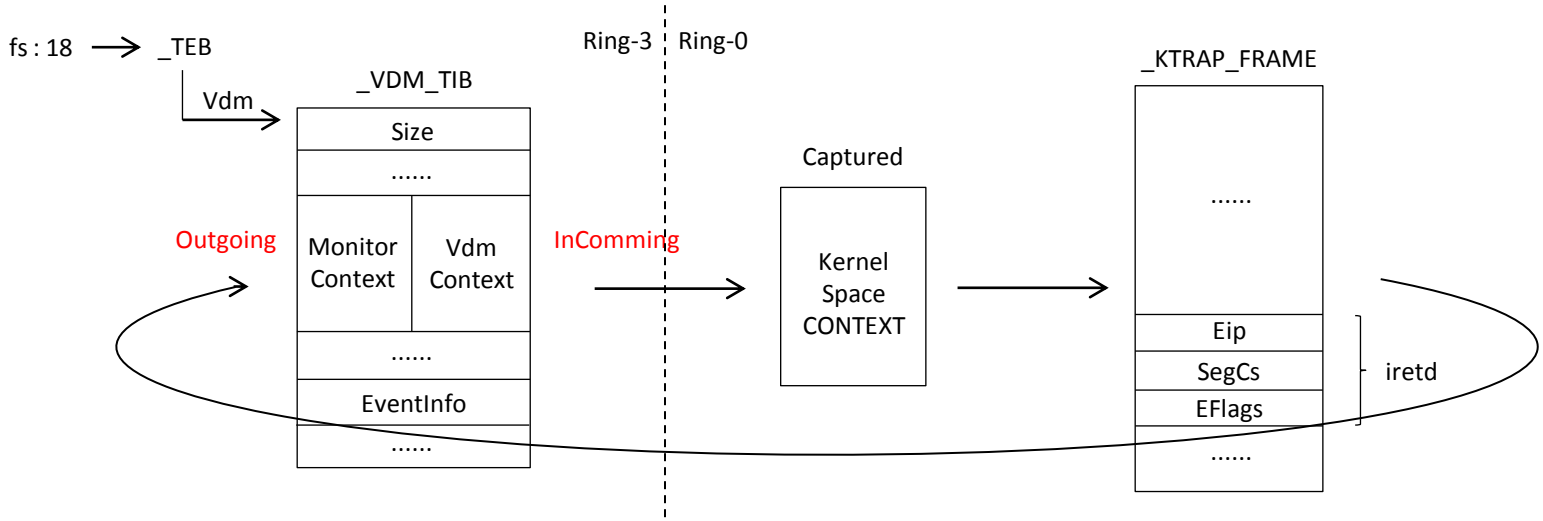
Simulator's Kernel Mode — VdmpStartExecution

NtVdmControl

└ VdmpStartExecution

- Check the VdmAllowed / VdmObjects fields in EPROCESS Structure
- Check _VDM_TIB parameter by calling VdmpGetVdmTib routine
- Resever TrapFrame space from _KTHREAD.InitialStack
- Capture _VDM_TIB.VdmContext into kernel. aka "InComming"
- Call VdmSwapContexts to swap CONTEXT information

-> NtVdmControl -> VdmpStartExecution -> KiFastCallEntry
-> KiServiceExit -> Kei386EoiHelper -> iretd -> Virtual 8086 Mode



Simulator's Kernel Mode — VdmSwapContexts

Before

```
0: kd> dt 02f32f3c _context
nt!_CONTEXT
   +0x000 ContextFlags     : 0
   +0x004 Dr0              : 0
    .....
   +0x09c Edi              : 0
   +0x0a0 Esi              : 0
   +0x0a4 Ebx              : 0
   +0x0a8 Edx              : 0
   +0x0ac Ecx              : 0
   +0x0b0 Eax              : 0
   +0x0b4 Ebp              : 0
   +0x0b8 Eip              : 0
   +0x0bc SegCs            : 0
   +0x0c0 EFlags           : 0
   +0x0c4 Esp              : 0
   +0x0c8 SegSs            : 0
```

```
0: kd> dt _ktrap_frame 9dd48d64
nt!_KTRAP_FRAME
   +0x034 SegEs            : 0x23
   +0x038 SegDs            : 0x23
   +0x03c Edx              : 0x77bd9a94
   +0x040 Ecx              : 0x5dd09aec
   +0x044 Eax              : 0x7ffdf000
   +0x048 PreviousPreviousMode : 1
   +0x050 SegFs            : 0x3b
   +0x054 Edi              : 0x200
   +0x058 Esi              : 0x2f32f30
   +0x05c Ebx              : 0
   +0x060 Ebp              : 0x11ff738
   +0x064 ErrCode          : 0
   +0x068 Eip              : 0x77bd9a94
   +0x06c SegCs            : 0x1b
   +0x070 EFlags           : 0x246
   +0x074 HardwareEsp      : 0x11ff718
   +0x078 HardwareSegSs    : 0x23
```

After

```
0: kd> dt 02f32f3c _context
nt!_CONTEXT
   +0x000 ContextFlags     : 0
   +0x004 Dr0              : 0
    .....
   +0x09c Edi              : 0x200
   +0x0a0 Esi              : 0x2f32f30
   +0x0a4 Ebx              : 0
   +0x0a8 Edx              : 0x77bd9a94
   +0x0ac Ecx              : 0x5dd09aec
   +0x0b0 Eax              : 0x7ffdf000
   +0x0b4 Ebp              : 0x11ff738
   +0x0b8 Eip              : 0x77bd9a94
   +0x0bc SegCs            : 0x1b
   +0x0c0 EFlags           : 0x246
   +0x0c4 Esp              : 0x11ff718
   +0x0c8 SegSs            : 0x23
```

```
0: kd> u 0x77bd9a94
ntdll!KiFastSystemCallRet :
77bd9a94 c3              ret
```

```
0: kd> dt _ktrap_frame 9dd48d64
nt!_KTRAP_FRAME
   +0x034 SegEs            : 0x23
   +0x038 SegDs            : 0x23
   +0x03c Edx              : 0
   +0x040 Ecx              : 0
   +0x044 Eax              : 3
   +0x048 PreviousPreviousMode : 1
   +0x050 SegFs            : 0x3b
   +0x054 Edi              : 0
   +0x058 Esi              : 0
   +0x05c Ebx              : 0
   +0x060 Ebp              : 0
   +0x064 ErrCode          : 0
   +0x068 Eip              : 0
   +0x06c SegCs            : 0x70
   +0x070 EFlags           : 0xa0202
   +0x074 HardwareEsp      : 0
   +0x078 HardwareSegSs    : 0
```

```
0: kd> u 81874590
nt!Kei386EoiHelper+0x128 :
83c43c          add     esp,3Ch
5a              pop     edx
59              pop     ecx
58              pop     eax
8d6554          lea     esp,[ebp+54h]
5f              pop     edi
5e              pop     esi
5b              pop     ebx
5d              pop     ebp
66817c24088000  cmp     word ptr [esp+8],80h
7706            ja      818745ac
83c404          add     esp,4
cf              iretd
```

Simulator's Kernel Mode — VdmEndExecution

nt!KiTrap06 #UD

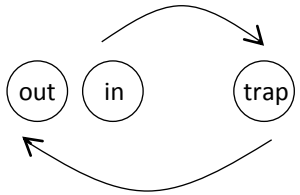
VdmEndExecution

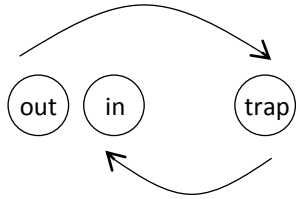
- Virtual 8086 Mode returns to Protected Mode with EIP、CS、EFLAGS pushed onto the stack

```
0: kd> dd esp
9be4bdcc  00000427 00000070 000b0246 00000700

0: kd> db (70<<4)+427
00000b27  c4 c4 5e 33 d2 8e da 8e-c2 33 c0 bf 34 05 ab ab  ..^3.....3..4...
00000b37  8c c8 c7 06 6c 00 85 01-a3 6e 00 c7 06 a4 00 54  ....l....n.....T
```

- Resever TrapFrame space from current stack and then initialize it
- Call VdmDispatchBop routine to dispatch BOP event, ProbeForWrite EventInfo field in fs:18
- Call VdmSwapContexts swap CONTEXT information
- Returns to user mode's ntvdm!cpu_simulate





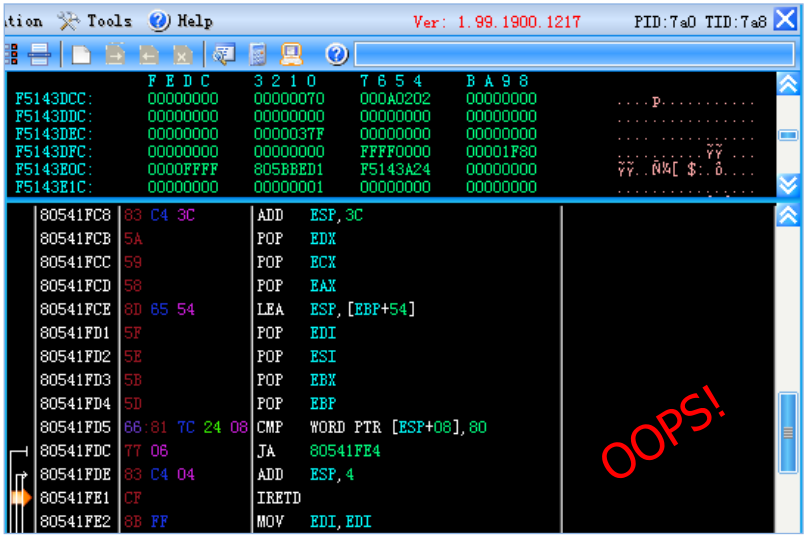
-> nt!KiTrap06 -> VdmDispatchBop -> VdmEndExecution
-> Kei386EoiHelper -> iretd -> ntvdm!cpu_simulate

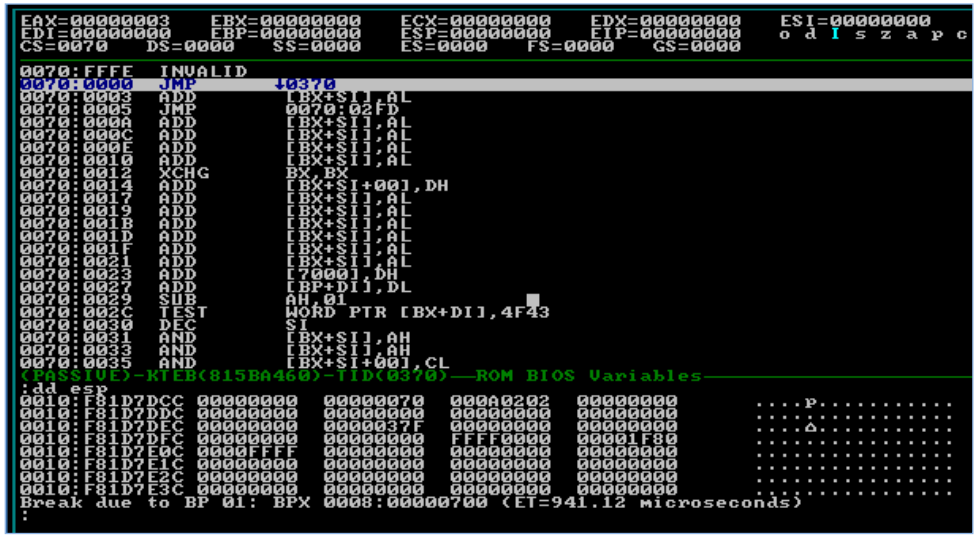
```
1: kd> dt _ktrap_frame 9be4bd64
nt!_KTRAP_FRAME
   +0x068 Eip              : 0x427
   +0x06c SegCs            : 0x70
   +0x070 EFlags           : 0xb0246
```

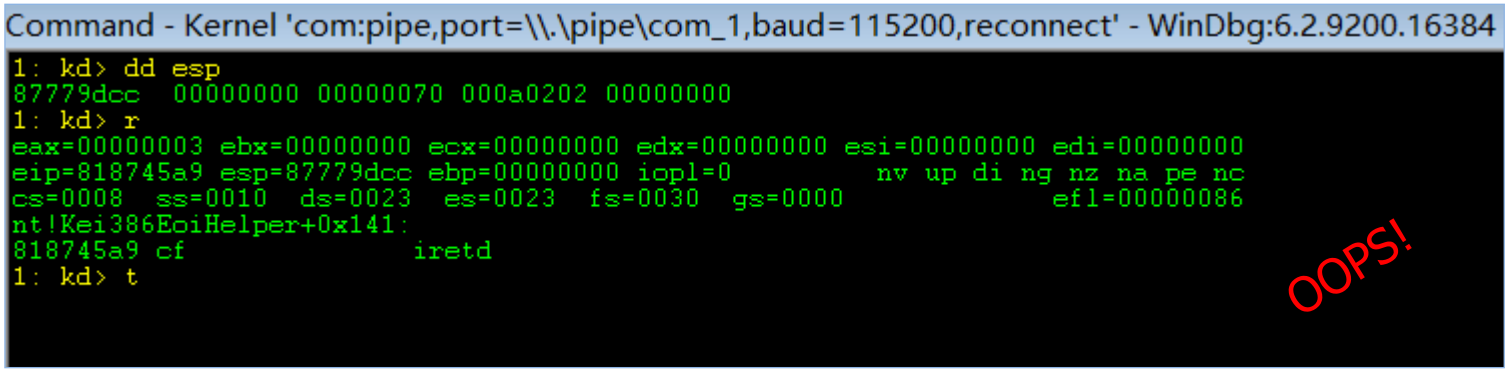
→

```
1: kd> dt _ktrap_frame 9be4bd64
nt!_KTRAP_FRAME
   +0x068 Eip              : 0x77bd9a94
   +0x06c SegCs            : 0x1b
   +0x070 EFlags           : 0x246
```

Debuggers' Support

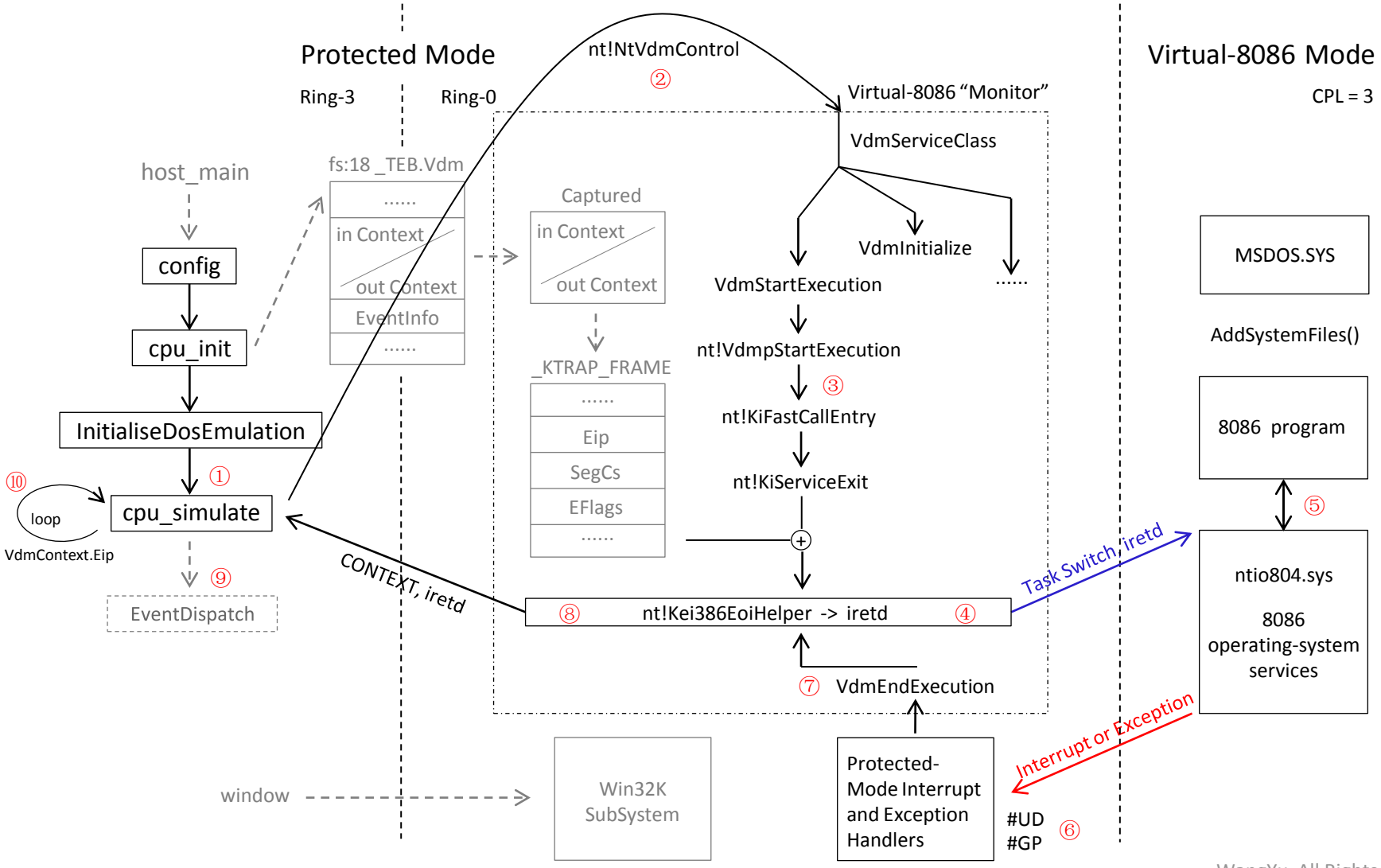






Missing SoftICE ?! ;-)

Let's Review the Simulator Work Flow



Part Three

The Story of NtVDM Subsystem's Vulnerability

Where is the ELSE ?

What do you think of the (old) code's quality ? ;-)

WRK : base\ntos\ps\i386\psvdm.c (Line:1551)
from Windows NT-4.0 to Windows Blue RC

```
NTSTATUS
Psp386CreateVdmIoListHead(
    IN PEPROCESS Process
    )
{
    PVDM_PROCESS_OBJECTS pVdmObjects = Process->VdmObjects;
    NTSTATUS Status;
    PVDM_IO_LISTHEAD HandlerListHead=NULL;
    KIRQL    OldIrql;
    PAGED_CODE();

    Status = STATUS_SUCCESS;

    // if there isn't yet a head, grab the resource lock and create one
    if (pVdmObjects->VdmIoListHead == NULL) {
        KeRaiseIrql(APC_LEVEL, &OldIrql);
        ExAcquireResourceExclusiveLite(&VdmIoListCreationResource, TRUE);

        // if no head was created while we grabbed the spin lock
        if (pVdmObjects->VdmIoListHead == NULL) {

            ......

        }
    }
    return STATUS_SUCCESS;
}
```

don't forget me!
Please!

CVE-2004-0208 (nt!KiTrap06 / #UD)

Working out the details, however, is left as an exercise for the reader.

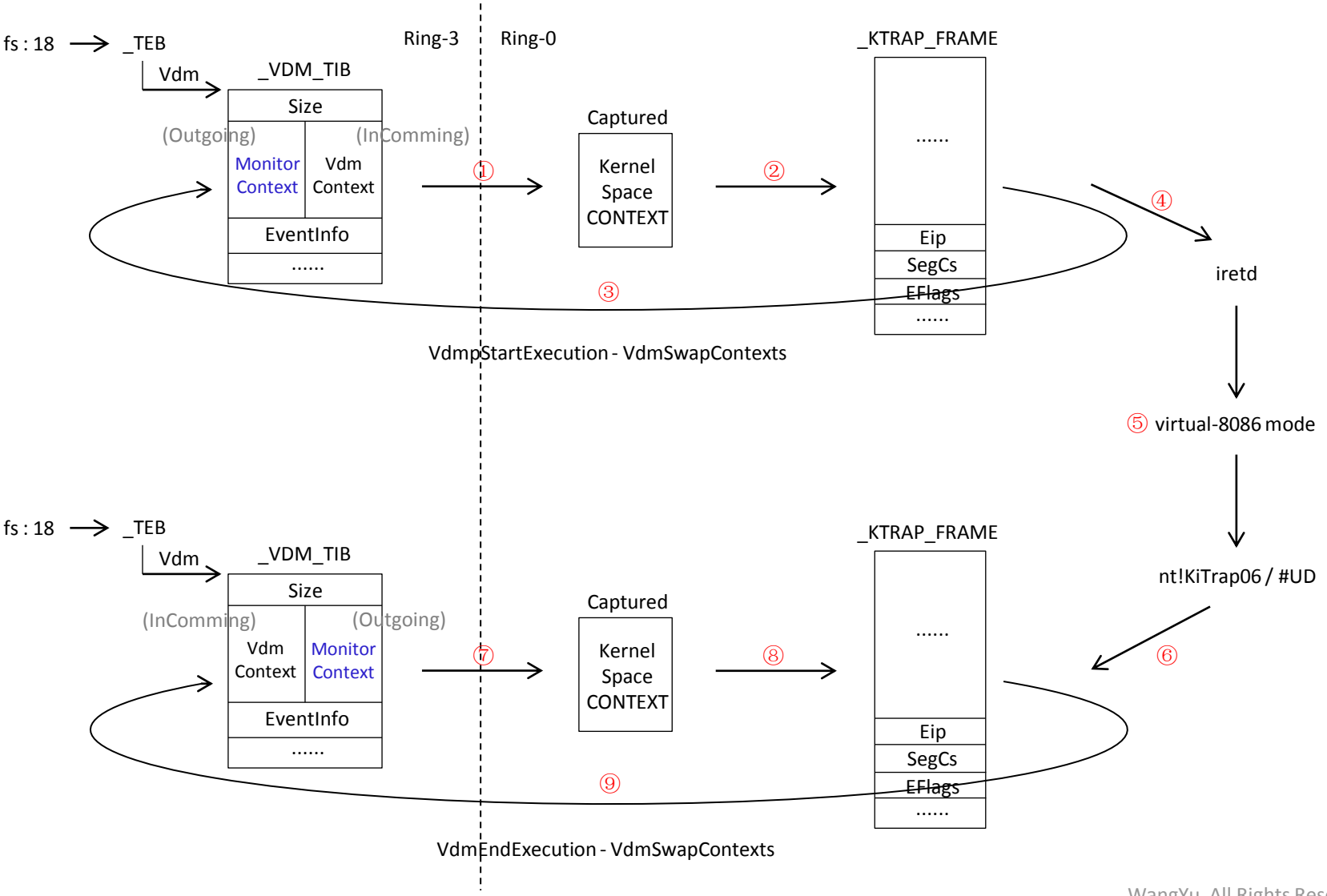
Just kidding. — Derek Soeder

CVE-2004-0208 / AD20041012
Windows VDM #UD Local Privilege Escalation

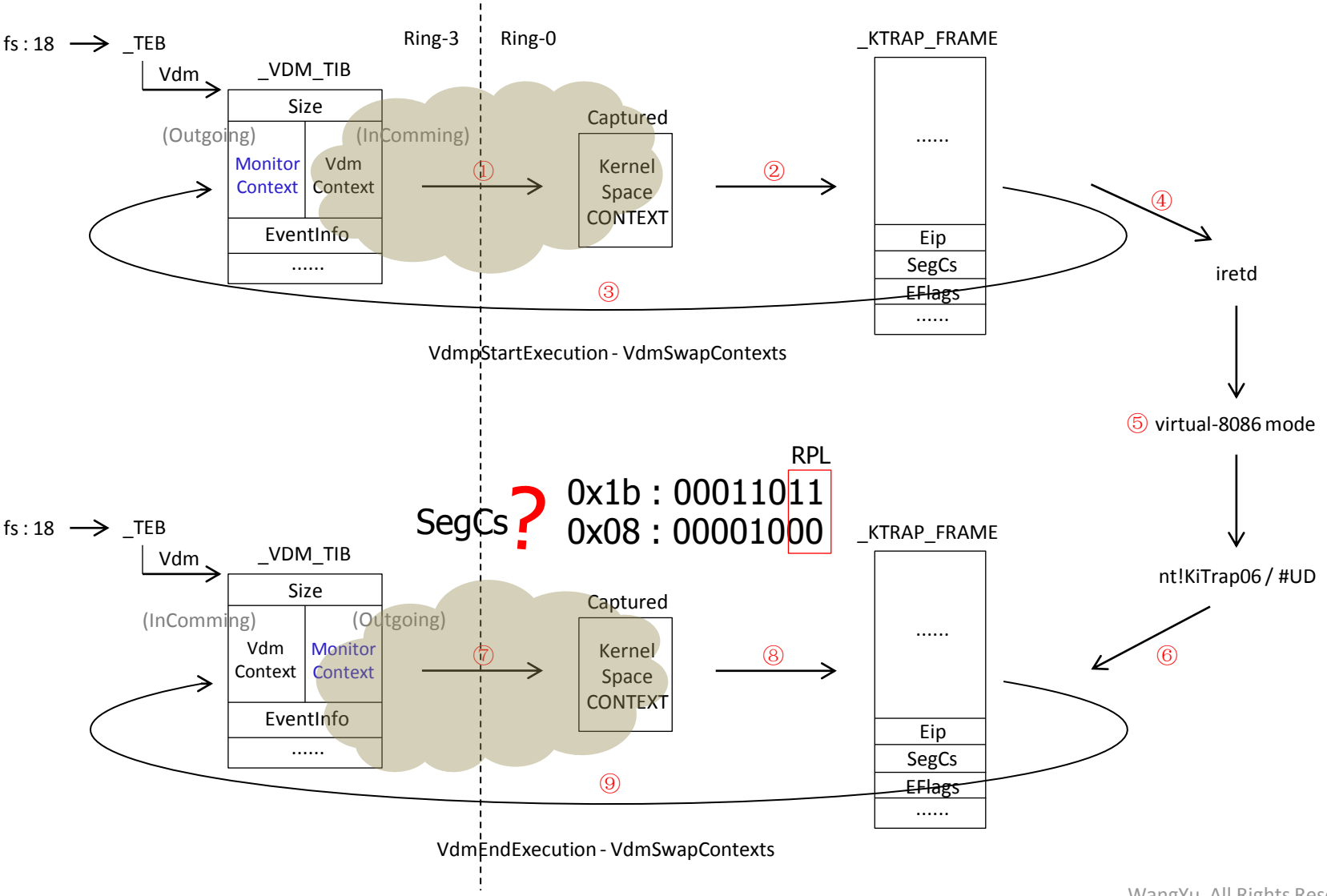
"(Outgoing) context is contained in user memory but is not sanitized in any way by the #UD handler, so any process with or without a formally-initialized VDM can place arbitrary values in the host execution context and get the handler to IRETD to any CS:EIP, allowing kernel privileges to be retained while user-supplied code is executed."

eEye Digital Security, 2004.

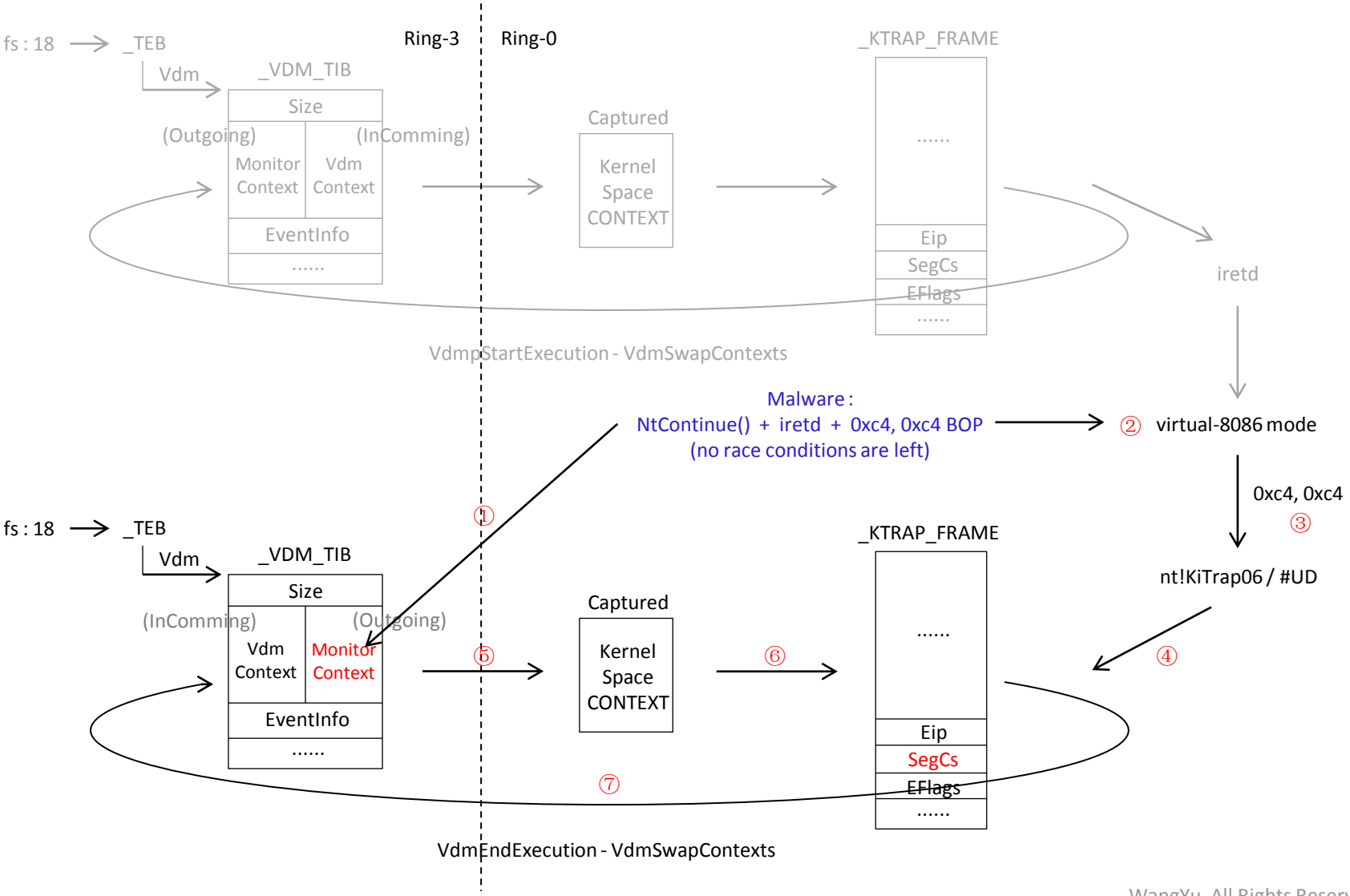
CVE-2004-0208 (nt!KiTrap06 / #UD)



CVE-2004-0208 (nt!KiTrap06 / #UD)



CVE-2004-0208 (nt!KiTrap06 / #UD)



CVE-2004-0208 (nt!KiTrap06 / #UD)

POC && Mitigation

```
FAST_V86_TRAP_6  MACRO  (Line:515)
...
;
; Load Monitor context
;
add   eax, VtMonitorContext - VtVdmContext ; (eax)->monitor context
mov   ebx, [eax].CsSegSs
mov   esi, [eax].CsEsp
mov   edi, [eax].CsEFlags
mov   edx, [eax].CsSegCs
mov   ecx, [eax].CsEip
sub   esp, 20
mov   [esp + 16], ebx
mov   [esp + 12], esi
mov   [esp + 8], edi
mov   [esp + 4], edx
mov   [esp + 0], ecx
mov   ebx, [eax].CsEbx
mov   esi, [eax].CsEsi
mov   edi, [eax].CsEdi
mov   ebp, [eax].CsEbp
...
iretd


kd> p
nt!KiTrap06+0x1cf:
80467039 cf              iretd
kd> dd esp
f82a7d98  016c0000 00000008 00000202 0105fc40
```

```
TrapFrame->Eip = InContext->Eip;
EFlags = InContext->EFlags;

if ( !(EFlags & 0x20000) )
{
  TrapFrame->SegCs |= 3u;
  SegCs = TrapFrame->SegCs;
  TrapFrame->HardwareSegSs |= 3u;

  if ( SegCs < 8 )
    TrapFrame->SegCs = 0x1Bu;
}
VdmSwapContexts:60
```

CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)

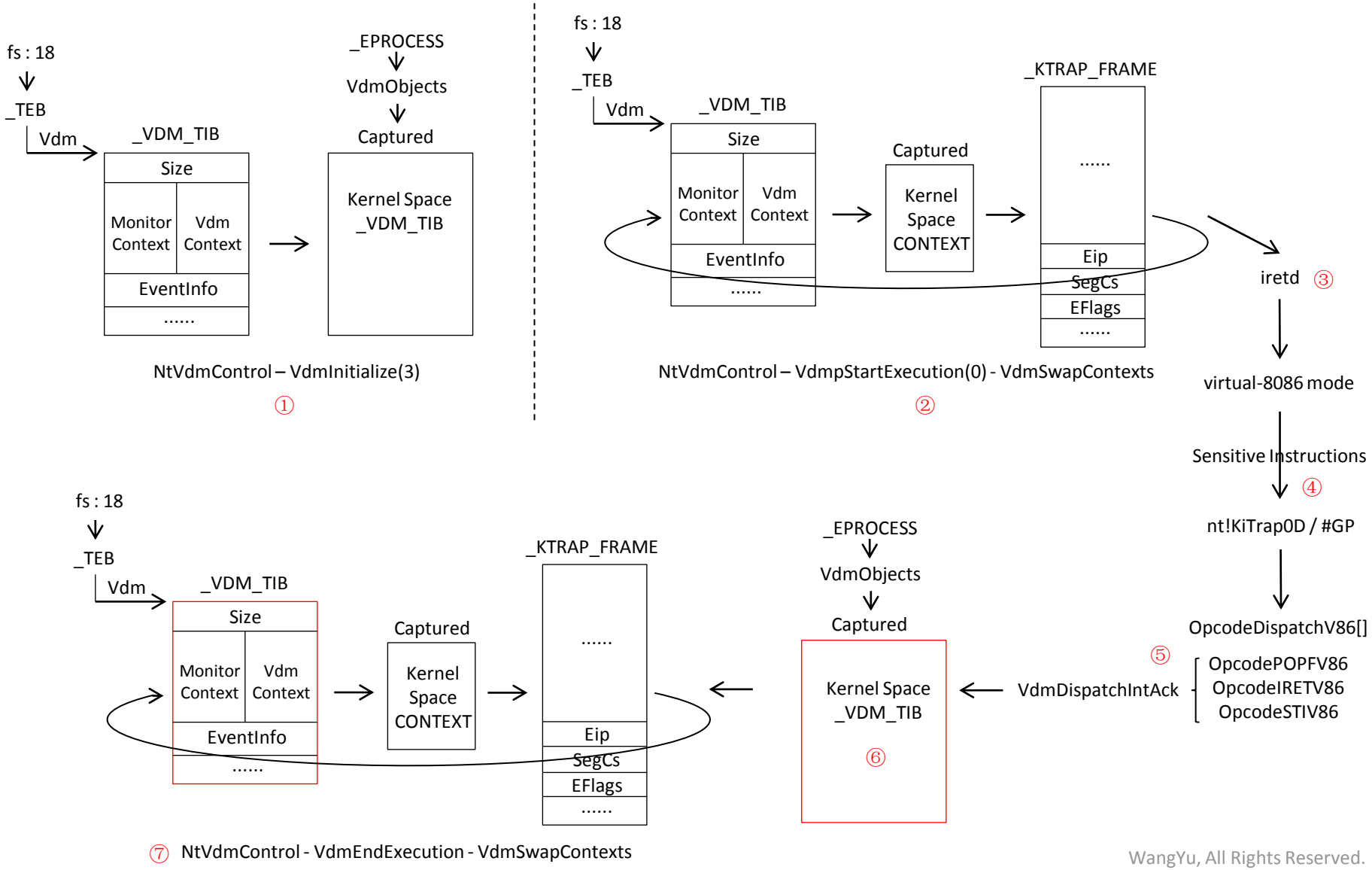
CVE-2004-0118 / AD20040413E
Windows VDM TIB Local Privilege Escalation

"The problem lies in a certain area of the Windows kernel that supports 16-bit code executing in a Virtual DOS Machine (VDM).

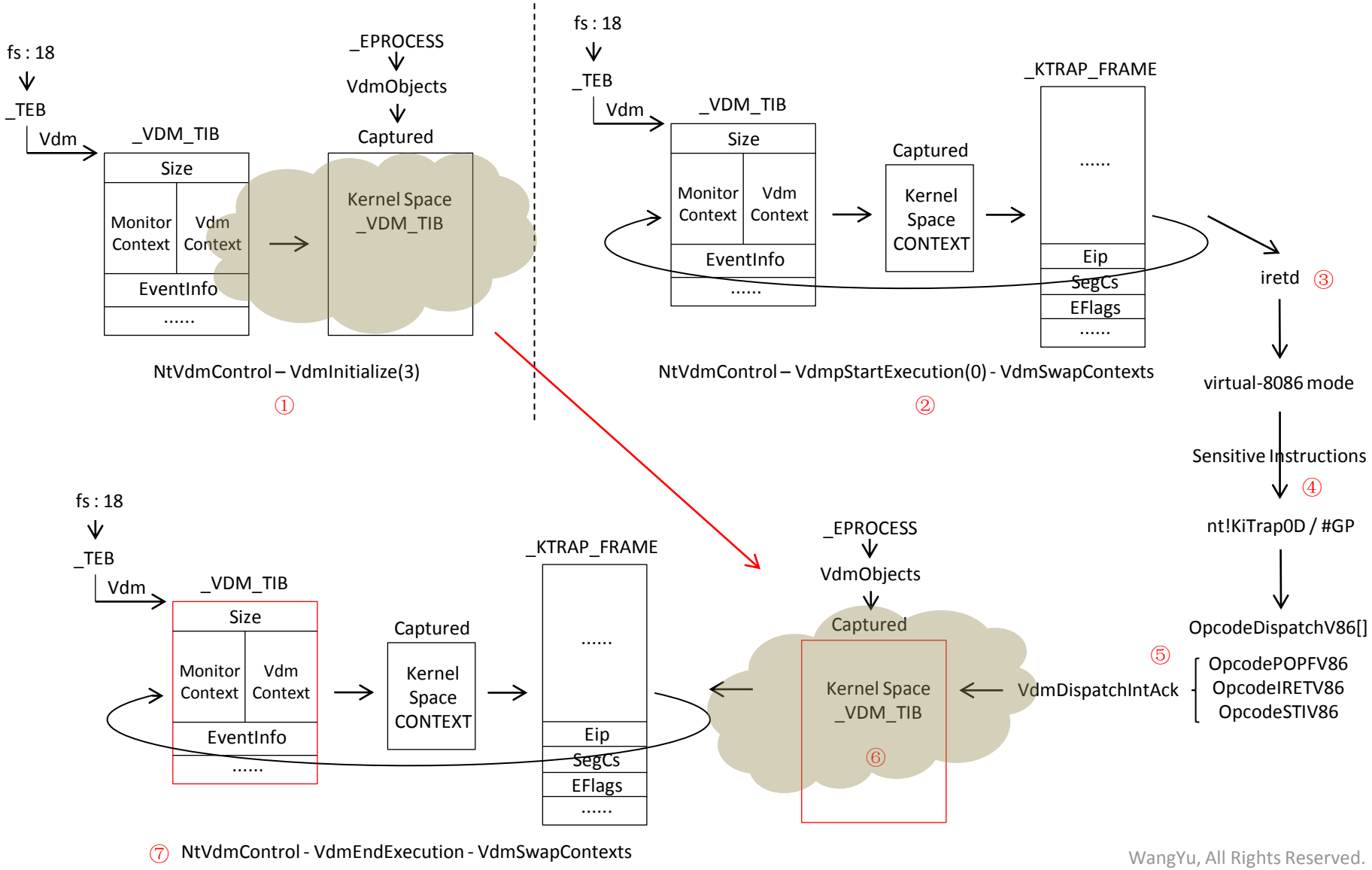
By causing the processor to execute code in Virtual86 (essentially "16-bit emulation") mode without first initializing a VDM for the process, specific routines in the Windows 2000 kernel code may be caused to dereference a null pointer, which actually functions as a pointer to attacker-controlled data if memory is allocated at virtual address 0."

eEye Digital Security, 2004.

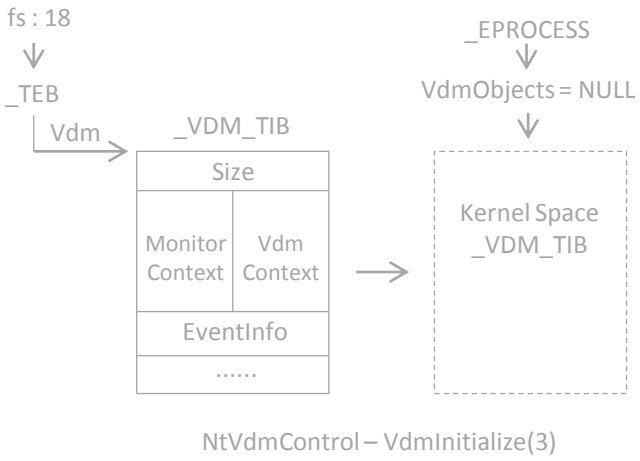
CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)



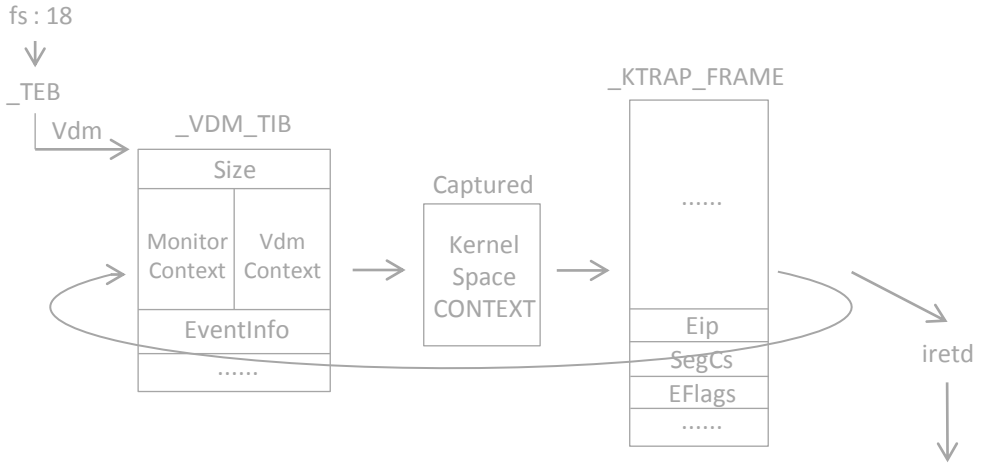
CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)



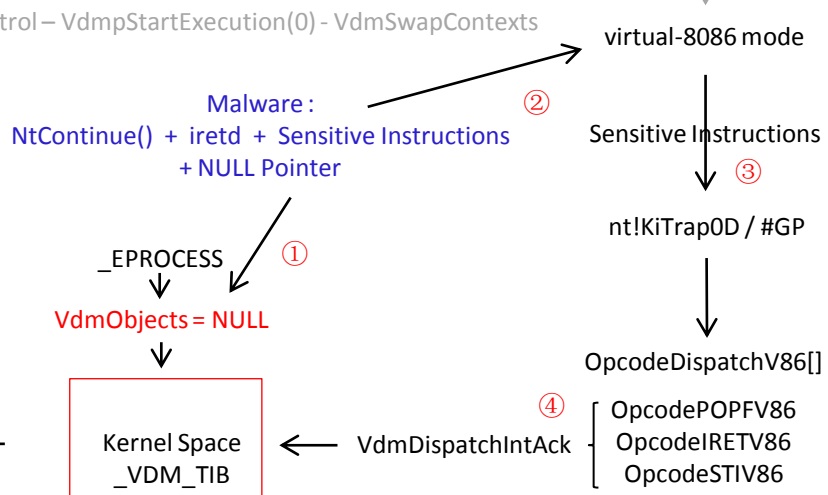
CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)

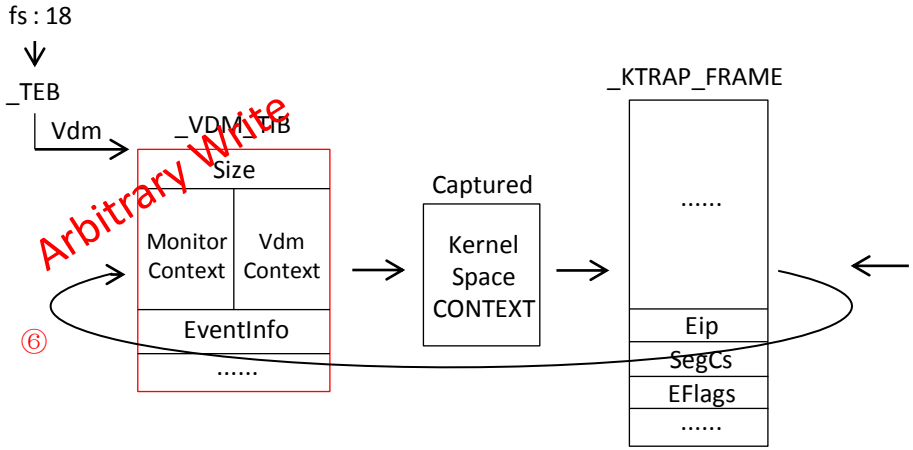


NtVdmControl – VdmInitialize(3)



NtVdmControl – VdmpStartExecution(0) - VdmSwapContexts





NtVdmControl - VdmEndExecution - VdmSwapContexts

Sensitive Instructions

20.2.7 Sensitive Instructions

When an IA-32 processor is running in virtual-8086 mode, the CLI, STI, PUSHF, POPF, INT *n*, and IRET instructions are sensitive to IOPL.
The IN, INS, OUT, and OUTS instructions, which are sensitive to IOPL in protected mode, are not sensitive in virtual-8086 mode.

Intel IA-32 Architectures SDM Vol.3B 20-10

```
FAST_V86_TRAP_D  MACRO  (Line:761)
...
mov   eax, [esp].TsSegCs      ; (eax) = H/W Cs
shl   eax,4
add   eax,[esp].TsEip         ; (eax) -> flat faulted addr
xor   edx, edx
mov   ecx, ss:[eax]           ; (ecx) = faulted instruction
mov   dl, cl
mov   dl, ss:OpcodeIndex[edx] ; (edx) =  opcode index
jmp   ss:V86DispatchTable[edx * type V86DispatchTable]
...

; V86DispatchTable - table of routines used to
; emulate instructions in v86 mode.

dtBEGIN V86DispatchTable,V86PassThrough
        dtS    VDM_INDEX_PUSHF           , V86Pushf
        dtS    VDM_INDEX_POPF            , V86Popf
        dtS    VDM_INDEX_INTnn           , V86Intnn
        dtS    VDM_INDEX_IRET            , V86Iret
        dtS    VDM_INDEX_CLI             , V86Cli
        dtS    VDM_INDEX_STI             , V86Sti
dtEND   MAX_VDM_INDEX
```

CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)

POC && Mitigation

```
VdmDispatchIntAck proc  (Line:1551)

mov     eax,_VdmFixedStateLinear
test    [eax],VDM_INT_HARDWARE
mov     eax,PCR[PcPrcbData+PbCurrentThread]
mov     eax,[eax]+ThApcState+AsProcess
mov     eax,[eax].EpVdmObjects
mov     eax,[eax].VpVdmTib    ; get pointer to VdmTib
jz      short dia20

...

;
; Switch to monitor context
;

mov     dword ptr [eax].VtEIEvent,VdmIntAck
mov     dword ptr [eax].VtEIInstSize,0
mov     dword ptr [eax].VtEiIntAckInfo,0
stdCall _VdmEndExecution, <ebp, eax>
jmp short dia10

...
```

```
VdmTib = *(_Vdm_Tib **)(__readfsdword(24) + 0xF18);

if ( v714 & 1 )  // *VdmFixedStateLinear & VDM_INT_HARDWARE
{
  if ( (unsigned int)VdmTib < (unsigned int)MmUserProbeAddress )
    VdmDispatchInterrupts(TrapFrame, VdmTib);
}
else
{
  if ( (unsigned int)VdmTib < (unsigned int)MmUserProbeAddress )
  {
    VdmTib->EventInfo.Event = 3;
    VdmTib->EventInfo.InstructionSize = 0;
    *(_DWORD *)&VdmTib->EventInfo.___u3.IoInfo.PortNumber = 0;

    VdmEndExecution(TrapFrame, VdmTib);
  }
}
```

VdmDispatchIntAck:22

CVE-2010-0232 (nt!KiTrap0D / #GP)

Working out the details of the attack is left as an exercise for the reader.

Just kidding, that was an homage to Derek Soeder :-)

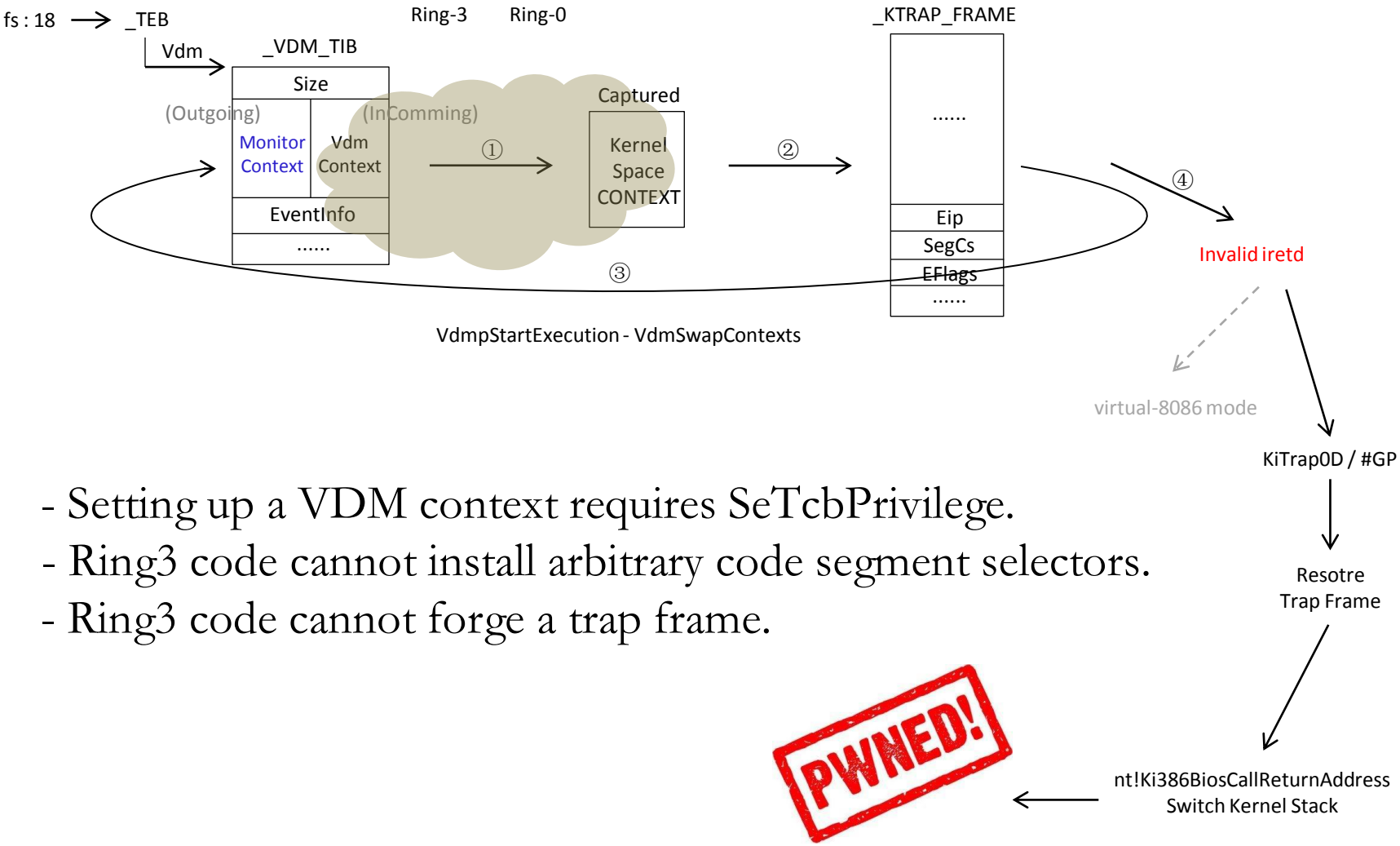
— Tavis Ormandy

CVE-2010-0232
Microsoft Windows NT #GP Trap Handler
Allows Users to Switch Kernel Stack
Best Privilege Escalation Bug, 2010.

"It's one of those rare, but fascinating design-level errors dealing with low-level system internals. Its exploitation requires skills and ingenuity.

what is important is that the two stages must be perfectly synchronised, if the kernel transitions to the second stage incorrectly, a hostile user can take advantage of this confusion to take control of the kernel and compromise the system."

CVE-2010-0232 (nt!KiTrap0D / #GP)



- Setting up a VDM context requires SeTcbPrivilege.
- Ring3 code cannot install arbitrary code segment selectors.
- Ring3 code cannot forge a trap frame.

CVE-2010-0232 (nt!KiTrap0D / #GP)

```
1: kd> p
nt!KiSystemCallExit:
80541710 cf              iretd
1: kd> dd esp
f519bdcc  80502903 0000000b 00000300 56565656
```

①

```
nt!KiTrap0D:
1: kd> dt _KTRAP_FRAME f519bd58
nt!_KTRAP_FRAME
   +0x030 SegGs            : 0
   +0x034 SegEs            : 0x23
   +0x038 SegDs            : 0x23
   +0x03c Edx              : 0x56565656
   +0x040 Ecx              : 0x56565656
   +0x044 Eax              : 0x56565656
   +0x050 SegFs            : 0x30
   +0x054 Edi              : 0x56565656
   +0x058 Esi              : 0x0217e814
   +0x05c Ebx              : 0x56565656
   +0x060 Ebp              : 0x56565656
   +0x064 ErrCode          : 8
   +0x068 Eip              : 0x80541710
   +0x06c SegCs            : 8
   +0x070 EFlags           : 0x10002
   +0x074 HardwareEsp      : 0x80502903
   +0x078 HardwareSegSs    : 0xb
```

②

```
trap.asm:
mov   eax, OFFSET FLAT:Ki386BiosCallReturnAddress
cmp   eax, [edx]                    ; [edx]= trapped eip
                                    ; Is eip what we're expecting?
jne   short Kt0d0005                ; No, continue

mov   eax, [edx]+4                  ; (eax) = trapped cs
cmp   ax, KGDT_R0_CODE OR RPL_MASK  ; Is Cs what we're exptecting?
jne   short Kt0d0005                ; No

jmp   Ki386BiosCallReturnAddress    ; with interrupts off
```

③

```
1: kd> r
eax=0000000b ebx=56565656 ecx=56560008 edx=f519bdcc esi=0217e814 edi=80541710
eip=80502903 esp=f519bd58 ebp=f519bd58 iopl=0         nv up di pl zr na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000046
nt!Ki386BiosCallReturnAddress:
80502903 64a11c000000    mov     eax,dword ptr fs:[0000001Ch]
```

④

```
nt!Ki386BiosCallReturnAddress+0x18:
8050291b 8b6558          mov     esp,dword ptr [ebp+58h]
8050291e 83c404          add     esp,4
```

⑤

```
1: kd> r
eax=f78e8d70 ebx=4b4b4b4b ecx=815164e8 edx=00007ffd esi=4b4b4b4b edi=4b4b4b4b
eip=8050295b esp=0217e830 ebp=4b4b4b4b iopl=0         nv up ei pl nz na po cy
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000203
nt!Ki386BiosCallReturnAddress+0x58:
8050295b c20400          ret     4
```

CVE-2010-0232 (nt!KiTrap0D / #GP)

Think:

Please compare similarities with CVE-2010-0232 and CVE-2012-0217

CVE-2012-0217 : Intel's sysret Kernel Privilege Escalation

Non-canonical Address -> sysret -> #GP (with User Stack)

Invalid CS:IP -> iret -> #GP (with User Stack)

CVE-2012-2553 (ppi->pwpi Pointer Overwrite)

CVE-2012-2553
Windows Kernel VDM Use-After-Free

"As you can imagine, the code quality of the vulnerable routine wasn't (perhaps still isn't) exceptionally good; it assumed that a process would only call it once during its entire lifespan ..."

```
xxxRegisterUserHungAppHandlers:

//
//  if sucess then intialize the pwpi, ppi structs
//  else free allocated memory
//
if (bRetVal) {
    pwpi->hEventWowExecClient = hEventWowExec;
    pwpi->lpfnWowExitTask = (DWORD)pfnW32EndTask;
    ppi = PpiCurrent();
    ppi->pwpi = pwpi;

    // add to the list, order doesn't matter
    pwpi->pwpiNext = gpwpiFirstWow;
    gpwpiFirstWow  = pwpi;
    }

.....
```

```
xxxInitTask, Windows NT-4.0:

/*
* Alloc and Link in new task into the task list
*/
if ((ptdb = (PTDB)UserAllocPoolWithQuota(
                        sizeof(TDB), TAG_WOW)) == NULL)
        return STATUS_NO_MEMORY;

pti->ptdb = ptdb;

......
```

CVE-2012-2553 (ppi->pwpi Pointer Overwrite)

Mitigation

```
if ( *(_DWORD *)(v2 + 0xB0)
  || ZwQueryInformationProcess((HANDLE)0xFFFFFFFF,
  || !ProcessInformation )
{
  result = 0;
}
else
{
  pwpi = ExAllocatePoolWithQuotaTag((POOL_TYPE)41,
  result = 0;
  if ( pwpi )
  {
    memset(pwpi, 0, 0x28u);
    v5 = 1;
    v6 = ObReferenceObjectByHandle(Handle, 0x1F000
    *((_DWORD *)pwpi + 4) = Object;
    if ( v6 < 0 )
    {
      v5 = 0;
      ExFreePoolWithTag(pwpi, 0);
    }
    else
    {
      *((_DWORD *)pwpi + 5) = Handle;
      *((_DWORD *)pwpi + 3) = a1;
      *(_DWORD *)(v2 + 0xB0) = pwpi;
      *(_DWORD *)pwpi = gpwpiFirstWow;
```

xxxRegisterUserHungAppHandlers:11

Efficiency ?

```
int __stdcall NtUserCallHwnd(int a1, int a2)
{
  int v2; // eax@1
  int v3; // esi@3
  int v5; // [sp+4h] [bp-Ch]@2
  int v6; // [sp+8h] [bp-8h]@2

  gptiCurrent = (struct tagBWL *)ExEnterPriorityRegionAndAcquireResourceExclusive(gpresUser);
  gbValidateHandleForIL = 1;
  v2 = ValidateHwndEx(a1, 1);
  if ( v2 )
  {
    v5 = *((_DWORD *)gptiCurrent + 48);
    *((_DWORD *)gptiCurrent + 48) = &v5;
    v6 = v2;
    ++**(_DWORD *)(v2 + 4);
    if ( (unsigned int)(a2 - 77) > 4 )
      v3 = 0;
    else
      v3 = ((int (__stdcall *)(int))apfnSimpleCall[a2])(v2);
```

Never Stop Exploring !

CVE-2007-1206, Derek Soeder

CVE-2010-3941, Tarjei Mandt

CVE-2013-3196, j00ru

CVE-2013-3197, j00ru

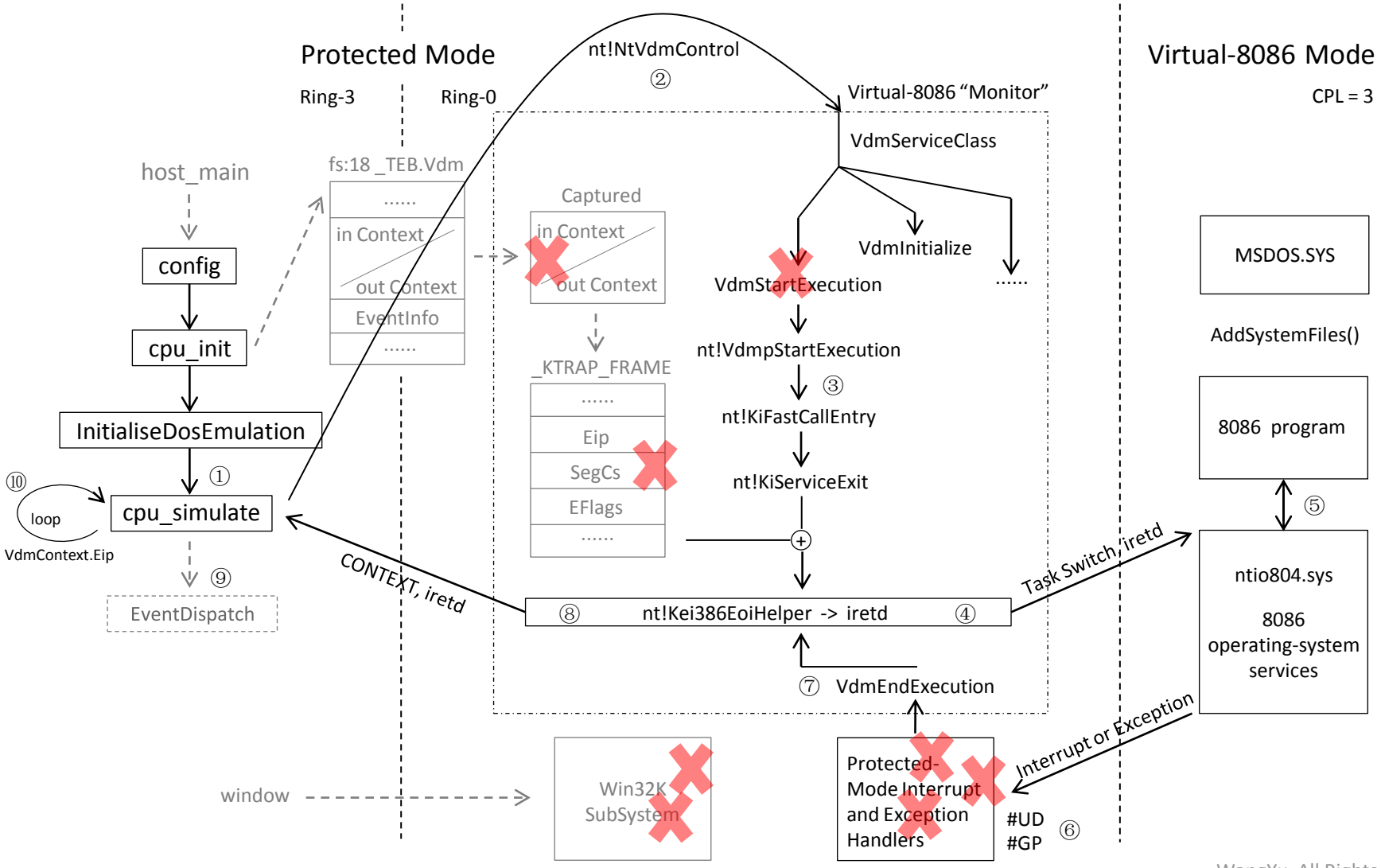
CVE-2013-3198, j00ru

……

Part Four

The Lessons We Have Learnt

The Lessons We Have Learnt



The Lessons We Have Learnt

From aspect of a Architect :

All inputs are harmful, protect your code boundaries

From aspect of a attacker :

Focus on the accumulation of knowledge

From aspect of a defender :

No fears, you have nothing to lose

Welcome to the Cyberwar Arms Race.

— Bruce Schneier

Thanks !

P1P1Winner PJF Bugvuln Royce Lu

MJ0011 Yajin Zhou Xuxian Jiang Prof.

360 Wireless Security Research Institute

360 Safe Team SyScan Fan.Tuan

Q&A

wangyu@360.cn