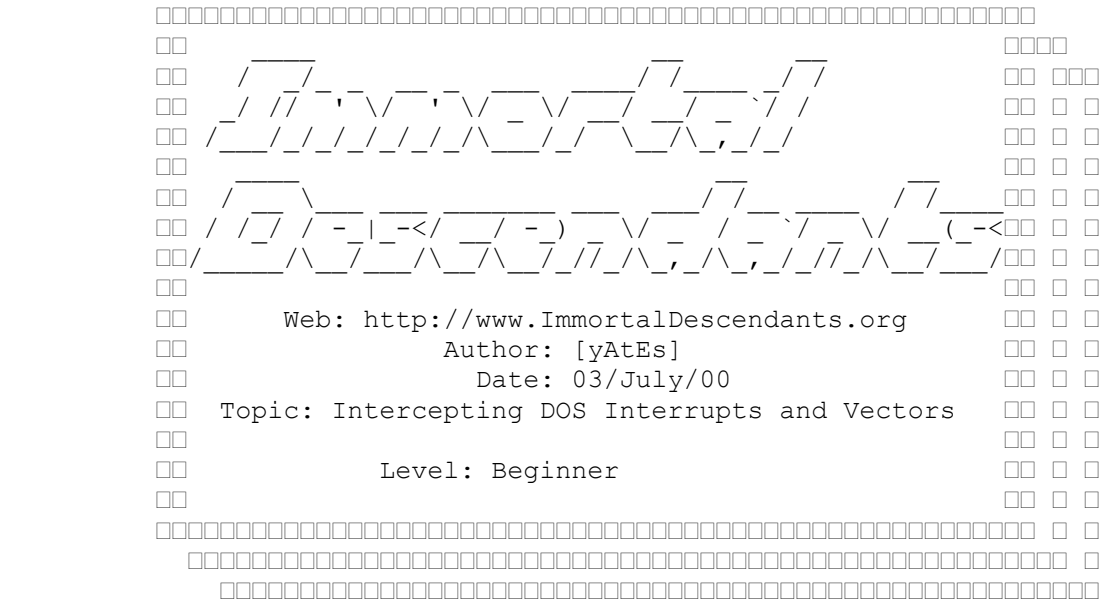


```
*****************************************************************
*
* Publishers Note: the whole package can be found at:
* http://www.reverse-engineering.info/files/DosINT.zip
*
*****************************************************************
```

back to old skool have you ever wanted to create programs
to monitor dos services/rom bios routines, perhaps a keylogger
or even a mini debugger, you can hook DOS services easily enough
and add your own code.

Each interrupts has a number such as Int 10h when you call this
the programs execution is interrupted and diverted to a set of
parameter functions in memory, the addresses for the interrupts
are stored in a vector table in the lower part of memory at 0000:0000

The vector table consists of two words for each interrupt, IP:CS,
so Int 0h's IP and CS are at 0000:0000, Int 01h is at 0000:0004
etc, so Int X 's address would be X * 4. Take a look at vector.asm
which is a small program to dump the vector table, it would be a good
idea to redirect the output to a file vector > list.txt. the program
will output the int num and its address and the position in the vector
table. we can alter this vector table with function 25 in Int 21, we
can also get vector addresses with function 35.

So with this information from here on its pretty simple, lets take
Int 05h, this is called when the print screen key is pressed, lets
alter this so it beeps

lets start off by creating our code segment

```
CODES SEGMENT
Assume CS:CODES, DS:CODES
```

since its going to be a .com we need our 100h byte PSP

```
ORG 100h ; PSP
```

next we will jump to initialising proc but we also need a variable
to hold the old vector address for Int 05 so we can call it again
in our routine, so I'll call this OLDVECT and its a dword since we
need 2 words.

```
start: JMP CODE
OLDVECT  DD ?
```

ok now for the code routine, in here we need to get the vector for Int 05
and save it, then change it to our new routine and terminate but stay
resident.

```
CODE    PROC    NEAR
Assume CS:CODES, DS:CODES

mov ah,35h  ; get vector
mov al,05h  ; get int 5
int 21h
mov word ptr OLDVECT,bx
mov word ptr OLDVECT[2],ES

mov ah,25h ; set vector
mov al,5   ; set int 5
mov dx,offset HOOK_CODE ; pointer to new proc
int 21h

mov dx,offset CODE   ; everything after this point(CODE)
INT 27h              ;  is removed

CODE    ENDP

CODES ENDS

END start
```

I've put this at the end of the program so when its terminated as a TSR
it can wipe this bit from memory and just keep the hook code at the top
which we shall do now.

In the hook code we need to be careful what we do here, first we shall call
the old int, normally when an Int is called all the flags are pushed onto

the stack but since we are using a call this will not happen so we should push the flags first, there is no need to pop them off again as the old interrupt will take care of this.

```
HOOK_CODE       PROC    FAR
Assume CS:CODES, DS:Nothing

pushf           ;emulate int call
call OLDVECT

pushf
push ax
push bx
push cx

mov al,7 ; character, 7 produces a beep
mov bh,0 ; page number
mov cx,1 ; number of bytes
mov ah,0Eh ; write text in teletype mode
int 10h

pop cx
pop bx
pop ax
popf

ret 2     ; ret but keep status flag as is
HOOK_CODE       ENDP
```

above you can see the old int is called then comes the beepy bit we'll use int 10's output a char function but output 7 which produces a tone rather than a character, you probably noticed that I didnt push all the registers but just the ones that I use, this is because we have to be careful with stack space as we don't know how much we have because we are borrowing it from somewhere else.

after our code is done and the registers are restored we exit the proc with ret 2, this keeps the status flag intact so we have simulated an Interrupt call by keeping all flags and registers as they would be after the old vector address is called.

thats pretty much the basics there is just 1 or 2 other important things, in your hook code you should never really make dos calls unless you are certain dos isn't in the middle of another function request, the above code doesn't take this into account but there are ways we can check to see if its safe, function 34 of int 21 could be used to get the dos busy flag which can be checked also we should never interrupt disk processing as this could cause
data loss so, we could hook int 13 and replace with something like


```
Int13        PROC    FAR
   mov     CS:dflag,1
   pushf
   cli
   call    CS:oldint13
   mov     CS:dflag,0
   RET     2
Int13        ENDP
```

then use this dflag to check for disk activity, cli to disable interrupts
and sti to enable would be a good idea when calling the old int code.

To compile these programs i used tasm,

```
vector = tasm vector;
         tlink vector;

hook = tasm hook;
       tlink hook;
       exe2bin hook;
       del hook.exe
       rename hook.bin hook.com
```

some versions of exe2bin might not create a bin file and name it .com already.

ok thats about it :)

---

yates@immortaldescendants.org