Web: http://www.ImmortalDescendants.org
Author: [yAtEs]
Date: 19/Sept/00
Topic: Hooking Interrupts (VxD)
(Part II of III)
Level: Beginner

This is part 2 of my VxD tutorial set, this second essay will look at hooking Interrupts.

So why hook an interrupt service? Most programs if not all, call functions from interrupt services, we can alter the way these services respond and work, this allows us have control over certain functions and to let us trick programs.

Will we learn the basic syntax of hooking INTs and then work with an example, I have chosen the softice debug trick of INT 68h, we shall build a small program to test if softice is loaded using the INT 68 function, then build a VxD to trick
the interrupt into responding as if softice wasn't loaded.

ok first lets look at the basic syntax of commands we need to use, providing you know how to build a basic dynamic VxD, i.e. read my first tutorial ;d, then its simple pimple to add INT Hooking code.

The service we need to call to hook interrupts is Hook_V86_Int_Chain, I'll paste what it says in the Win98DDK help file.


```
;_____________________________________________________________________________

Hook_V86_Int_Chain
include vmm.inc

mov     eax, Interrupt          ; number of interrupt to hook
mov     esi, OFFSET32 HookProc  ; points to hook procedure
VMMCall Hook_V86_Int_Chain

jc      not_installed           ; carry flag set if procedure not installed
```

Installs a hook procedure that the system calls whenever the specified interrupt occurs. Virtual devices use this service to monitor software interrupts, and simulated hardware interrupts in V86 mode. Unlike Windows 3.1 in which this service was available only during initialization, Windows 95 allows V86 interrupt hooks to be installed after initialization is complete. Uses Flags.

Returns with the carry flag clear if successful, set otherwise.
Interrupt Number of the interrupt for which to install the hook procedure.
HookProc Address of the hook procedure. For more information about the hook procedure, see below. The system calls the hook procedure whenever the corresponding interrupt occurs, a virtual device calls the Simulate_Int service, or the system simulates a hardware interrupt. This means a hook procedure must make no assumptions about the origin of the interrupt.

The system calls the procedure as follows:

```
mov     eax, Interrupt      ; number of interrupt hooked
mov     ebx, VM             ; current VM handle
mov     ebp, OFFSET32 crs   ; points to a Client_Reg_Struc
call    [HookProc]

jc      pass_to_next        ; carry set if interrupt not serviced
```

The Interrupt parameter is the number of the current interrupt, the VM parameter is a handle identifying the current virtual machine, and the crs parameter points to a Client_Reg_Struc structure containing the register values of the current virtual machine. If the hook procedure services the interrupt, it must clear the carry flag to prevent the system from passing the interrupt to the next hook procedure.

Any number of virtual devices can install a hook procedure for a given interrupt. The system always calls the last hook procedure first. A hook procedure either services the interrupt or directs the system to pass the interrupt to the next hook procedure. If no hook procedure services the interrupt, the system reflects the interrupt to the virtual machine.

This service is recommended instead of hooking the V86 interrupt vector directly.

See Also

Set_V86_Int_Vector, Simulate_Int

Built on Tuesday, May 18, 1999
;_____

So from that you can see the two parameters its takes, one for the interrupt number and another is a pointer to your new procedure.

ok now to start working on our first example, first we must create a program to test if softice is installed or not using INT 68, here is how that works

```
    mov     ah,43h
    int     68h
    cmp     ax,0F386h
    jz      detect
```

EAX=00004300 and int 68 is called, if the interrupt returns 0F386h then softice IS installed, if doesn't return that value then softice is not installed.

So we should create this small exe first,..here is my TASM code.


```
;_____________.SoftICE.ASM._____________________________________________________
.486
locals
jumps

.Model Flat ,StdCall

Extrn  MessageBoxA:PROC
Extrn  exitprocess:PROC

.data

fbox  db   'INT68 Test',0
ftitle db  'SoftICE is not loaded',0
ftitle2 db 'SoftICE is loaded',0


.code

main:

    mov     ah,43h
    int     68h
    cmp     ax,0F386h
    jz      detect

call MessageBoxA,0,offset ftitle,offset fbox,0
jmp endprog

detect:
 call MessageBoxA,0,offset ftitle2,offset fbox,0


endprog:

push 0
call exitprocess

end main
;_____________________________________________________________________________
```

Now if you have softice loaded and providing you have no anti-softice patches or frogsice running, you will get a messagebox saying softice is installed

The time has come to create our VxD, we shall hook interrupt 68 and modify the return code for 4300. So what do we need to do? lets look at it in pseudocode

1 Set-up basic template with deviceio handling
2 On creation of VxD enable our hook
3 On deactivation of VxD disable our hook

The Hook procedure
==========

this couldn't be more simple, when Int 68 is called, it will execute this piece of code
afterwards, so we just check the value of AX and modify if necessary.

Here is my full source, which i shall explain afterwards

```
;_____________________________________________________________________________
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;                         ;;
;;;;;;;;;;;;;                               ;;
;;;;;;;;          INT 68h HOOK              ;;
;;;;;                  BOOM BOOM            ;;
;;;           i'm the old skool rocker      ;;
;;;;;                     [yAtEs]           ;;
;;;;;;;                                     ;;
;;;;;;;;;;;;                                ;;
;;;;;;;;;;;;;;;;;                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.386p
include vmm.inc
include vwin32.inc
include shell.inc

DECLARE_VIRTUAL_DEVICE FIRST,1,0, FIRST_Control,\
     UNDEFINED_DEVICE_ID, UNDEFINED_INIT_ORDER

Begin_control_dispatch FIRST
    Control_Dispatch Sys_Dynamic_Device_Init, HOOK68
    Control_Dispatch Sys_Dynamic_Device_Exit, UNHOOK68
    Control_Dispatch w32_DeviceIoControl, OnDeviceIoControl
End_control_dispatch FIRST


;_____________________________________________________________________________
VxD_LOCKED_CODE_SEG
;_____________________________________________________________________________

BeginProc OnDeviceIoControl
    assume esi:ptr DIOCParams
    .if [esi].dwIoControlCode==DIOC_Open
        xor eax,eax
    .endif
    ret
```


```
EndProc OnDeviceIoControl



BeginProc UNHOOK68
pushfd
pushad
  mov       eax, 68h
  mov       esi, offset32 Hook68New
  VMMCall   UnHook_V86_Int_Chain
popad
popfd
clc
ret
EndProc UNHOOK68



BeginProc HOOK68
pushfd
pushad
  mov       eax, 68h
  mov       esi, offset32 Hook68New
  VMMCall   Hook_V86_Int_Chain
popad
popfd
clc
ret
EndProc HOOK68

BeginProc Hook68New
pushfd
pushad
  ;int 3
 cmp ax,0F386h PUBLISHER NOTE: THIS IS AN ERROR; CONTEXT REGS SHOULD BE USED

 jne skip
 mov ax,43h
  skip:

popad
popfd
ret
EndProc Hook68New


;______________________________________________________________________________
VxD_LOCKED_CODE_ENDS
;______________________________________________________________________________
end

;______________________________________________________________________________
```

lets look at the code, most of it you should recognise

```
    Control_Dispatch Sys_Dynamic_Device_Init, HOOK68
    Control_Dispatch Sys_Dynamic_Device_Exit, UNHOOK68
```

here we have our control message handles, when the VxD is initialised
we goto the procedure HOOK86 which has been defined in a Locked segment

```
BeginProc HOOK68
pushfd
pushad
  mov      eax, 68h
  mov      esi, offset32 Hook68New
  VMMCall  Hook_V86_Int_Chain
popad
popfd
clc
ret
EndProc HOOK68
```

Pushfd is push all flags and Pushad is push all registers, this code says we are hooking int 68 and our new control procedure is called Hook68New, we then restore all the flags and registers and CLC, CLear Carry-flag, if there was an error hooking this interrupt then the carry-flag is set, we haven't put any error checking in there so we just clear the flag and presume everything is working ok.. :)

Now lets look at our new int 68 handle

```
BeginProc Hook68New
pushfd
pushad
  ;int 3
 cmp ax,0F386h
 jne skip
 mov ax,43h
  skip:

popad
popfd
ret
EndProc Hook68New
```

again straight forward, when an external program calls INT 68 this code is called afterwards, our code saves all flag and reg data, then checks AX for 0F386h, remember this is what is return if softice is installed, if that value is returned we move 43h back into the register, if not

we just jump over that code and restore our flag/reg data.

The other line in our control message handle was

```
    Control_Dispatch Sys_Dynamic_Device_Exit, UNHOOK68
```

when the VxD is shutdown we call the procedure UNHOOK68 which is as follows:-

```
BeginProc UNHOOK68
pushfd
pushad
  mov      eax, 68h
  mov      esi, offset32 Hook68New
  VMMCall  UnHook_V86_Int_Chain
popad
popfd
clc
ret
EndProc UNHOOK68
```

Exactly the same as the hooking procedure, but we use UnHook_V86_Int_Chain.

that's a fair overview of the code i reckon ;), if you use the template from the last tutorial and make a new drawer and copy the stuff over, have your VxD name as First and then compile it, your DEF file and everything else should match up.

Ok so now,..run your SoftIce.exe.. says 'Softice is installed' now using your old dynamic VxD loader(loader.exe) use it to run your new VxD, A messagebox pops up saying "Loaded", DON'T click ok, now rerun Softice.exe, it says "Softice is NOT installed" woo ;) so our VxD is workin now you may click OK on the Loader.. in the background our VxD initialises the shutdown procedure, now rerun Softice.exe and you'll find Softice is installed again :) .

Other things to do
==========

if you want to see what's happening behind the scenes we can add a debug line, if you go back
to this code in our VxD

```
BeginProc Hook68New
pushfd
pushad
  ;int 3
 cmp ax,0F386h
 jne skip
 mov ax,43h
  skip:

popad
popfd
ret
EndProc Hook68New
```

this is what is called when another program calls INT 68, where the line ;int 3 is remove the ;
so we have:

```
BeginProc Hook68New
pushfd
pushad
  int 3
 cmp ax,0F386h
 jne skip
 mov ax,43h
  skip:

popad
popfd
ret
EndProc Hook68New
```

recompile your VxD. Now in softice type 'I3HERE ON' and exit, load your vxd up and run softice.exe when ur test program executes int68 softice will break and you can trace your own code and see what is happening, or... load symbol loader with Softice.exe and trace from entry point, and when you trace over, INT 68, you will see the VxD code kick in.


```
thats it till next time :), which is..um..ah.. API Hooking ;d
l8rs.

[yAtEs]
"Keep it locked, keep it hardcore. Roots 'n' phuture. Peace."
```