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."
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

Real-Address Mode

Real Mode Code

Protected Mode

Protected-Mode Task

Win32K SubSystem

Virtual-8086 Mode

Virtual-8086 Mode Tasks (8086 Programs)

CPL = 3

Protected-Mode Task

NtVDM.EXE

cpu_simulate() loop

nt!NtVdmControl

CONTEXT, iretd

VdmEndExecution()

VdmStartExecution()

Protected-Mode Interrupt and Exception Handlers

Protected-Mode Task

Task Switch, iretd

EFLAGS.VM = 0

EFLAGS.VM = 1

CALL

RET

PE = 1

PE = 0 or RESET

PE = 0

or RESET

RESET

virtual-8086 “Monitor”

hardware interrupt or exception; INT n when IOPL is 3

general-protection exception caused by software interrupt

IRETD, normal return from interrupt or exception handler

Redirect Interrupt to 8086 Program

Interrupt or Exception Handler

Reference: Figure 20-3. Entering and Leaving Virtual-8086 Mode
Simulator's User Mode — NtVDM Process Logic

NtVDM Process Sequence Overview

- `ntvdm!main`
  - TimerInit
  - CpuEnvInit
  - nls_init
  - `host_main`
    - host_applInit
    - gfi_init
    - mouse_driver_initialisation
    - `config`
    - `cpu_init`
    - setup_vga_globals
    - nt_init_screen
    - `InitialiseDosEmulation`
    - `host_start_cpu / cpu_simulate`
    - host_applClose

- [\mvdm\softpc.new\obj.vdm\ntvdm.c ]
- [\mvdm\softpc.new\base\support\main.c ]
- [\mvdm\softpc.new\host\src\nt_reset.c ]
- [\mvdm\softpc.new\host\src\config.c ]
- [\mvdm\v86\monitor\i386\monitor.c ]
- [\mvdm\sofcpc.new\host\src\nt_msscs.c ]
- [\mvdm\v86\monitor\i386\monitor.c ]
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 information
      - -> 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"
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

- **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;
    }
}```
From User Mode Aspect to View Working Flow

Protected Mode

Ring-3

Ring-0

Virtual-8086 Mode

CPL = 3

host_main

config

cpu_init

InitialiseDosEmulation

Virtual-8086 “Monitor”

EventDispatch

EventInfo

in Context

out Context

Protected-Mode Interrupt and Exception Handlers

nt!NtvdmControl

nto804.sys

8086 operating-system services

AddSystemFiles()

MSDOS.SYS

8086 program

Task Switch, iretd

Interrupt or Exception

8086 operating-system services

AddSystemFiles()
Let's Think Different

Virtual-8086 "Monitor"

Protected-Mode Interrupt and Exception Handlers

Task Switch, iretd

NtVDM.EXE

8086 program

8086 operating-system services

config

cpu_init

InitialiseDosEmulation

cpu_simulate

nt!NTvdmControl

CONTEXT, iretd

EventDispatch

Virtual-8086 Mode

Protected Mode

EFLAGS.VM = 0

EFLAGS.VM = 1

CPL = 3

Ring-3

Ring-0

Interrupt Exception

Virtual-8086 Mode

Protected Mode

4 GB

0 GB

Win32K SubSystem

EventInfo in Context

out Context

......

fs:18_TEB.Vdm

......
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

<table>
<thead>
<tr>
<th>Command - Kernel 'com:pipe,port=\pipe\com_1,baud=115200'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: kd&gt; bp nt!EiTrap06 &quot;db (poi(esp+4)&lt;&lt;4)+poi(esp) l4:q&quot;</td>
</tr>
<tr>
<td>0: kd&gt; g</td>
</tr>
<tr>
<td>000024cd c4 c4 09 58</td>
</tr>
<tr>
<td>00011c2a c4 c4 58 01</td>
</tr>
<tr>
<td>00032c9 c4 c4 52 00</td>
</tr>
<tr>
<td>0008bf4 c4 c4 57 0f</td>
</tr>
<tr>
<td>001039f3 c4 c4 50 3c</td>
</tr>
<tr>
<td>001002eb c4 c4 17 5a</td>
</tr>
<tr>
<td>0009e24e c4 c4 54 01</td>
</tr>
<tr>
<td>00000b27 c4 c4 5e 33</td>
</tr>
<tr>
<td>00000bac c4 c4 50 11</td>
</tr>
<tr>
<td>00001626 c4 c4 12 b1</td>
</tr>
<tr>
<td>0008e2ac c4 c4 50 3b</td>
</tr>
<tr>
<td>0009b18c c4 c4 50 0f</td>
</tr>
<tr>
<td>0009b1d2 c4 c4 50 1b</td>
</tr>
<tr>
<td>0009b1de c4 c4 50 32</td>
</tr>
<tr>
<td>0009b1eb c4 c4 54 05</td>
</tr>
<tr>
<td>0009b2d6 c4 c4 50 46</td>
</tr>
<tr>
<td>0009b513 c4 c4 50 4a</td>
</tr>
<tr>
<td>0008e3a5 c4 c4 15 26</td>
</tr>
<tr>
<td>0008e3b7 c4 c4 50 0d</td>
</tr>
<tr>
<td>0009f18c c4 c4 50 21</td>
</tr>
<tr>
<td>000967bc c4 c4 50 1a</td>
</tr>
<tr>
<td>0008e482 c4 c4 54 0c</td>
</tr>
<tr>
<td>0009a30e c4 c4 50 12</td>
</tr>
</tbody>
</table>
Simulator's Kernel Mode — VdmpStartExecution

- Check the VdmAllowed / VdmObjects fields in EPROCESS Structure
- Check _VDM_TIB parameter by calling VdmpGetVdmTib routine
- Reserve 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

After

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 : 0x77bd9a94

       +0x074 HardwareEsp : 0x11ff718
       +0x078 HardwareSegSs : 0
Simulator's Kernel Mode — VdmEndExecution

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

- Reserve 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

\[
\begin{align*}
0: & \text{kd} > \text{dd esp} \\
& 9be4bdcc \quad 00000427 \quad 00000070 \quad 00b0246 \quad 00000700 \\
0: & \text{kd} > \text{db (70<<4)+427} \\
& 00000b27 \quad \text{c4 c4 3e 33 d2 8e da 8e-c2 33 c0 bf 34 05 ab} \quad \text{ab ..}^3 \ldots 3 \ldots 4 \ldots \\
& 00000b37 \quad \text{8c c8 c7 06 6c 00 85 01-a3 6e 00 c7 06 a4 00 54} \quad \ldots 1 \ldots n \ldots T
\end{align*}
\]

\[
\begin{align*}
1: & \text{kd} > \text{dt _ktrap_frame 9be4bd64} \\
\text{nt!KTRAP_FRAME} +0x068 \quad \text{Eip} : \ 0x427 \\
+0x06c \quad \text{SegCs} : \ 0x70 \\
+0x070 \quad \text{EFlags} : \ 0xb0246
\end{align*}
\]

\[
\begin{align*}
1: & \text{kd} > \text{dt _ktrap_frame 9be4bd64} \\
\text{nt!KTRAP_FRAME} +0x068 \quad \text{Eip} : \ 0x77bd9a94 \\
+0x06c \quad \text{SegCs} : \ 0x1b \\
+0x070 \quad \text{EFlags} : \ 0x246
\end{align*}
\]
Debuggers' Support

Missing SoftICE ?! ;-)
Let's Review the Simulator Work Flow

Protected Mode

Virtual-8086 Mode

Virtual-8086 "Monitor"

Captured

VdmServiceClass

VdmInitialize

VdmStartExecution

nt!VdmStartExecution

nt!KiFastCallEntry

nt!KiServiceExit

#UD

#GP

Win32K SubSystem

Protected-Mode Interrupt and Exception Handlers

#UD

#GP
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;
}
```
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."

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

Vdm

fs : 18 → _TEB

_VDM_TIB

<table>
<thead>
<tr>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdm Context</td>
</tr>
<tr>
<td>Monitor Context</td>
</tr>
<tr>
<td>EventInfo</td>
</tr>
<tr>
<td>......</td>
</tr>
</tbody>
</table>

Ring-3

_Captured

Kernel Space CONTEXT

Eip
SegCs
EFlags
......

_IRETD

virtual-8086 mode

nt!KiTrap06 / #UD

VmpStartExecution - VdmSwapContexts

VdmEndExecution - VdmSwapContexts

WangYu, All Rights Reserved.
CVE-2004-0208 (nt!KiTrap06 / #UD)

Malware:
NtContinue() + iretd + 0xc4, 0xc4 BOP  
(no race conditions are left)

Virtual-8086 mode

WangYu, All Rights Reserved.
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
"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."

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

NtVdmControl - VdmInitialize(3)

NtVdmControl - VdmStartExecution(0) - VdmSwapContexts

virtual-8086 mode

Sensitive Instructions

nt!KiTrap0D / #GP

OpcodeDispatchV86[]

OpcodePOPFV86 OpcodeIRETV86 OpcodeSTIV86
CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)

NtVdmControl – VdmInitialize(3)

NtVdmControl – VdmStartExecution(0) - VdmSwapContexts

fs : 18
↓
_TEB

_VDM_TIB

Size
Monitor Context Vdm Context
EventInfo
......

Kernel Space CONTEXT

Eip
SegCs
EFlags
......

Sensitive Instructions
OpcodeDispatchV86[]
OpcodePOPFV86
OpcodeIRETV86
OpcodeSTIV86

iretd

virtual-8086 mode

nt!KiTrap0D / #GP

①

②

③

④

⑤

⑥

⑦

NtVdmControl - VdmEndExecution - VdmSwapContexts

WangYu, All Rights Reserved.
CVE-2004-0118 (nt!VdmDispatchIntAck / #GP)

NtVdmControl – VdmInitialize(3)

NtVdmControl – VdmStartExecution(0) - VdmSwapContexts

Malware: NtContinue() + iretd + Sensitive Instructions + NULL Pointer

nt!KiTrap0D / #GP

OpcodeDispatchV86[]

OpcodePOPFV86

OpcodeIRETV86

OpcodeSTIVV86

NtVdmControl - VdmEndExecution - VdmSwapContexts

WangYu, All Rights Reserved.
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.

```
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[PCrCbData+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

...
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

"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."
- 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)

nt!KiSystemCallExit:

kd> p
80541710 cf iretd
dd esp

nt!KiTrap0D:

dd esp

trap.asm:

mov eax, OFFSET FLAT:Ki386BiosCallReturnAddress

cmp eax, [edx] ; [edx]= trapped eip

jne short Kt0d0005
  ; Is eip what we're expecting?

mov eax, [edx]+4 ; (eax) = trapped cs

jmp Ki386BiosCallReturnAddress ; with interrupts off
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)
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 ..."
Mitigation

Efficiency?

```c
if ( ppi )
{
    memset(ppi, 0, 640u);
    v5 = 1;
    v6 = ObReferenceObjectByHandle(Handle, 0xF0000000,
        ((_DWORD *)ppi + 4) = Object;
    if ( v6 < 0 )
    {
        v5 = 0;
        ExFreePoolWithTag(ppi, 0);
    }
    else
    {
        ((_DWORD *)ppi + 5) = Handle;
        ((_DWORD *)ppi + 6) = a1;
        ((_DWORD *)ppi + 8) = ppi;
        ((_DWORD *)ppi) = gpwipFirstWow;
        xxxRegisterUserHungAppHandlers();
    }
}
```
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

Protected Mode

Ring-0

Virtual-8086 Mode

CPL = 3

WangYu, All Rights Reserved.
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